

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# Uso efetivo da Matemática Intervalar em Supercomputadores Vetoriais

por

*Tiarajú Asmuz Diverio*



Tese submetida como requisito parcial  
para a obtenção de grau de Doutor em  
Ciência da computação

Orientadores:

Prof Dr Dalcidio Moraes Claudio  
Prof Dr Philippe Olivier Navaux

Porto Alegre, junho de 1995.

UFRGS  
INSTITUTO DE INFORMÁTICA  
BIBLIOTECA

## CIP - Catalogação na Publicação

Diverio, Tiaraju Asmuz

Uso efetivo da Matemática Intervalar em Supercomputadores Vetoriais/ Tiarajú Asmuz Diverio - Porto Alegre: CPGCC da UFRGS, 1995.

291p.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1995. Orientadores: Claudio, Dalcidio Moraes e Navaux, Philippe Olivier Alexandre

Tese: Matemática Computacional, Processamento de alto desempenho, Aritmética computacional, Processamento vetorial.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. Dr. Hégio Casses Trindade

Pró-Reitor de Pós-Graduação: Prof. Dr. Claudio Scherer

Diretor do Instituto de Informática: Prof. Dr. Roberto Tom Price

Coordenador do CPGCC: Prof. Dr. José Palazzo Moreira de Oliveira

Bibliotecária-Chefe do Instituto de Informática: Dra. Zita Prates de Oliveira

UFRGS  
INSTITUTO DE INFORMÁTICA  
BIBLIOTECA

## AGRADECIMENTOS

As Secretarias do Instituto de Informática e do Pós-Graduação em Ciência da Computação da UFRGS;

Aos colegas professores do Instituto de Informática que sempre demonstraram confiança no meu trabalho e amizade a minha pessoa.

Aos orientadores, Profs Drs Dalcidio e Navaux, por toda paciência, orientação, amizade e pelo exemplo profissional que tanto tenho observado e seguido.

Ao Grupo de Matemática Computacional, especialmente aos meus bolsistas e amigos, por toda a dedicação, todo apoio e colaboração dado durante estes últimos anos, em especial a: Alessandra Dahmer, Úrsula Fernandes, Carlos Holbig, Ivan Tosmann e Ivan Tartaruga por toda a colaboração dada para o desenvolvimento desta pesquisa. Ao Rafael Sagula pelo comprometimento e pela ajuda na implementação das rotinas da biblioteca intervalar, ao Fabiano, Sandro, Cristiano, Charles e Christian pela colaboração nos testes e documentação, e ainda, a Betty Loeffler pelo auxílio na revisão deste material. E aos demais membros do Grupo pela amizade.

As professoras Maria Lucia Lisboa e Iara Claudio que a tantos anos me apoiam e me incentivam. Aos demais colegas que também me incentivaram nesta jornada.

Aos meus amigos, que em tantos momentos me apoiaram durante estes últimos anos, entre eles quero agradecer em especial ao Jorge e Rejane Audy, Giraffa, Pedrinho e André, Silvia Leão, Alexandre Carissimi, Vanderlei Rodrigues (que tanto suportou a minha bagunça e barulho na sala), Beatriz Franciosi, Catia e Edmeia (de Pederneiras SP).

Aos amigos de Recife e de Karlsruhe que me acolheram durante período deste trabalho, em especial ao João Fernando Marar, Débora, Ana Teresa, Alexandre Cunha, Marum, Suzana e tantos outros que me apoiaram lá.

A minha Família.

Muito Obrigado.

## SUMÁRIO

LISTA DE FIGURAS.....	8
LISTA DE FÓRMULAS.....	10
LISTA DE TABELAS.....	13
LISTA DE SÍMBOLOS.....	15
<b>RESUMO.....</b>	<b>16</b>
<b>ABSTRACT.....</b>	<b>18</b>
<b>1 INTRODUÇÃO.....</b>	<b>20</b>
1.1 Contextualização.....	20
1.2 Modelagem.....	27
1.3 O Trabalho.....	32
<b>2 ARITMÉTICA DE ALTA EXATIDÃO.....</b>	<b>35</b>
2.1 Características da Aritmética de Alta Exatidão.....	35
2.1.1 Arredondamentos Direcionados e Operações aritméticas básicas.....	36
2.1.2 Características da Matemática Intervalar.....	38
2.1.2.1 Aritmética intervalar real.....	39
2.1.2.2 Aritmética intervalar estendida.....	44
2.1.2.3 Aritmética intervalar complexa.....	45
2.1.2.4 Vetores e matrizes de intervalos.....	48
2.1.3 Produto escalar exato ou ótimo.....	49
2.2 Verificação Automática do Resultado.....	49
2.2.1 Princípios da verificação numérica.....	51
2.2.2 Aplicações da verificação automática.....	55
2.3 Linguagens para Computação Científica.....	57
2.3.1 Características.....	58
2.3.1.1 Notação matemática das expressões.....	58
2.3.1.2 Facilidades de Programação.....	58
2.3.1.3 Suporte para métodos numéricos com verificação automática do resultado.....	59
2.3.1.4 Alta exatidão.....	59
2.3.2 Disponibilidade.....	60
2.3.2.1 Pascal XSC.....	60
2.3.2.2 C XSC.....	62
2.3.2.3 Linguagens do tipo Fortran.....	63
2.4 Considerações finais.....	64
<b>3 AMBIENTES COMPUTACIONAIS.....</b>	<b>67</b>
3.1 O Cray Y MP e a linguagem Fortran 90.....	67
3.1.1 Arquitetura do Cray.....	68
3.1.2 Principais Características.....	73
3.1.3 O Compilador Fortran 90.....	75
3.1.3.1 Tipos-padrões de dados.....	76



3.1.3.2 Interfaces, Subrotinas e Funções .....	78
3.1.3.3 Módulos .....	79
3.1.3.4 Compilação e Otimização .....	80
3.1.3.5 <i>Arrays</i> e <i>Arrays</i> dinâmicos .....	80
3.2 O PC486DX e a linguagem Pascal XSC.....	81
3.2.1 Requisitos de <i>Hardware</i> .....	82
3.2.2 O Pascal XSC.....	82
3.2.2.1 Tipos-padrões de dados, operadores pré-definidos e funções .....	83
3.2.2.2 O conceito de operador geral.....	85
3.2.2.3 <i>Overloading</i> de subrotinas .....	86
3.2.2.4 O conceito de módulo.....	87
3.2.2.5 <i>Arrays</i> dinâmicos .....	88
3.2.2.6 Expressões exatas.....	89
3.2.2.7 O conceito de <i>String</i> .....	91
<b>4 RESOLUÇÃO DE SISTEMAS LINEARES.....</b>	<b>92</b>
4.1 Aplicabilidade.....	92
4.2 Formalização do problema.....	94
4.2.1 Revisão de Matrizes e Álgebra matricial.....	95
4.3 Métodos Pontuais.....	99
4.3.1 Métodos Diretos.....	99
4.3.1.1 Família dos métodos de eliminação de Gauss .....	100
4.3.1.2 Método de Gauss-Jordan .....	101
4.3.2 Métodos Iterativos .....	102
4.3.2.1 Método de Gauss-Jacobi.....	102
4.3.2.2 Método de Gauss-Seidel.....	103
4.3.2.3 Método da Sobre-relaxação .....	103
4.3.3 Métodos Compactos.....	104
4.3.3.1 Método de Banachiewicz.....	104
4.3.3.2 Método de Cholesky.....	104
4.3.3.3 Método de Cholesky para sistemas simétricos .....	105
4.4 Métodos Intervalares .....	106
4.4.1 Resolução intervalar de equações algébricas .....	106
4.4.2 Resolução intervalar de sistemas lineares .....	108
4.4.2.1 Métodos intervalares baseados em operações algébricas .....	109
4.4.2.2 Métodos intervalares baseados em refinamento .....	118
4.4.2.3 Métodos intervalares baseados em iteração .....	122
4.5 Considerações finais .....	123
<b>5 PROJETO E ESPECIFICAÇÃO DA BIBLIOTECA INTERVALAR.....</b>	<b>124</b>
5.1 Formalização da proposta.....	126
5.1.1 Justificativa e Motivação .....	128
5.1.2 Objetivos.....	129
5.1.3 Metodologia.....	130
5.2 Aritmética de Alto Desempenho .....	131

5.2.1 A Necessidade .....	131
5.2.2 Aritmética Intervalar de Máquina.....	131
5.2.3 Características do Processamento vetorial.....	133
5.3 Biblioteca de rotinas Intervalares <i>libavi.a</i> .....	135
5.3.1 Módulo de intervalos reais - <i>básico</i> .....	136
5.3.2 Módulo de vetores e matrizes de intervalos reais - <i>mvi</i> .....	140
5.3.2.1 Operações com vetores de intervalos reais .....	141
5.3.2.2 Operações com matrizes de intervalos reais.....	143
5.3.2.3 Operações de diferentes dados com vetores e matrizes de intervalos reais .....	146
5.3.3 Módulo de intervalos complexos - <i>ci</i> .....	150
5.3.4 Módulo de aplicações - <i>aplic</i> .....	154
5.3.5 Rotinas de entrada e saída de tipos intervalares .....	156
5.3.6 A Hierarquia dos módulos .....	159
5.4 A Biblioteca de rotinas científicas intervalares <i>libselint.a</i> .....	160
5.4.1 Módulo <i>dirint</i> .....	161
5.4.2 Módulo <i>refint</i> .....	161
5.4.3 Módulo <i>itrint</i> .....	161
5.4.4 Módulo <i>equalg</i> .....	161
5.5 Características e Potencialidades.....	162
5.5.1 Padrão Cray de documentação.....	164
5.5.2 Facilidade de Uso .....	166
5.5.3 Rotinas otimizadas e vetorizadas.....	170
5.5.4 Biblioteca Modular .....	170
5.5.5 Abrangência das bibliotecas .....	171
5.6 Conclusões.....	172
<b>6 COMPARAÇÃO DA <i>libavi.a</i> COM OUTRAS BIBLIOTECAS.....</b>	<b>173</b>
6.1 A <i>Numerals</i> da Burroughs .....	173
6.1.1 Módulo <i>mathlib</i> .....	174
6.1.2 Comparação da <i>libavi.a</i> com <i>numerals</i> .....	174
6.2 A Biblioteca científica da Cray <i>libsci.a</i> .....	176
6.2.1 Módulos científicos .....	177
6.2.2 Comparação da <i>libavi.a</i> com a <i>libsci.a</i> .....	181
6.3 Pascal XSC - módulos avançados .....	181
6.3.1 Módulo I_ARI (Intervalos reais).....	182
6.3.2 Módulo MV_ARI (matrizes e vetores reais) .....	184
6.3.3 Módulo MVI_ARI (matrizes e vetores de intervalos reais).....	185
6.3.4 Módulo CI_ARI (Intervalos complexos).....	187
6.3.5 Comparação da <i>libavi.a</i> com o Pascal XSC .....	189
6.4 Conclusões.....	190
<b>7 USO EFETIVO DA MATEMÁTICA INTERVALAR .....</b>	<b>191</b>
7.1 Verificação da Biblioteca de Rotinas Intervalares.....	191
7.1.1 Aritmética Básica .....	192
7.1.1.1 Adição e subtração de intervalos.....	192

7.1.1.2 Multiplicação de Intervalos .....	194
7.1.1.3 Divisão de intervalos.....	196
7.1.1.4 Funções entre conjuntos - intersecção e união.....	198
7.1.1.5 Radiciação e potenciação - funções $\text{sqr}$ e $\text{sqrt}$ .....	200
7.1.1.6 Funções trigonométricas .....	202
7.1.2 Influência do erro de arredondamento e representatividade .....	204
7.1.2.1 Influência dos erros de arredondamento.....	204
7.1.2.2 Teste de Representatividade.....	206
7.1.3 Testes do Produto escalar.....	209
7.1.4 Cálculos com matrizes de intervalos.....	212
7.1.5 Outros testes .....	214
7.2 Validação do Uso da Matemática Intervalar.....	218
7.2.1 Resolução de equação - Método de Newton Intervalar .....	218
7.2.2 Matriz de Hilbert - inversa e determinante.....	221
7.2.3 Cálculo da Matriz inversa .....	225
7.2.4 Resolução de um sistema de equações lineares.....	228
7.3 Os resultados.....	233
7.4 A avaliação do desempenho.....	234
<b>8 CONCLUSÕES E CONSIDERAÇÕES.....</b>	<b>235</b>
8.1 Resumo do trabalho.....	235
8.2 Comentário sobre resultados obtidos .....	240
8.3 Limitações do trabalho .....	241
8.4 Melhorias que podem ser feitas .....	244
8.5 Propostas para continuidade .....	244
<b>9 BIBLIOGRAFIA.....</b>	<b>246</b>
<b>10 ANEXOS .....</b>	<b>259</b>
10.1 Recomendação IMACS/GAMM.....	259
10.1.1 Escopo .....	259
10.1.2 Definição de aritmética computacional.....	259
10.1.3 Opções de implementação.....	263
10.1.4 Referência bibliografia da recomendação.....	264
10.2 Livros existentes sobre matemática intervalar e tópicos relacionados.....	267
10.3 Tabela das constantes de máquina.....	271
10.4 Extrato da listagem da <i>libavi.a</i> .....	272

## LISTA DE FIGURAS

Figura 1.1 Etapas na resolução paralela de um problema numérico.....	20
Figura 1.2 Fluxograma de uma instalação de um processo químico.....	21
Figura 1.3 Matriz de coeficientes para a instalação química.....	23
Figura 1.4 Solução do sistema de processo químico.....	23
Figura 2.1 Operações (15) fundamentais de arredondamentos.....	38
Figura 2.2 Atributos de um intervalo X.....	42
Figura 2.3 Distância de dois intervalos X e Y.....	43
Figura 2.4 Intersecção de intervalos complexos.....	46
Figura 2.5 Vetor tridimensional de intervalos reais ou caixa.....	48
Figura 2.6 Teorema de Ponto-Fixo de Brouwer.....	52
Figura 2.7 Teorema de Ponto-Fixo de Brouwer Modificado.....	53
Figura 2.8 Método de Inclusão <i>a priori</i> .....	54
Figura 2.9 Método de Inclusão <i>a posteriori</i> sem inflação.....	54
Figura 2.10 Método de Inclusão <i>a posteriori</i> com inflação $\epsilon$ .....	55
Figura 2.11 Critério de parada para provar a unicidade de uma solução.....	55
Figura 2.12 Tipos de dados disponíveis em Pascal XSC.....	59
Figura 3.1 Arquitetura do Supercomputador Cray Y-MP.....	68
Figura 3.2 Exemplo de segmentação e <i>pipelineização</i> .....	74
Figura 3.3 Operadores relacionais.....	76
Figura 3.4 Tipos de dados do Fortran 90.....	76
Figura 3.5 Sintaxe da definição de tipos de dados.....	78
Figura 3.6 Definição de intervalos reais, vetores e matrizes de intervalos.....	78
Figura 3.7 Referência aos novos tipos em Fortran 90.....	78
Figura 3.8 Função soma de intervalos.....	78
Figura 3.9 Definição da interface para soma de intervalos.....	79
Figura 3.10 Estrutura do comando de criação de módulos.....	80
Figura 3.11 Tipos de dados numéricos em Pascal XSC.....	84
Figura 3.12 Funções de transferência.....	84
Figura 3.13 Adição de intervalos por rotinas.....	85
Figura 3.14 Adição de intervalos por funções.....	85
Figura 3.15 Adição de intervalos pela definição de operadores.....	86
Figura 3.16 Exemplo Uso de complexos.....	86
Figura 3.17 Rotinas de impressão de complexos.....	87
Figura 4.1 Circuito elétrico I.....	93
Figura 4.2 Circuito elétrico II.....	94
Figura 4.3 Família dos métodos de Eliminação de Gauss.....	100
Figura 4.4 Conjunto solução X no primeiro quadrante.....	112
Figura 4.5 Conjunto solução X no segundo quadrante.....	113
Figura 4.6 Conjunto solução X no terceiro quadrante.....	113
Figura 4.7 Conjunto solução X no quarto quadrante.....	114
Figura 4.8 Conjunto solução X - área total.....	114
Figura 4.9 Ilustração do sistema e da aproximação.....	118
Figura 4.10 Dois erros na solução de um sistema.....	119
Figura 4.11 Algoritmo Resolução de SELAS intervalares.....	121



Figura 5.1 Aritmética Vetorial Intervalar .....	126
Figura 5.2 Arquivo <i>inter.inc</i> .....	137
Figura 5.3 Definição de intervalos complexos arquivo <i>inter.inc</i> .....	150
Figura 5.4 Definição de vetores e matrizes de intervalos complexos .....	151
Figura 5.5 Hierarquia dos módulos da biblioteca <i>libavi.a</i> .....	159
Figura 5.6 Formato da documentação detalhada das rotinas.....	165
Figura 6.1 Convenção utilizada na composição dos mnemônicos .....	173
Figura 6.2 Hierarquia dos tipos aritméticos .....	182
Figura 7.1 Programa para a adição e subtração de intervalos em Fortran 90 .....	193
Figura 7.2 Programa para a adição e subtração de intervalos em Pascal XSC .....	193
Figura 7.3 Nove casos da definição de produto de intervalos.....	194
Figura 7.4 Programa do produto de intervalos em Fortran 90 .....	195
Figura 7.5 Programa do produto de intervalos em Pascal XSC .....	195
Figura 7.6 Intervalos utilizados na divisão.....	196
Figura 7.7 Intervalos utilizados na divisão como divisores (zero não está incluído).....	196
Figura 7.8 Programa da divisão em Pascal XSC.....	197
Figura 7.9 Programa da divisão em Fortran 90.....	197
Figura 7.10 Programa de união e intersecção de intervalos em Pascal XSC.....	198
Figura 7.11 Programa de união e intersecção de intervalos em Fortran 90.....	199
Figura 7.12 Programa teste das funções <i>sqr</i> e <i>sqrt</i> em Pascal XSC .....	200
Figura 7.13 Programa teste das funções <i>ssqr</i> e <i>ssqrt</i> em Fortran 90.....	201
Figura 7.14 Programa teste das funções trigonométricas intervalares em Pascal XSC .....	202
Figura 7.15 Programa teste das funções trigonométricas intervalares em Fortran 90.....	203
Figura 7.16 Programa e execução do teste da influência dos erros de arredondamento em Pascal XSC	205
Figura 7.17 Programa e execução do teste da influência dos erros de arredondamento em Fortran 90	206
Figura 7.18 Programa e execução do teste da representatividade em Pascal XSC .....	207
Figura 7.19 Programa e execução do teste da representatividade em Fortran 90 .....	208
Figura 7.20 Programa e execução do produto escalar em Pascal XSC .....	210
Figura 7.21 Programa e execução do produto escalar em Fortran 90.....	211
Figura 7.22 Programa de cálculo de matrizes em Pascal XSC.....	213
Figura 7.23 Programa e execução em Fortran 90 de operações com matrizes .....	214
Figura 7.24 Testes de somatórios em Pascal XSC .....	216
Figura 7.25 Testes de somatórios em Fortran 90 .....	217
Figura 7.26 Programa e execução do método de Newton intervalar simplificado em Pascal XSC.....	219
Figura 7.27 Programa e execução do método de Newton intervalar simplificado em Fortran 90 .....	220
Figura 7.28 Programa e execução da matriz de Hilbert - determinante .....	223
Figura 7.29 Procedure <i>Matinv</i> .....	226
Figura 7.30 Módulo <i>Iseles</i> .....	227
Figura 7.31 Rotina <i>Checkzeros</i> .....	228
Figura 7.32 Função <i>accurate</i> .....	229
Figura 7.33 Rotina de verificação .....	229
Figura 7.34 Rotina <i>linsolve</i> .....	230
Figura 7.35 Programa <i>Hansen3</i> .....	232

## LISTA DE FÓRMULAS

Fórmula 1.1 Números em ponto-flutuante .....	25
Fórmula 1.2 Números em ponto-flutuante .....	25
Fórmula 1.3 Arredondamento para baixo.....	27
Fórmula 1.4 Arredondamento para cima.....	27
Fórmula 1.5 Arredondamento simétrico .....	27
Fórmula 2.1 Regra geral (RG) .....	37
Fórmula 2.2 Definição de arredondamento .....	37
Fórmula 2.3 Propriedade monotônica .....	37
Fórmula 2.4 Propriedade anti-simétrica.....	37
Fórmula 2.5 Propriedade de intervalos.....	37
Fórmula 2.6 Definição de arredondamentos direcionados.....	37
Fórmula 2.7 Operações com arredondamento direcionado .....	38
Fórmula 2.8 Definições de intervalos .....	39
Fórmula 2.9 Conjunto de intervalos .....	40
Fórmula 2.10 Definições das operações entre intervalos.....	40
Fórmula 2.11 Soma intervalar.....	40
Fórmula 2.12 Subtração de intervalos.....	40
Fórmula 2.13 Multiplicação intervalar.....	40
Fórmula 2.14 Divisão intervalar.....	40
Fórmula 2.15 Relação: ínfimo, médio e supremo do intervalo .....	42
Fórmula 2.16 Limite intervalar.....	43
Fórmula 2.17 Extensão intervalar de funções .....	43
Fórmula 2.18 Imagem da função.....	44
Fórmula 2.19 Propriedade semi-distributiva .....	44
Fórmula 2.20 Conjunto dos reais estendido .....	44
Fórmula 2.21 Conjunto dos intervalos reais estendidos .....	44
Fórmula 2.22 Divisão de intervalos estendidos.....	45
Fórmula 2.23 Subtração x-Y estendida .....	45
Fórmula 2.24 Intervalo complexo .....	45
Fórmula 2.25 Relações $=, \subset, \subseteq$ para intervalos complexos.....	46
Fórmula 2.26 Relações $\cap, \cup$ para intervalos complexos .....	46
Fórmula 2.27 Adição de intervalos complexos .....	46
Fórmula 2.28 Subtração de intervalos complexos .....	46
Fórmula 2.29 Multiplicação de intervalos complexos .....	46
Fórmula 2.30 Divisão de intervalos complexos .....	46
Fórmula 2.31 Vetores de intervalos .....	47
Fórmula 2.32 Matrizes de intervalos .....	47
Fórmula 2.33 Norma máxima de vetor de intervalos .....	48
Fórmula 2.34 Norma máxima de matriz de intervalos.....	48
Fórmula 2.35 Diâmetro máximo absoluto de vetor de intervalos .....	48
Fórmula 2.36, Diâmetro relativo de vetor de intervalos.....	48
Fórmula 2.37 Definição de produto escalar.....	49
Fórmula 2.38 Fórmula de Iteração.....	53



Formula 2.39 Fórmula de intersecção sucessiva .....	53
Formula 2.40 Fórmula de iteração inflada .....	54
Formula 2.41 Fórmula de iteração com critério de parada .....	55
Fórmula 2.42 Multiplicação de matrizes por comandos .....	57
Fórmula 4.1 Sistemas de Equações Lineares .....	94
Fórmula 4.2 Sistemas de Equações - forma matricial .....	95
Fórmula 4.3 Fórmula das operações elementares .....	101
Fórmula 4.4 Fórmula da determinação dos pivos .....	101
Fórmula 4.5 Cálculo do X por Gauss-Jordan .....	102
Fórmula 4.6 Fórmula da sobre-relaxação .....	103
Fórmula 4.7 Matrizes Triangulares Inferior (L) e Superior (U).....	104
Fórmula 4.8 Matriz superior do método de Cholesky .....	105
Fórmula 4.9 Matriz inferior do método de Cholesky simétrico .....	105
Fórmula 4.10 Método de Newton-Raphson .....	107
Fórmula 4.11 Método de Newton-Intervalar Ingênuo .....	107
Fórmula 4.12 Operador Newtoniano .....	108
Fórmula 4.13 Método de Newton intervalar simplificado .....	108
Fórmula 4.14 Método de Hansen - fórmula geral .....	111
Fórmula 4.15 Método de Hansen - pontos do I quadrante.....	112
Fórmula 4.16 Método de Hansen - Inequações .....	112
Fórmula 4.17 Método de Gauss - MEGI - Matriz concatenada ao vetor .....	115
Fórmula 4.18 Método de Gauss - MEGI - Operações elementares - 1 passo .....	115
Fórmula 4.19 MEGI - Matriz concatenada transformada - 1 coluna zerada .....	116
Fórmula 4.20 Método de Gauss - MEGI - Operações elementares - j-ésimo passo .....	116
Fórmula 4.21 MEGI - Matriz concatenada transformada triangular superior .....	116
Fórmula 4.22 MEGI - Fórmulas da retrossubstituição.....	116
Fórmula 4.23 Cálculo da aproximação da inversa - fórmulas .....	117
Fórmula 4.24 Triangularização inferior - substituição progressiva.....	117
Fórmula 4.25 Triangularização superior - retrossubstituição .....	118
Fórmula 4.26 Esquema de iteração de Newton com inversa .....	120
Fórmula 4.27 Esquema de iteração com a aproximação R.....	120
Fórmula 4.28 Esquema de iteração modificada .....	120
Fórmula 4.29 Refinamento da solução .....	120
Fórmula 4.30 Cálculo da inversa por decomposição.....	121
Fórmula 4.31 Condição básica .....	122
Fórmula 4.32 Condição básica transformada com decomposição.....	122
Fórmula 4.33 Fórmula da iteração básica.....	122
Fórmula 4.34 Método de Jacobi .....	122
Fórmula 4.35 Método de Gauss-Seidel.....	122
Fórmula 4.36 Método da Sobre-relaxação .....	122

Fórmula 5.1 Sintaxe do comando <i>open</i> .....	156
Fórmula 5.2 Exemplo do uso do comando <i>read</i> .....	156
Fórmula 5.3 Referência do uso da rotina <i>sread</i> .....	156
Fórmula 5.4 Referência do uso da rotina <i>svread</i> .....	157
Fórmula 5.5 Exemplo de declaração de vetor de intervalos .....	157
Fórmula 5.6 Referência do uso da rotina <i>smread</i> .....	157
Fórmula 5.7 Exemplo de declaração de matriz de intervalos .....	157
Fórmula 5.8 Referência do uso da rotina <i>scread</i> .....	157
Fórmula 5.9 Exemplo do uso do comando <i>write</i> .....	157
Fórmula 5.10 Referência do uso da rotina <i>swrite</i> .....	158
Fórmula 5.11 Referência do uso da rotina <i>svwrite</i> .....	158
Fórmula 5.12 Referência do uso da rotina <i>smwrite</i> .....	158
Fórmula 5.13 Referência do uso da rotina <i>scwrite</i> .....	158
Fórmula 6.1 Formato de leitura de intervalos complexos .....	188
Fórmula 7.1 Vetores do teste do produto escalar .....	210
Fórmula 7.2 Matrizes do teste de matrizes .....	212
Fórmula 7.3 Método de Newton-Raphson (real) .....	218
Fórmula 7.4 Método de Newton Intervalar Ingênuo .....	218
Fórmula 7.5 Operador Newtoniano .....	218
Fórmula 7.6 Método de Newton Intervalar Simplificado .....	219
Fórmula 7.7 Definição da matriz de Hilbert de ordem <i>n</i> .....	221
Fórmula 7.8 Determinante da matriz de Hilbert de ordem <i>n</i> .....	221
Fórmula 7.9 Sistema de Boothroyd/Dekker .....	229
Fórmula 10.1 Projeção para reais .....	260
Fórmula 10.2 Monotonicidade para reais .....	260
Fórmula 10.3 Anti-simetria nos reais .....	260
Fórmula 10.4 Arredondamento direcionado .....	260
Fórmula 10.5 Projeção em um espaço geral .....	260
Fórmula 10.6 Monotonicidade em um espaço geral .....	260
Fórmula 10.7 Anti-simetria em um espaço geral .....	260
Fórmula 10.8 Inclusão .....	261
Fórmula 10.9 Operação Arredondada .....	261
Fórmula 10.10 Produto escalar exato .....	262

## LISTAS DE TABELAS

Tabela 2.1 Multiplicação de intervalos .....	41
Tabela 2.2 Divisão de intervalos, $0 \notin Y$ .....	41
Tabela 3.1 Tabela dos modos de arredondamento para expressões exatas .....	90
Tabela 5.1 Funções de Transferência de intervalos.....	137
Tabela 5.2 Operadores relacionais entre intervalos reais.....	138
Tabela 5.3 Operações entre conjuntos reais .....	138
Tabela 5.4 Operações aritméticas entre intervalos reais .....	139
Tabela 5.5 Funções elementares intervalares .....	139
Tabela 5.6 Funções de Transferência de Vetores de Intervalos.....	141
Tabela 5.7 Operadores relacionais entre vetores de intervalos .....	142
Tabela 5.8 Operações entre conjuntos de vetores de intervalos reais .....	142
Tabela 5.9 Operações aritméticas entre vetores de intervalos reais .....	142
Tabela 5.10 Funções Básicas de Vetores de Intervalos.....	143
Tabela 5.11 Funções de Transferência de Matrizes de Intervalos.....	144
Tabela 5.12 Operadores relacionais entre matrizes de intervalos .....	144
Tabela 5.13 Operações entre conjuntos de matrizes de intervalos reais.....	145
Tabela 5.14 Operações aritméticas entre matrizes de intervalos reais.....	145
Tabela 5.15 Funções Básicas de Matrizes de Intervalos .....	145
Tabela 5.16 Tabela das operações aritméticas pré-definidas .....	146
Tabela 5.17 Operações aritméticas de reais com intervalos .....	147
Tabela 5.18 Operações aritméticas de reais com vetores de intervalos.....	147
Tabela 5.19 Operações aritméticas de reais com matrizes de intervalos .....	148
Tabela 5.20 Operações aritméticas de intervalos com vetores de intervalos.....	148
Tabela 5.21 Operações aritméticas de intervalos com matrizes de intervalos .....	149
Tabela 5.22 Operações aritméticas entre vetores e matrizes de intervalos.....	149
Tabela 5.23 Operações entre vetores e matrizes reais e intervalares .....	150
Tabela 5.24 Funções de Transferência de Intervalos Complexos .....	152
Tabela 5.25 Operadores relacionais entre intervalos Complexos.....	153
Tabela 5.26 Operações entre conjuntos de intervalos complexos.....	153
Tabela 5.27 Operações aritméticas entre intervalos complexos.....	153
Tabela 5.28 Funções Elementares Intervalares Complexas .....	154
Tabela 5.29 Rotinas entre matrizes e vetores da álgebra linear .....	155
Tabela 5.30 Sobrecarga dos operadores = e + .....	167
Tabela 5.31 Sobrecarga dos operadores aritméticos -, * e / .....	168
Tabela 5.32 Sobrecarga dos operadores relacionais .....	169
Tabela 5.33 Sobrecarga das funções trigonométricas .....	169
Tabela 5.34 Sobrecarga das funções exponenciais e logarítmicas .....	169
Tabela 5.35 Sobrecarga das demais funções .....	170

Tabela 6.1 Rotinas entre matrizes da <i>Numerals</i> .....	174
Tabela 6.2 Rotinas de resolução de sistemas da <i>Numerals</i> .....	175
Tabela 6.3 Rotinas reais da BLAS nível 1 .....	177
Tabela 6.4 Rotinas reais da BLAS nível 2 .....	178
Tabela 6.5 Rotinas reais da BLAS nível 3 .....	178
Tabela 6.6 Rotinas de resolução de sistemas lineares densos - LAPACK.....	179
Tabela 6.7 Estrutura do resultado das operações com matrizes e vetores .....	182
Tabela 6.8 Operadores do módulo I_ARI .....	182
Tabela 6.9 Funções pré-definidas do módulo I_ARI .....	183
Tabela 6.10 Funções de transferência do módulo I_ARI .....	183
Tabela 6.11 Operadores do módulo MV_ARI .....	184
Tabela 6.12 Funções pré-definidas do módulo MV_ARI.....	184
Tabela 6.13 Operadores do módulo MVI_ARI .....	185
Tabela 6.14 Funções de transferência do módulo MVI_ARI .....	186
Tabela 6.15 Funções pré-definidas do módulo MVI_ARI .....	186
Tabela 6.16 Funções de transferência do módulo CI_ARI.....	187
Tabela 6.17 Funções pré-definidas do módulo CI_ARI .....	188
Tabela 7.1 Tabulação dos resultados do Cray para o programa do exemplo 7.1.5 .....	215
Tabela 7.2 Tabulação dos resultados em Pascal XSC para o programa do exemplo 7.1.5 ..	215
Tabela 7.3 Cálculo do determinante da matriz de Hilbert em Fortran 90.....	224
Tabela 7.3 Testes do condicionamento com matrizes de Hilbert em Fortran 90 .....	224

## LISTA DE SÍMBOLOS

$\Delta x$	Arredondamento para cima
$\nabla x$	Arredondamento para baixo
$Ox$	Arredondamento simétrico
$[]x$	Arredondamento geral
$\#$	Operação genérica
$+$	Operação de adição
$-$	Operação de subtração
$*$	Operação de multiplicação
$/$	Operação de divisão
$:$	Operação de divisão
$\Sigma$	Somatório
$b^e$	Potenciação: b elevado a e
$\cdot$	Produto escalar
$[\#]$	Operação arredondada, definida segundo RG
$\Delta\#$	Operação arredondada para cima
$\nabla\#$	Operação arredondada para baixo
$-\infty$	Menos infinito
$+\infty$	Mais infinito
$\leq$	Menor ou igual
$\geq$	Maior ou igual
$\subseteq$	Contido ou igual
$\in$	Relação de pertinência: pertence
$\forall$	Quantificador Universal: Para todo
$\exists$	Quantificador existencial: Existe
$\cap$	Intersecção
$\cup$	União
(RG)	Regra geral de definição de operações
$X=[x1, x2]$	Um intervalo cujos extremos são a e b
$\emptyset$	Conjunto vazio
<b>F</b>	Conjunto dos números em ponto-flutuante
<b>R</b>	Conjunto dos Números reais
<b>C</b>	Conjunto dos Números complexos
<b>IR</b>	Conjunto dos intervalos reais
<b>IF</b>	Conjunto dos intervalos em ponto-flutuante
<b>IC</b>	Conjunto dos intervalos complexos
<b>VC, MC</b>	Conjuntos dos vetores e matrizes de <b>C</b>



## RESUMO:

Este trabalho apresenta um estudo do uso da Matemática Intervalar na resolução de problemas em supercomputadores, através da biblioteca de rotinas intervalares denominada *libavi.a* (aritmética vetorial intervalar), proporcionando não só aumento de velocidade de processamento via vetorização, mas exatidão e controle de erros nos cálculos através do emprego da aritmética intervalar.

Foram identificadas duas das barreiras que a resolução de problemas numéricos em computadores enfrenta. Estas barreiras se referem a qualidade do resultado e ao porte do problema a ser resolvido. Verificou-se a existência de uma grande lacuna entre o avanço tecnológico, incluindo o desenvolvimento de computadores cada vez mais rápidos, e poderosos e a qualidade com que os cálculos são feitos. Através dos supercomputadores (geralmente computadores vetoriais e/ou paralelos), os resultados são obtidos com extrema rapidez, mas nem sempre se sabe quão confiáveis realmente são.

Como a definição da aritmética da máquina ficava a cargo do fabricante, cada sistema tinha as suas próprias características e defeitos. Cálculos efetuados em diferentes máquinas raramente produziam resultados compatíveis. Então, em 1980, a IEEE adotou o padrão de aritmética binária de ponto-flutuante, conhecida como padrão IEEE 754. Isto foi um passo no sentido de se resolver a questão de qualidade numérica dos resultados, mas este padrão não especificou tudo.

A pesquisa evoluiu para a proposta de uma aritmética de alta exatidão e alto desempenho, que torne disponível operações com intervalos e a própria matemática intervalar aos usuários do supercomputador vetorial Cray Y-MP2E. Como protótipo desta aritmética de alto desempenho, foi desenvolvido um estudo, uma especificação e, posteriormente, implementada uma biblioteca de rotinas intervalares no supercomputador Cray Y-MP2E, denominada *libavi.a*. O nome *libavi.a* significa biblioteca (*lib*) composta da aritmética vetorial intervalar (*avi*). O sufixo *.a* é o sufixo padrão de bibliotecas no Cray.

Com a *libavi.a* definiu-se a aritmética de alto desempenho, composta do processamento de alto desempenho (vetorial) e da matemática intervalar. Não se tem a aritmética de alta exatidão e alto desempenho, pois no ambiente vetorial, como do supercomputador Cray Y-MP2E com a linguagem de programação Fortran 90, a aritmética não segue o padrão da IEEE 754 na especificação do tamanho da palavra nem na forma como os arredondamentos e operações aritméticas em ponto-flutuante são efetuadas. Foi necessário desenvolver rotinas que simulassem os arredondamentos direcionados e operações em ponto-flutuante com controle de erro de arredondamento.



A biblioteca *libavi.a* é um conjunto de rotinas intervalares que reúne as características da matemática intervalar no ambiente do supercomputador vetorial Cray Y-MP. A *libavi.a* foi desenvolvida em Fortran 90, o que possibilitou as características de modularidade, sobrecarga de operadores e funções, uso de *arrays* dinâmicos na definição de vetores e matrizes e a definição de novos tipos de dados próprios à análise matemática.

A biblioteca foi organizada em quatro módulos: *básico* (com 52 rotinas que implementam intervalos reais), *mvi* (com 151 rotinas sobre matrizes e vetores de intervalos reais), *aplic* (com 29 rotinas intervalares sobre aplicações da álgebra linear) e *ci* (com 58 rotinas que implementam intervalos complexos). O módulo *básico* contém a aritmética intervalar básica, sendo, por isso, utilizado por todos os demais. O módulo *aplic* contém os demais módulos, pois ele se utiliza deles. O módulo de intervalos complexos, contém o módulo *básico*.

Além da aritmética vetorial intervalar (operações, funções e avaliação de expressões), sentiu-se a necessidade de providenciar bibliotecas que tornassem disponíveis os métodos intervalares para usuários do Cray (na resolução de problemas). Inicialmente foi especificada a biblioteca científica aplicada *libselint.a*, composta por algumas rotinas intervalares de resolução de equações algébricas e sistemas de equações lineares. Observa-se que desta biblioteca aplicada foram implementadas apenas algumas rotinas visando verificar e validar o uso da biblioteca intervalar e da matemática intervalar em supercomputadores.

Por fim, foram desenvolvidos vários testes que verificaram a biblioteca de rotinas intervalares quanto a sua correção e compatibilidade com a documentação. Todos os resultados obtidos através de programas que utilizavam a *libavi.a* foram comparados com os resultados produzidos por programas análogos em Pascal XSC.

A validação do uso da Matemática Intervalar no supercomputador vetorial se deu através da resolução de problemas numéricos implementados em Fortran 90, utilizando a *libavi.a*, e seus resultados foram confrontados com o de outras bibliotecas.

## **PALAVRAS CHAVES:**

Aritmética de Alto Desempenho, Vetorização de Algoritmos, Processamento Vetorial, Aritmética Intervalar, Intervalos, *libavi.a* - Aritmética Vetorial Intervalar.

**ABSTRACT:**

**TITLE:** "EFFECTIVE USE OF INTERVAL MATHEMATICS ON VECTOR SUPERCOMPUTERS"

In this study a practical use of Interval Mathematics, for the resolution of numerical problems, through a new tool, *libavi.a* (Vector and Interval Arithmetic Library) is presented. A new tool for resolution of numerical problems in supercomputers is proposed, providing increase in processing speed through vectorization and adding accuracy and error control at the performance of interval arithmetic.

Two limitations of numerical problems resolution in computers were identified. These limitations are related to the quality of results and the size of the problem to be solved. A big distance between technology improvement, including development of more powerful and faster computers, and the quality of calculus performance is the consequence of this progress. Among supercomputers (vectorial and parallel computers) the results are quickly obtained, but we may not know how exact they are.

Since the definition of machine arithmetic was in charge of makers, each system has its own characteristics and problems. Compatible or equal results are rarely produced when calculus are made in different machines. Then in 1980, the IEEE adopted the pattern of binary floating-point arithmetic, known as pattern IEEE754. This was one step in the correct direction for solving the matter of results numerical quality. Anyway this pattern was incomplete.

Research has come to a development proposal of a high accuracy and high performance arithmetic, which supports interval operations and interval mathematics itself for the user of Cray supercomputer. A study and specification were developed as a prototype application of this definition of high performance arithmetic. Later also a design and implementation of the library of interval routines programmed in FORTRAN 90 were made on Cray Y-MP supercomputer environment, called *libavi.a*. The name *libavi.a* means library (*lib*) composed of vector interval arithmetic (*avi*, in Portuguese). The suffix *.a* is the suffix of libraries on Cray.

High performance arithmetic was defined for *libavi.a*, which is composed of high performance processing and interval mathematics. The high accuracy and high performance arithmetic was not possible because, on Cray Y-MP supercomputer environment with the programming language FORTRAN 90, the native arithmetic is not according to the pattern of IEEE 754. The specification of the word size, the way that the arithmetic operations in floating-point are made and the kind of roundings are different from the pattern. It was necessary to simulate these operations and roundings.

The library *libavi* is a set of interval routines that meets characteristics of interval mathematics in the environment of vector supercomputer Cray Y-MP. It was developed in

FORTRAN 90, making available some characteristics as modularity, overloading of operators and functions, the use of dynamic arrays in the definition of vectors and matrix and the definition of new kinds of data from analysis mathematics.

It was organized in four modules: *basic* (with 52 routines of real intervals), *mvi* (with 151 routines over real interval matrix and vectors), *aplic* (with 29 routines over linear algebra) and *ci* (with 58 routines of complex intervals). The *basic* module contains the basic interval arithmetic and therefore it is used by all other modules. The *aplic* module contains the three other modules, because it uses their routines. Then the complex interval module contains the *basic* module.

Finally, some tests are made to verify the correctness of interval routines library and compatibility with its documentation. All the results from FORTRAN and Pascal XSC programs for the same problems were compared.

The validation of interval mathematics use on Cray supercomputer was made through the resolution of numerical problems programmed in FORTRAN 90, using the library *libavi* and the results was compared with other libraries.

#### **KEY WORDS:**

High Performance Arithmetic, Vector Processing, Algorithms Vectorization, Intervals, Interval Arithmetic, Libavi library.

# 1 INTRODUÇÃO

## 1.1 Contextualização

Muitos dos problemas das engenharias e das ciências precisam ser modelados matematicamente para serem resolvidos. Para isso é necessário que se projetem algoritmos numéricos eficientes. Em outras palavras, tais problemas são resolvidos via matemática simbólica, gráfica, numérica, etc... A matemática numérica tem por objetivo estudar processos numéricos (algoritmos) para solução de problemas, visando a máxima economia e confiabilidade em termos dos fatores envolvidos, como por exemplo: tempo de execução, memória utilizada, exatidão, etc.

O problema de tradução do algoritmo computacional em uma implementação física não é, em geral, trivial, principalmente quando se está num ambiente de processamento paralelo. Para este trabalho devem-se envolver: conhecimentos da teoria da computação, uma representação implícita ou explícita do algoritmo correspondente à aplicação, uma arquitetura computacional abstrata e uma implementação física. Do relacionamento destes itens, podem-se identificar os níveis de abstração na resolução de um problema numérico, conforme a figura 1.1, onde temos: o problema no mundo real, o modelo matemático que representa o problema, o algoritmo numérico que esboça a resolução do problema matemático e o programa em uma linguagem de programação que é a implementação do algoritmo. Esta implementação, em computadores de alto desempenho, pode envolver operações vetoriais (processamento vetorial) ou tarefas sendo executadas em paralelo (processamento paralelo).

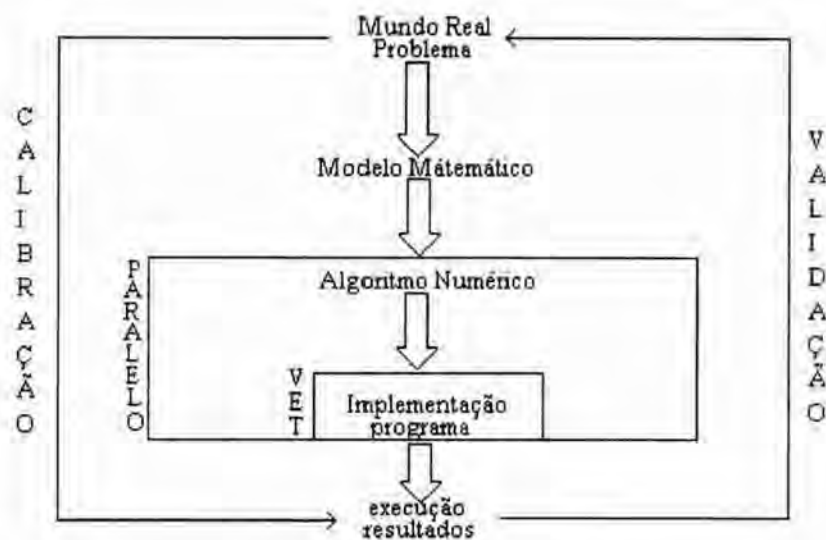


Fig.1.1 Etapas na resolução paralela de um problema numérico



A resolução de problemas numéricos em computadores enfrenta pelo menos duas grandes barreiras, decorrentes das limitações da máquina. A primeira refere-se à qualidade do resultado, que como será visto, depende de fatores que vão desde a forma como os números são representados e armazenados até a instabilidade do próprio problema e do algoritmo. A segunda barreira diz respeito ao porte do problema, que é limitado pelo número de elementos ou parâmetros (dimensão do problema) que podem ser armazenados (espaço - memória) e manipulados - quantidade de operações que envolvem tempo. Portanto o tempo de processamento e o espaço de armazenagem limitam a dimensão do problema a ser resolvido.

Para ilustrar a resolução de problemas, será considerado um exemplo hipotético da química<sup>1</sup>. A figura 1.2 representa um processo contínuo de uma instalação química hipotética. A alimentação ou entrada é designada pelo fluxo 1, que consiste de 40Kg/h de um composto chamado A, 40Kg/h do composto B e 20Kg/h do composto C. A função da instalação é converter parte de A e parte de B em C e separar parcialmente os componentes de modo que o produto consista principalmente de C. A situação idealizada ilustra processos usados na química e proporciona um bom exemplo de como os sistemas de equações aparecem no trabalho de projeto industrial prático, ou seja, através da modelagem.

A separação destes três componentes é baseada nas diferenças de seus pontos de ebulição: C tem um ponto de ebulição mais baixo que A e B. Quando uma mistura destes três compostos entra em ebulição, o vapor contém maior proporção de C que o líquido. Do mesmo modo, quando um recipiente contendo os três componentes nos estados líquido e vapor é esfriado, o líquido tem uma maior fração de A e B que o vapor. Com isto estabelecido, pode-se descrever a operação de instalação e escrever as equações que a governam.

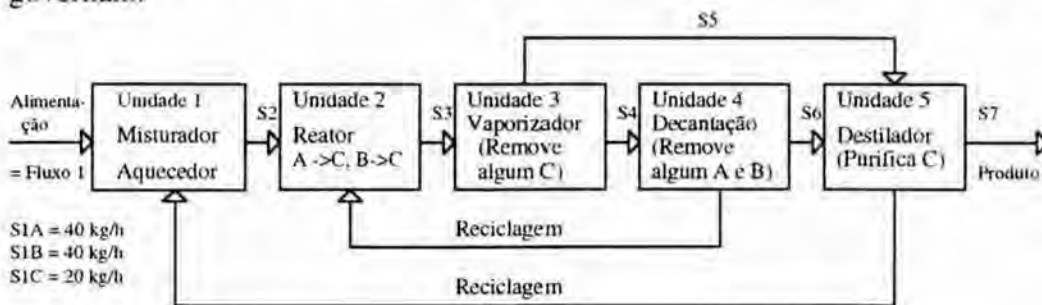


Fig 1.2 Fluxograma de uma instalação de um processo químico

A unidade 1 é um aquecedor-misturador. O aquecimento é necessário para se levar a mistura ao ponto de operação do reator para o qual ela irá em seguida. A mistura envolve o fluxo de entrada (conhecido) e o material que está sendo devolvido ou reciclado na unidade 5. Se os Kg/h de A, B e C no fluxo 2 forem chamados por S2A, S2B e S2C e assim por diante, pode-se escrever três equações que descrevam a saída da unidade 1. Para cada componente que deixa a unidade, sabe-se que é simplesmente a soma das duas

<sup>1</sup>Retirado do livro [DOR81] Cálculo Numérico com estudos de casos em Fortran IV.

entradas. O fluxo de alimentação é conhecido e os componentes do fluxo 9 são três variáveis que se deseja determinar. Assim, têm-se as primeiras equações:

$$\begin{aligned} S2A &= S9A + 40 \\ S2B &= S9B + 40 \\ S2C &= S9C + 20 \end{aligned}$$

A unidade 2 é um reator. Sua função é converter parte do A e do que entra em C. Numericamente, ele converte metade do B que entra em C e 35/90 do A que entra em C. Em ambos os casos, o material que é agora convertido em C simplesmente deixa a unidade inalterado. Visto que a entrada nesta unidade consiste na soma dos fluxos 2 e 8, podem-se escrever as equações do fluxo 3 da seguinte forma:

$$\begin{aligned} S3A &= 55/90(S2A + S8A) \\ S3B &= 0.5(S2B + S8B) \\ S3C &= 0.5(S2B + S8B) + 35/90(S2A + S8A) + S2C + S8C \end{aligned}$$

A unidade 3 é chamada de vaporizador. Calor é adicionado causando a vaporização de parte da mistura na unidade; o vapor será relativamente rico em C uma vez que seu ponto de ebulição é mais baixo do que os de A e B. Esta porção rica em C é enviada adiante para a unidade 5. Desta vez os dados numéricos estão em forma de frações, uma para cada concentração dos três componentes nos dois fluxos de saída, estas são:  $S5A/S4A = 0.1$   $S5B/S4B = 1/6$   $S5C/S4C = 6/5$ .

Sabe-se que a soma dos fluxos 4 e 5 deve ser igual ao fluxo 3, ou seja,  $S3A = S4A + S5A$ , e assim por diante. Combinando este fato com as relações dadas, chega-se às equações para os fluxos 4 e 5:

$$\begin{aligned} S4A &= 10/11 S3A & S5A &= 1/11 S3A \\ S4B &= 6/7 S3B & S5B &= 1/7 S3B \\ S4C &= 5/11 S3C & S5C &= 6/11 S3C \end{aligned}$$

A unidade 4 é chamada unidade de decantação. Decantar significa retirar. A mistura de entrada é esfriada, levando a uma fase líquida que tem uma alta proporção de A e B. Parte deste líquido rico em A e B é levado de volta (reciclado) à unidade 2. O processo na unidade 4 é assim algo similar àquele da unidade 3, mas com diferentes frações,  $S6A/S8A = 1.5$   $S6B/S8B = 2$   $S6C/S8C = 4$ .

Combinando estas relações com o fato de que a soma dos fluxos 6 e 8 é igual ao fluxo 4, têm-se as seguintes equações:

$$\begin{aligned} S6A &= 3/5 S4A & S8A &= 2/5 S4A \\ S6B &= 2/3 S4B & S8B &= 1/3 S4B \\ S6C &= 4/5 S4C & S8C &= 1/5 S4C \end{aligned}$$

A unidade 5 é um destilador. Pode-se imaginar sua operação como similar ao vaporizador da unidade 3, um processo de vaporização e condensação leva a uma saída que contém uma proporção de C maior do que a da entrada. Esta fração rica em C, fluxo 7, é o produto. O resto é reciclado de volta para a unidade 1. As frações são:  $S7A/S9A = 1/6$ ,  $S7B/S9B = 1/4$  e  $S7C/S9C = 9$ . Combinando estas relações com o fato de que a soma dos fluxos 7 e 9 é igual à soma dos fluxos 5 e 6, obtêm-se as equações finais:



$$\begin{aligned}
 S7A &= 1/7 (S5A + S6A) & S9A &= 6/7 (S5A + S6A) \\
 S7B &= 1/5 (S5B + S6B) & S9B &= 4/5 (S5B + S6B) \\
 S7C &= 9/10 (S5C + S6C) & S9C &= 1/10 (S5C + S6C)
 \end{aligned}$$

Têm-se 24 equações com 24 variáveis, uma variável para cada um dos três componentes em cada um dos oito fluxos que não os de entrada. Para encontrar os valores das variáveis, deve-se montar o sistema de equações e resolvê-lo. A figura 1.3 consta da matriz dos coeficientes.

i \ j	S2			S3			S4			S5			S6			S7			S8			S9			25	
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C		
1	1																								40	
2		1																								40
3			1																							20
4	-35900			1																						
5		-1/2			1																					
6	-35900	-1/2	-1			1																				
7							1																			
8								1																		
9									1																	
10										1																
11											1															
12												1														
13													1													
14														1												
15															1											
16																1										
17																	1									
18																		1								
19																			1							
20																				1						
21																					1					
22																						1				
23																							1			
24																									1	

Fig.1.3 Matriz de coeficientes para a instalação química

Resolvendo este sistema, obtêm-se como solução os valores da figura 1.4, em uma máquina com quatro dígitos decimais depois da vírgula. Note que os valores estão agrupados em oito linhas de três elementos, correspondendo aos 8 fluxos dos três componentes.

70.0003	60.0000	30.0000
55.0002	35.0000	110.0001
50.0002	30.0000	50.0000
5.0000	5.0000	60.0000
30.0001	20.0000	40.0000
<b>4.9998</b>	<b>5.0000</b>	<b>90.0000</b>
20.0001	10.0000	10.0000
30.0003	20.0000	10.0000

Fig 1.4 Solução do sistema do Processo químico

Observa-se que os 100 Kg/h de entrada (fluxo 1) consistindo de 40 Kg/h de A, 40Kg/h de B e 20Kg/h de C, foram convertidos em 100Kg/h de saída no fluxo 7 ( o que deve ser verdadeiro se as equações estavam corretas), mas as quantidades de A, B e C agora são 5, 5 e 90, respectivamente. Observa-se ainda que uma instalação química real envolveria muitos outros processos e componentes, levando a sistemas de centenas de equações, talvez na mesma quantidade de variáveis e ocorreriam erros resultantes de arredondamentos.

Para resolver este sistema de equações, que é da ordem de 24 equações por 24 variáveis pelo método de Cramer, também conhecido como o método dos determinantes, seriam necessárias mais de  $(n+1)!$  operações aritméticas. Se cada uma destas operações fosse efetuada em um décimo de microsegundo ( $10^{-5}$ ) seriam necessários milhões de séculos para se calcular a solução. Já utilizando o método de Eliminação de Gauss, onde o número de operações aritméticas necessárias é da ordem polinomial, ou seja da ordem de  $n^3$  multiplicações e adições, o problema seria resolvido em poucos instantes.

Tanto o método de Cramer quanto o método de Eliminação de Gauss necessitam armazenar toda a matriz, cuja dimensão, no caso, é  $n=24$ , tendo-se, então, 576 elementos. Isto resulta em um *Megaword* de memória no Cray. Isto pode limitar consideravelmente o porte do problema a ser resolvido. Mas, neste caso, como muitos dos elementos são nulos, pois a matriz é esparsa, podem-se utilizar técnicas apropriadas de armazenagem para reduzir a memória necessária.

A seguir serão feitas algumas considerações sobre estas barreiras, através da formalização de alguns conceitos a elas relacionados. Também serão caracterizadas algumas das soluções que vêm sendo adotadas para minimizar estas restrições.

Muitos dos algoritmos numéricos, quando implementados em máquinas digitais produzem resultados inesperados e, muitas vezes, indesejáveis, pois eles dependem de fatores como: o conjunto sobre o qual os cálculos são efetuados na máquina, a forma como os cálculos são efetuados (aritmética), formas adotadas para representar os números, os tipos de arredondamento e da estabilidade tanto do algoritmo quanto do problema.

Referente ao conjunto sobre o qual os cálculos são efetuados, temos que muitos dos números reais não pertencem ao conjunto de números de máquina. O exemplo clássico é o número  $\pi$ , um número real, irracional, com infinitas casas decimais. No computador, os números possuem um número finito de dígitos, portanto o  $\pi$  não pertence a este conjunto. O que se tem é uma representação finita do número real  $\pi$  no computador. O conjunto dos números de máquina geralmente é chamado de conjunto de ponto-flutuante, devido a sua forma de representação. Esta passagem dos reais para ponto-flutuante, do contínuo para o discreto, introduz erros, os quais, em geral, resultam em aproximações da solução e não na solução exata.

A maioria dos computadores utiliza um sistema de números em ponto-flutuante, denotado por  $F(\mathbf{b}, \mathbf{n}, \mathbf{l}, \mathbf{u})$ , onde  $\mathbf{b}$  é o número da base,  $\mathbf{n}$  a precisão da máquina (número de dígitos da mantissa) e  $\mathbf{l}$  e  $\mathbf{u}$  são o menor e maior expoente do intervalo de representação do expoente da base.

Qualquer número  $x$ , não nulo, pertencente a  $F$  tem a forma:

$$x := \pm \left( \frac{d_1}{b^1} + \frac{d_2}{b^2} + \frac{d_3}{b^3} + \dots + \frac{d_n}{b^n} \right) \cdot b^e \quad (1.1)$$

ou,

$$x := \pm m \cdot b^e = \pm 0.d_1 d_2 d_3 \dots d_n \cdot b^e, \quad (1.2)$$

onde  $d_1, d_2, \dots, d_n$  da parte fracionária satisfazem (num sistema normalizado):

- i)  $1 \leq d_1 < \mathbf{b}$ ; (condição de normalização)
- ii)  $0 \leq d_j < \mathbf{b}$  ( $2 \leq j \leq n$ )
- iii) E o expoente  $e$  é tal que  $\mathbf{l} \leq e \leq \mathbf{u}$ . Os valores  $\mathbf{l}$ ,  $\mathbf{e}$ ,  $\mathbf{u}$  são inteiros e  $\mathbf{l} \leq 0$  e  $\mathbf{u} \geq 1$ .

Com isto pode-se formalizar a definição de número de ponto-flutuante normalizado, que é: um número real  $x$  é dito ser um número em ponto-flutuante se é da forma  $x = m \cdot b^e$ , onde a mantissa  $m$  é da forma descrita acima. A União desses números com o zero ( $0 := 0.000\dots 0 \cdot b^1$ ) é dito sistema de números de ponto-flutuante. O conceito de normalização foi introduzido para evitar que um mesmo número real tivesse mais de uma representação no sistema de ponto-flutuante.

Os números transcendentais e irracionais não têm representação finita em nenhuma base, já os números racionais podem ter representação finita em uma base e infinita em outra.

Pode-se observar na representação do sistema de ponto-flutuante que no intervalo compreendido entre zero e um se concentra uma grande quantidade de números de máquina, ocasionando que a distância entre dois números seja a menor, ou seja, é o intervalo mais denso. Esta propriedade é uma das causas de se utilizar a técnica de pivotamento na resolução de sistemas de equações, onde se divide pelo maior, em módulo, resultando em que o quociente esteja no intervalo, numa tentativa de se minimizar o erro de arredondamento.

Outra consideração a ser feita é sobre as operações aritméticas fundamentais, isto é, como elas são efetuadas no sistema de ponto-flutuante, o que caracteriza a aritmética de ponto-flutuante. Referente à soma e subtração de números em ponto-flutuante, só podem ser somados (ou subtraídos) números com expoentes iguais, para tanto são feitos alguns deslocamentos a fim de preparar a soma (ou subtração) e, então, normaliza-se o resultado após a operação soma (ou subtração). Suponha a situação de dois números que são aproximadamente iguais mas de sinais opostos, tendo ambos o mesmo expoente muito pequeno. Ao se normalizar o resultado, o expoente da soma precisa ser reduzido pelo número de casas do deslocamento solicitado à esquerda, mas se tal expoente não puder ser representado, teremos uma perda de significado do resultado. Este problema é chamado de "*underflow*". No Cray Y-MP, estabeleceu-se que quando isto acontece o resultado é tomado por zero. Isto pode acarretar erros graves como é visto em [DIV86], *underflow* destrutivo.

O problema correspondente à outra extremidade é chamado de "*overflow*" do ponto-flutuante, que acontece quando somamos dois números e o resultado em grandeza supera a capacidade de representação. Neste caso, em geral, as máquinas digitais interrompem a execução ou o processamento.

A multiplicação em  $F$  é efetuada multiplicando as mantissas como aparecem. O expoente do resultado é a soma dos expoentes dos fatores modificados pela normalização, quando necessária, no final. A divisão é feita como o produto do inverso do divisor. O caso extremo de *overflow* na divisão ocorre ao se tentar dividir por zero (ou algo muito pequeno); tal tentativa causa a interrupção do processamento, sendo caracterizada por divisão por zero, no Cray Y-MP.

Outro conceito importante é o de arredondamento, que é uma função dos reais em  $F$ , tal que para todo número real  $x$ , há uma representação no sistema de ponto-flutuante  $F$  e para todo número  $y$  pertencente a  $F$ , sua representação é o próprio  $y$ , pois já se deixa representar em  $F$ .

Arredondamento é uma função monotônica, ou seja se  $a$  e  $b$  são números reais e  $a$  é menor ou igual que  $b$ , então os valores que representam  $a$  e  $b$  em  $F$  manterão esta relação.

É a regra que determina a aproximação ou representação do número real  $x$  a um número de máquina de  $F$  que caracteriza o tipo de arredondamento. A seguir são caracterizados, informalmente, os três tipos de arredondamento mais usados. As duas primeiras definições referem-se aos arredondamentos direcionados (conhecidos como para cima e para baixo) muito utilizados dentro da matemática intervalar.

Arredondamento para cima (ou por excesso) é a função que aproxima o número real  $x$  para o maior número de máquina que o contém. Informalmente se tem que para ter a precisão de  $n$  dígitos, trunca-se na posição do dígito  $n$  e soma-se uma unidade a esta posição. Será anotado por  $\Delta x$  e definido por (1.3).



$$\Delta x := \min \{ y \in \mathbf{F} / y \geq x \} \quad \forall x \in \mathbf{R} \quad (1.3)$$

Arredondamento para baixo (ou truncamento) é a função que aproxima o número real  $x$  para o menor número de máquina que o contém. Para se ter a precisão de  $n$  dígitos, trunca-se, simplesmente, na posição do dígito  $n$ . Este é muito comum nas calculadoras e computadores. Será anotado por  $\nabla x$  e definido por (1.4).

$$\nabla x := \max \{ y \in \mathbf{F} / y \leq x \} \quad \forall x \in \mathbf{R} \quad (1.4)$$

O arredondamento simétrico (ou para o número mais próximo de máquina) é a função que aproxima o número real  $x$  para o número de máquina mais próximo. Para se ter  $n$  dígitos de precisão, soma-se meia unidade a posição do dígito  $n+1$  e, após, trunca-se na posição do dígito  $n$ . Este é o arredondamento mais popular, sendo anotado por  $Ox$  e definido por (1.5).

$$Ox := \begin{cases} \nabla x & \text{se } |x - \nabla x| < |x - \Delta x| \\ \nabla x & \text{se } |x - \nabla x| = |x - \Delta x| \text{ e } \nabla x \text{ é par} \\ \Delta x & \text{em caso contrário} \end{cases} \quad (1.5)$$

O erro de arredondamento depende do tipo de arredondamento (função) adotado, da base e da precisão da máquina. O limite do erro de arredondamento para os arredondamentos para baixo e para cima é da ordem de  $b^{(1-n)}$ , enquanto que para o arredondamento simétrico é a metade, ou seja, da ordem de  $(1/2)b^{(1-n)}$ .

O *underflow* e o *overflow* geralmente eram tratados como excessões, mas com a representação de infinito da aritmética de ponto-flutuante ordinária e com a matemática intervalar, eles se tornam um simples caso de arredondamento.

Outra variação do conjunto dos números de máquina, dos números em ponto-flutuante são os números em dupla precisão, ou seja, existe o dobro de casas na mantissa disponíveis para a representação dos números em ponto-flutuante. Ela é utilizada para minimizar o erro de arredondamento.

Como foi visto, o segundo fator que influencia o resultado de problemas é como os cálculos são efetuados na máquina. Isto diz respeito à aritmética e às regras adotadas para representar em ponto-flutuante os resultados reais. Elas também geram incertezas nos resultados, as quais necessitam ser controladas e quantificadas para se evitar surpresas desagradáveis nas soluções.

## 1.2 Modelagem

Modelos físicos não são suscetíveis a certas propriedades como por exemplo, modelagem simples e rápida, universalidade, transportabilidade, custos mínimos. Por isto, tem-se utilizado um princípio universal, que consiste na modelagem através de fórmulas

matemáticas, interdependências e propriedades lógicas. Levando ao conceito do mapeamento do objeto sobre o modelo computacional, o qual geralmente é representado através de propriedades, fórmulas matemáticas e estruturas lógicas.

No processo de resolução de um problema podem ser constatadas algumas fontes de erros. Este processo foi ilustrado pela figura 1.1 e entre as fontes de erro destacam-se:

a) **Erros inerentes aos dados** - São erros nos valores dos dados, causados por inexatidão em medidas (como tempo e distância);

b) **Erros de modelagem** - São erros provenientes das simplificações das situações reais, feitas através de modelos ou procedimentos ou resultantes do mapeamento do objeto observado no modelo matemático e computacional;

c) **Erro de truncamento do modelo** - São erros introduzidos pelo procedimento numérico infinito que, ao serem implementados, são truncados produzindo um erro no resultado; ou decorrentes da perda de estrutura algébrica pela redução da estrutura infinitesimal (discretização);

d) **Erros de arredondamento** - São erros decorrentes do fato das máquinas operam com uma aritmética de ponto-flutuante, onde as operações aritméticas e funções elementares são realizadas com um número finito de dígitos no sistema de representação do computador.

Para se quantificar o efeito final destes erros sobre o resultado, pode-se utilizar qualquer das seguintes formas de quantificação de erros:

a) **Erro absoluto** - mede a diferença entre o valor arredondado e o valor exato;

b) **Erro relativo** - é uma taxa, um quociente entre o erro absoluto e o valor exato. É o mais utilizado pois varia em comparação com a unidade, com a grandeza do número.

c) **Erro nas operações aritméticas** - através do cálculo das expressões para o erro absoluto e relativo no resultado das operações aritméticas, desenvolve-se uma árvore da propagação do erro, obtendo o erro final do resultado. Estudos desta metodologia podem ser encontrados em Dorn [DOR81];

d) **Fórmula do número de algarismos significativos corretos** - esta fórmula consta em [CLA83] e foi proposta com o objetivo de determinar, entre duas aproximações, o número de dígitos significativos corretos de uma aproximação. Depende do sistema de ponto-flutuante, do tipo de arredondamento, da base e das aproximações. É muito utilizada como critério de parada de processos iterativos.

Uma vez caracterizados os tipos de erros e formas de quantificá-los, deve-se fazer considerações sobre a possibilidade ou não de se controlar ou evitar os erros de modelo, do método e de arredondamentos. O erro de modelo não pode, em geral, ser eliminado; é necessário, no entanto, que seja delimitado através de experiências e experimentos.



Por outro lado, considerando que o erro do método seja decorrente de processos infinitesimais contidos em séries, integrais e equações diferenciais, poderia-se supor, por exemplo, ser uma tarefa impossível conseguir a verificação automática do resultado. Mas é possível através de uma aritmética de alta exatidão, controle de arredondamentos, intervalos e produto escalar exato.

Quando se quer falar de um sistema de ponto-flutuante ou quando se quer caracterizá-lo, é necessário especificar: a base numérica do sistema, o número de dígitos da mantissa, o intervalo de abrangência do expoente da base e como é feita a representação destes números. Também deve-se considerar como é feito o tratamento de *underflow* e *overflow*, como são efetuados as operações aritméticas e os tipos de arredondamentos disponíveis e utilizados nas operações, pois isto influenciará a análise e a quantificação dos erros de cálculo efetuado neste sistema.

O Padrão IEEE 754<sup>2</sup> surgiu na tentativa de padronizar os diferentes sistemas de ponto-flutuante existentes nas diversas máquinas disponíveis no mercado, estabelecendo um parâmetro de qualidade nos resultados gerados por operações em ponto-flutuante. Ele define o formato dos números em precisão simples e estendida, as regras para se realizar a aritmética binária de ponto-flutuante e como tratar os casos especiais. São definidos quatro tipos de arredondamentos: o simétrico, o direcionado para zero, para menos infinito e para mais infinito. As operações definidas segundo este padrão possuem máxima exatidão.

Em *hardware* se utiliza a técnica do dígito de guarda para minimizar a propagação do erro. Ela consiste em se ter na arquitetura interna mais um dígito (pelo menos) para guardar o dígito  $n+1$  de cada número. As informações contidas no(s) bit(s) de guarda permitem que a operação de normalização preserve a significância de todos os bits da mantissa.

Por fim, destaca-se o problema de estabilidade ou instabilidade de problemas e algoritmos. Diz-se que um problema matemático é dito instável ou mal condicionado se sua solução é muito sensível a pequenas mudanças nos dados de entrada. Caso pequenas alterações nos dados de entrada sejam acompanhadas de pequenas variações na solução o problema é dito estável ou bem condicionado.

A instabilidade de algoritmos refere-se à dependência da maneira pela qual os dados são manipulados, à forma como os cálculos são feitos, podendo ser muito boa algumas vezes e em outras não.

Devido ao sistema e à aritmética de ponto-flutuante, podem-se ter resultados não confiáveis, ou mesmo totalmente errados em função de uma acumulação catastrófica dos

---

<sup>2</sup>Institute of Electrical and Electronics Engineers. Padrão de aritmética binária de sistema de ponto flutuante proposto em 1980 e reconhecido em 1985.

erros. Uma alternativa é realizar uma análise de erros, que muitas vezes é complicada e as vezes proibitiva em termos de tempo computacional. A situação é tão complexa que se torna impossível ao usuário de um *software* numérico estabelecer considerações sobre o erro do processo utilizado para o cálculo da solução do problema.

Uma validação tem que se certificar quanto ao modelo, para que não seja incorreto quanto à aritmética, para que não cause surpresas de imprecisão e quanto ao *software*, para que não tenha erros.

- a) A **incerteza do modelo** diz respeito às teorias, dados que não refletem a realidade corretamente, devido, por exemplo, ao problema ser mal condicionado.
- b) A **incerteza numérica** refere-se à propagação do erro de arredondamento (devido a aritmética) e suas consequências no resultado por causa da instabilidade de algoritmos.
- c) A **incerteza do software** refere-se à possibilidade de *softwares* ainda conterem erros depois da depuração, ou a erros provenientes de defeitos do compilador, refletidos numa execução imprópria do programa.

A principal idéia para verificar modelos é a consistência, ou seja, o modelo deve ser consistente com as observações experimentais e deve ter o comportamento esperado. A consistência através da observação do experimento é a melhor forma de conferir a validade de um modelo, mas nem sempre é possível utilizá-la. Portanto, a alternativa é conferir a consistência dos resultados de um programa com outros modelos.

A **análise sensitiva** parte do princípio de que se tem a solução do problema pelo programa. Mudam-se então levemente os dados ou a precisão e observa-se o quanto mudou o resultado. As variações nos resultados podem ser resultantes de mudanças nos dados de entrada, na precisão, nos métodos usados ou no modelo.

A análise sensitiva de um modelo pode ser complicada mas é necessária para que se tenha um alto grau de confiança no resultado correto. Ela tem suas limitações, sendo uma destas que os modelos complexos podem ter centenas de parâmetros, o que resulta em muitos testes para que se examinem todos os parâmetros.

Uma alternativa para análise sensitiva é a análise de erros, onde uma análise matemática (analítica) é feita com base nas mudanças causadas na solução do sistema. Existem duas dificuldades nisto, a primeira é a da análise matemática e a segunda é que erros podem ser tão grandes que a estimativa seja invisível, ou uma super-estimativa pode mascarar a utilidade do algoritmo.

Em virtude destes fatores, desde o início do uso de computadores para solucionar problemas numéricos, sempre houve a preocupação com a qualidade dos resultados. Várias formas de controle de erro na solução vêm sendo empregadas para minimizar e controlar a propagação de erros, mas eles ocorrem!

Surge então a questão: quando é possível afirmar que um resultado está correto? Existem duas possibilidades de se obter resultados corretos:

- a) Fornecendo-se fronteiras dentro das quais a solução (com certeza) se situa;
- b) Fornecendo-se apenas o número de casas que coincidem com a solução exata.

Pelo fato da aritmética de ponto-flutuante produzir erros na representação de números reais, sentiu-se a necessidade de se ter uma nova ferramenta ou metodologia para o controle de erros. A aritmética intervalar, que utiliza intervalos de números reais para representar valores infinitos, providenciou uma ferramenta para estimar e controlar estes erros automaticamente. Portanto, uma das justificativas de se trabalhar com intervalos é o fato da aritmética real não ser suficientemente confiável em muitos problemas ligados, por exemplo, à física ou química experimental, onde medições nem sempre podem ser feitas de forma exata, mas dentro de uma certa "faixa", a qual pode ser representada por intervalos. Um exemplo disto é a aceleração da gravidade, que como a maioria dos dados físicos, é um dado inexato, mas com o limite de erro conhecido. A aceleração no nível do mar é conhecida por  $g = 9.807 \pm 0.027 \text{ m/s}^2$ . Na forma intervalar tem-se:  $G = [9.780, 9.834] \text{ m/s}^2$ .

Portanto, intervalos podem ser aplicados para representar valores desconhecidos ou para representar valores contínuos que podem ser conhecidos ou não. No primeiro sentido servem para controlar o erro de arredondamento, para representar dados inexatos, para representar aproximações e erros de truncamento de procedimentos, como consistência lógica de programas, como critério de parada de processos iterativos e como critério de parada de métodos de contração. No segundo sentido servem, por exemplo, para testes de verificação computacional quanto a existência e unicidade de soluções de sistemas não lineares. Esta aplicação procura providenciar condições suficientes para a existência e unicidade de zeros de sistemas de equações não lineares. A condição necessária não é verificada manualmente, mas automaticamente pelo computador.

Por outro lado, com o desenvolvimento de máquinas que processam mais velozmente as operações aritméticas e, que têm uma capacidade de armazenamento muito grande, tem-se conseguido implementar métodos numéricos que há bem pouco tempo atrás não eram nem sonhados, como computações de larga escala.

Computações de larga escala são processos que exigem uma grande quantidade de cálculos, de onde vem o termo "larga escala". Elas ocorrem nas áreas científicas e militares, nas engenharias, no estudo e exploração de fontes de energia, na medicina, na inteligência artificial e na pesquisa básica. Para resolver problemas nestas áreas são necessários computadores de alto desempenho, que são muitas vezes chamados de supercomputadores. Nestas máquinas, o processamento é vetorial e/ou paralelo.

No processamento vetorial dispõe-se de um *hardware* que possibilita a troca de operações escalares por operações vetoriais, o que produz uma diminuição do tempo de processamento. No processamento paralelo deve-se dispor de duas ou mais unidades de processamento ou processadores (CPU's), onde tarefas são efetuadas ao mesmo tempo.



Um exemplo de área onde supercomputadores são necessários é na modelagem multidimensional da atmosfera, do ambiente terrestre, do espaço externo e do mundo econômico. Muitos cientistas têm-se concentrado nestas áreas. A modelagem preditiva é feita através de extensas simulações computacionais de experimentos, as quais geralmente envolvem computações de larga-escala para garantir a exatidão desejada e a minimização do tempo necessário para solução. Tais modelagens numéricas requerem um estado da arte computacional de velocidade aproximada de milhões de *Megaflops*.

O analista atmosférico necessita resolver equações de modelos gerais de circulação em computadores. O estado atmosférico é representado pela superfície pressionada, pelo campo virado, pela temperatura e pela variação do vapor d'água. Estes estados variáveis são governados pelas equações da dinâmica de fluidos da Navier-Stoke num sistema de ordenadas esféricas.

A computação é levada a cabo por grade tri-dimensional que particiona a atmosfera verticalmente em  $k$  níveis, horizontalmente em  $M$  intervalos de longitude e  $N$  intervalos de latitude. A quarta dimensão é adicionada com o número  $P$  de passos usados na simulação. Utilizando uma grade de 270 milhas de um lado (o correspondente a 435Km), uma previsão de 24h precisa realizar cerca de cem bilhões operações de dados.

Esta previsão pode ser feita num computador de 100 *megaflops* em cerca de 100 minutos, ou seja, uma hora e quarenta minutos. A grade de 435 Km dá uma previsão entre São Paulo e Rio de Janeiro, aproximadamente. Aumentando um pouco esta grade, para abranger uma região maior, seria necessário uma máquina mais veloz ou se correria o risco de levar mais tempo de cálculo para a previsão do que o dia que se quer prever.

### 1.3 O Trabalho

Recapitulando, verificou-se dois tipos de barreiras decorrentes das limitações da máquina, que são: qualidade do resultado e tamanho do problema capaz de ser resolvido, envolvendo memória e tempo necessários. Neste trabalho está sendo proposto um estudo da viabilidade da construção de uma aritmética de alto desempenho capaz de minimizar estas barreiras. No caso do supercomputador Cray, este estudo se concretiza na construção de uma ferramenta denominada aritmética vetorial intervalar, cujo protótipo é a biblioteca de rotinas intervalares *libavi.a*. Ela incorpora a aritmética intervalar e o processamento vetorial. Esta ferramenta foi projetada para o ambiente do Supercomputador Cray Y-MP2E, o que é constatado no padrão de documentação adotado, que é o da Cray.

Neste **capítulo inicial** foi feita uma introdução ao ambiente de cálculos e operações em computadores, através da descrição de sistemas de ponto-flutuante, problemas de representação e arredondamentos, tipos de erros e sua propagação no resultado de computações numéricas e problemas de instabilidade.



No **capítulo dois** são caracterizadas: a Aritmética de Alta Exatidão com seus requisitos, a verificação automática do resultado e sua influência nos métodos numéricos e as linguagens para computação científica, especialmente as linguagens XSC que incorporam estas características.

Foram desenvolvidos relatórios de pesquisa que contêm uma completa revisão sobre Matemática Intervalar, Aplicações da Matemática Intervalar, Processamento Vetorial e Limitações do Processamento vetorial ([DIV92, DIV94b, DIV91, DIV93, DIV95]), disponíveis na Biblioteca do Instituto de Informática da UFRGS.

No **capítulo três** são descritos os ambientes computacionais onde esta pesquisa se desenvolveu, ou seja o Supercomputador Cray Y MP2E com a linguagem Fortran 90 e Personal Computer 486dx com a linguagem de Pascal XSC. O objetivo deste capítulo é situar o leitor, caracterizando os ambientes.

Maiores detalhes sobre o supercomputador Cray podem ser encontrados em [DIV93]. Este relatório contém uma descrição detalhada da arquitetura, dos *softwares* disponíveis (incluindo as bibliotecas matemática e científica), indicativos sobre o desenvolvimento de análise de dependências e orientações para se montar uma seção de trabalho na estação de forma otimizada, obtendo assim, uma visão geral do processamento em cada uma das máquinas envolvidas. Existem exemplos detalhados do acesso e execução de programas.

Já em relação ao Pascal XSC, também foi desenvolvido um relatório, o [DIV94d], onde são dadas em detalhes as características da linguagem, instalação e execução, os módulos avançados e exemplos de aplicações.

No **capítulo quatro** é caracterizado o problema de resolução de sistemas de equações, sua aplicabilidade nas ciências e engenharias e os problemas de instabilidade numérica. São caracterizados os métodos de resolução de sistemas de equações lineares, através da descrição dos métodos pontuais (reais) e dos métodos intervalares de resolução de sistemas. Alguns destes métodos foram implementados e fazem parte da biblioteca de rotinas intervalares *libavi.a*.

No **capítulo cinco** é formalizada a proposta de desenvolvimento da biblioteca de rotinas intervalares de alto desempenho. É também caracterizada a biblioteca de rotinas intervalares *libavi.a*, composta de cerca de 290 rotinas intervalares, organizada em quatro módulos: o módulo *básico* de rotinas básicas intervalares; o módulo de rotinas de vetores e matrizes intervalares - *mvi*; o módulo de rotinas intervalares complexas *ci* e o módulo *aplic*, composto por rotinas intervalares da álgebra existentes na biblioteca BLAS. Estes módulos são descritos juntamente com a sua hierarquia. São descritas também, as características e potencialidades desta ferramenta. É especificada ainda a biblioteca de rotinas científicas intervalares *libselint.a*, resultante dos testes desenvolvidos para validar a biblioteca de rotinas intervalares e o uso da matemática intervalar. Ela é composta por rotinas que implementam métodos intervalares de resolução de equações algébricas; e de

sistemas de equações lineares baseados em operações algébricas, em refinamentos e em iterações.

No **capítulo seis** é feita uma breve comparação entre a biblioteca *libavi.a* com algumas bibliotecas que abordam álgebra linear e resolução de sistemas de equações, como *Numeral*, *libsci.a* e Pascal-XSC. A base para a comparação foi o tipo de função disponível em cada biblioteca e as características peculiares a cada biblioteca. Esta comparação auxiliou a definição das rotinas do módulo *aplic* da *libavi.a*.

A seguir, é analisado o uso efetivo da matemática intervalar na solução de problemas práticos em supercomputadores. A primeira parte dos testes visou a verificação e validação da biblioteca *libavi.a*, enquanto que a segunda parte visou o uso da matemática intervalar no Cray e a análise deste uso. Esta análise foi baseada nos resultados obtidos através da *libavi.a* e nos resultados dos programas desenvolvidos em Pascal-XSC e, abrangeu os aspectos de qualidade do resultado (exatidão) e tempo de processamento (ganho de velocidade com a vetorização).

Por fim, na **conclusão** é apresentado um resumo do trabalho, são comentados os resultados obtidos, são caracterizadas as limitações deste trabalho e sugestões de continuidade desta pesquisa.

## 2 ARITMÉTICA DE ALTA EXATIDÃO

Neste capítulo é abordado o estado da arte da aritmética computacional, que pode ser caracterizada pela aritmética de alta exatidão. Portanto, são descritos os requisitos desta aritmética, que são: o uso de arredondamentos direcionados e operações aritméticas com máxima exatidão, matemática intervalar e produto escalar ótimo. São ainda abordados tópicos como a verificação automática do resultado e linguagens para computação científica, conhecidas como XSC, que incorporam à sua definição estes requisitos.

### 2.1 Características da Aritmética de Alta Exatidão

Existe um abismo entre o atual estágio de desenvolvimento da capacidade de processamento e a aritmética de ponto-flutuante. Os computadores tornaram-se muito mais rápidos. A aritmética de ponto-flutuante está dentro do *hardware* e, conseqüentemente, é muito rápida. Na metade dos anos 50, computadores podiam realizar por volta de 100 operações de ponto-flutuante por segundo. Atualmente, computadores rápidos podem executar alguns milhões de operações por segundo, sendo que cada operação de ponto-flutuante tem máxima exatidão, isto é, o resultado de cada operação difere no máximo de 1 ou 1/2 unidade na última casa (decimal) de acordo com a escolha do arredondamento. Entretanto, após duas ou mais operações de ponto-flutuante o resultado pode estar completamente errado. Existem falhas quase inexplicáveis que ainda não foram excluídas depois de quarenta anos da aritmética de ponto-flutuante e dos grandes avanços da tecnologia.

Entende-se por aritmética computacional todas as operações que têm de ser definidas para os conjuntos dos números em ponto-flutuante, em precisão simples e em dupla precisão.

O padrão IEEE 754 foi um grande passo para a evolução da aritmética de ponto-flutuante. Ele define as 4 operações básicas (+, -, \*, /) com máxima exatidão, incluindo o controle de arredondamento para zero, para o mais próximo, para menos infinito e para mais infinito. Contudo o padrão não especifica os arredondamentos para variáveis complexas, vetores e matrizes e também não inclui o produto escalar ótimo, essencial para garantia da alta exatidão nos cálculos.

O uso da matemática intervalar aliada ao produto escalar ótimo proporcionaria uma **aritmética de alta exatidão**, com controle de erros com limites confiáveis, além de proporcionar condições necessárias para provas de existência e não existência da solução de diversas equações.

Existem, em princípio, dois métodos básicos distintos para a definição da aritmética computacional. Eles são chamados de método vertical e método horizontal. No método horizontal a aritmética é assumida como conhecida no conjunto dos reais. Por outro lado, no método vertical a aritmética é definida para os reais e para o conjunto de

ponto-flutuante. Então, a partir destes conjuntos, a aritmética é estendida a vetores, matrizes e intervalos.

O método vertical, segundo [KUL83], corresponde à forma tradicional de se definir aritmética computacional, ou seja, os números reais  $\mathbf{R}$  são mapeados por um sistema de ponto-flutuante  $\mathbf{F}$  e, sobre estes números de máquinas, são implementados os novos conjuntos como o conjunto dos intervalos ( $\mathbf{IF}$ ) ou o conjunto dos números complexos.

A definição das operações aritméticas é feita através de subrotinas e funções. Este método incorre sempre em algum erro, uma vez que os resultados exatos não são necessariamente representados em  $\mathbf{F}$ , o que dificulta a estimativa do erro.

O método horizontal é uma nova proposta de definição de aritmética computacional. Ela foi proposta por Kulisch ([KUL81,KUL83]) e se tornou disponível na prática no projeto da linguagem PASCAL-SC. Nesta proposta, os intervalos reais são obtidos a partir de números reais. Para os intervalos são definidas as operações pela regra geral (RG definida em (2.1)) onde se têm os arredondamentos direcionados gozando de certas propriedades de forma que elas definem um semimorfismo. Todas as operações definidas por semimorfismo proporcionam máxima exatidão, no sentido de que não existe nenhum elemento do subconjunto entre o resultado correto de uma operação e sua aproximação no subconjunto.

### 2.1.1 Arredondamentos Direcionados e Operações Aritméticas Básicas

Funções de mapeamento se fazem necessárias para representar reais em números de máquina. Estes mapeamentos são denominados de arredondamentos. Para se ter a aritmética de ponto-flutuante de máxima exatidão, devem-se ter os dois arredondamentos direcionados (para baixo  $\nabla$  e para cima  $\Delta$ , definidos pelas fórmulas (1.3) e (1.4)) juntamente com as operações aritméticas correspondentes para cada um dos arredondamentos monotônicos. Estes arredondamentos são necessários para se ter a garantia nos cálculos. Eles equivalem a operações sobre intervalos de número de máquina. Os algoritmos de implementação dos arredondamentos monotônico e anti-simétrico [], dos dois arredondamentos direcionados e das correspondentes operações aritméticas podem ser encontradas em [KUL81].

Seja  $M$  um dos conjuntos de números reais ( $\mathbf{R}$ ) ou complexos ( $\mathbf{C}$ ), de vetores reais ( $\mathbf{VR}$ ) ou complexos ( $\mathbf{VC}$ ), matrizes reais ( $\mathbf{MR}$ ) ou complexos ( $\mathbf{MC}$ ), ou ainda de intervalos reais ( $\mathbf{IR}$ ) ou complexos ( $\mathbf{IC}$ ), vetores e matrizes de intervalos reais e complexos ( $\mathbf{VIR}$ ,  $\mathbf{VIC}$ ,  $\mathbf{MIR}$ ,  $\mathbf{MIC}$ ). Seja  $N$  um subconjunto de  $M$ , ou seja,  $N \subseteq M$ . A seguir será feito um resumo dos requerimentos de um semimorfismo.

Antes, entretanto, serão analisadas as operações aritméticas  $+$ ,  $-$ ,  $*$ ,  $/$ . Essas operações para números reais são aproximadas pelas operações em ponto-flutuante. Se  $x$  e  $y$  são números em ponto-flutuante, o ponto exato  $x*y$  em si, em geral, não pertence a  $F(b,n,l,u)$  uma vez que a mantissa  $x*y$  tem  $2n$  dígitos. Por análogas razões, a soma exata



$x+y$  também em geral não é um número em ponto-flutuante. O resultado de operações em ponto-flutuante precisa ser um número em ponto-flutuante. O melhor que se pode fazer é arredondar o resultado exato para um filtro ponto-flutuante e tomar a versão arredondada como a definição da operação em ponto-flutuante.

Se  $\#$  é uma das operações exatas  $+$ ,  $-$ ,  $*$ ,  $/$ , seja  $[\#]$  denotando a operação em ponto-flutuante correspondente. Então a escolha de operação em ponto-flutuante é expressa pela fórmula matemática denominada de Regra Geral.

As operações em  $N$  são definidas pela regra geral (RG), onde:

$$(\forall a, b \in N) a [\#] b := [ ] (a\#b) \quad (2.1)$$

para todas operações  $\#$ , onde  $[ ]:M \rightarrow N$  denota um mapeamento monotônico e anti-simétrico, um arredondamento de  $M$  em  $N$ , tendo as seguintes propriedades:

$$(\forall a \in N) [ ] a := a \text{ (arredondamento)} \quad (2.2)$$

$$(\forall a, b \in M) (a \leq b \Rightarrow [ ] a \leq [ ] b) \text{ (monotônico)} \quad (2.3)$$

$$(\forall a \in M) [ ] (-a) := -[ ] a \text{ (anti-simetria)} \quad (2.4)$$

Em particular, entre o resultado correto (no sentido do número real) e o resultado aproximado  $x[\#]y$  (no sentido do filtro dos números em ponto-flutuante) nenhum outro número em ponto-flutuante pode ser achado no conjunto.

Pode-se resumir esta discussão dizendo que arredondamentos admissíveis geram aritmética de ponto-flutuante com exatidão máxima pelo uso da RG. No caso de  $M$  ser um conjunto de intervalos, o arredondamento  $[ ]$  tem a propriedade adicional (2.5).

$$(\forall a \in M) a \leq [ ] a \quad (2.5)$$

Neste caso  $\leq$  denota a inclusão  $\subseteq$ . As propriedades (2.1), (2.2), (2.3), (2.4) e no caso de conjuntos de intervalos, (2.5), definem um **semimorfismo**<sup>1</sup>. Todas as operações definidas por semimorfismo proporcionam máxima exatidão no sentido de que não existe nenhum elemento do subconjunto entre o resultado correto de uma operação e sua aproximação no subconjunto. Ainda mais, no caso de mapeamento de números reais em números de ponto-flutuante (números de máquina), os dois arredondamentos direcionados para baixo ( $\nabla x$ ) e para cima ( $\Delta x$ ), como definidos no capítulo 1, são importantes. Eles gozam das propriedades descritas pelas fórmulas (2.2), (2.3) e por (2.6).

$$(\forall a \in M) \nabla a \leq a \leq \Delta a \quad (2.6)$$

<sup>1</sup>Semimorfismo - estrutura algébrica definida em [Kul87]

Para estes arredondamentos direcionados, são definidos os operadores correspondentes pela regra geral (2.1).

$$(\forall a, b \in \mathbb{N}) a \# b := [ ] (a \# b) \quad (2.7)$$

onde  $[ ]$  pode ser o arredondamento para baixo  $\nabla x$  ou para cima  $\Delta x$  e a operação  $\#$  pode ser soma, subtração, multiplicação ou divisão.

Os arredondamentos direcionados para baixo ou para cima, assim como os operadores correspondentes são necessários para uma estimativa de erro realística em cálculos numéricos. O arredondamento direcionado para baixo mapeia todo intervalo entre dois números vizinhos do subconjunto  $\mathbb{N}$  em um limite inferior representável no computador do intervalo. O arredondamento monotônico direcionado para cima possui a propriedade oposta, dando o limite superior do intervalo.

Através de implementações de operações para diferentes espaços através de semimorfismo, foi observado que, em geral, as operações definidas por semimorfismo tornam-se mais rápidas do que rotinas que implementam estas operações pela maneira usual.

Foi verificado que todas as operações definidas por semimorfismo podem ser concretizadas em computadores pelo uso da técnica modular em termos de linguagem de programação de alto nível se: um conceito de operador universal ou uma notação de operador estiver disponível para todas as operações nesta linguagem de alto nível e, se as 15 operações fundamentais da figura 2.1 para números em ponto-flutuante estiverem disponíveis.

Arredondamento	Operações				
$O_x$	$O^+$	$O^-$	$O^*$	$O /$	$O \cdot$
$\nabla x$	$\nabla^+$	$\nabla^-$	$\nabla^*$	$\nabla /$	$\nabla \cdot$
$\Delta x$	$\Delta^+$	$\Delta^-$	$\Delta^*$	$\Delta /$	$\Delta \cdot$

Fig. 2.1 Operações (15) fundamentais de arredondamentos

Aqui,  $O_x$  indica as operações semimorfás definidas pela RG (2.1) usando arredondamento monotônico anti-simétrico, com arredondamento para número mais próximo de máquina. Os símbolos  $\nabla x$  e  $\Delta x$  denotam as operações definidas pela regra geral (2.1) e, respectivamente, arredondamento direcionado para baixo e para cima. A operação ponto indica produto escalar para cada um dos tipos de arredondamento com máxima exatidão.

## 2.1.2 Características da Matemática Intervalar

O uso da aritmética intervalar foi desenvolvido inicialmente por Moore ([MOO66]). Este ramo da matemática veio se desenvolvendo desde então. A matemática intervalar foi proposta em 1974 por Leslie Fox, combinando diferentes áreas como:

aritmética intervalar, análise intervalar, topologia intervalar, álgebra intervalar entre outras. Ela trata com dados na forma de intervalos numéricos e surgiu com o objetivo de automatizar a análise do erro computacional, trazendo uma nova abordagem que permite um controle de erros com limites confiáveis, além de provas de existência e não existência da solução de diversas equações.

Intervalos podem ser aplicados para representar valores desconhecidos ou para representar valores contínuos que podem ser conhecidos ou não. No primeiro sentido servem para controlar o erro de arredondamento e para representar dados inexatos, aproximações e erros de truncamento de procedimentos (como consistência lógica de programas, critério de parada de processos iterativos e critério de parada de métodos de contração). No segundo sentido servem, por exemplo, para testes de verificação computacional e para a existência e unicidade de soluções de sistemas não lineares. A condição necessária não é verificada manualmente, mas automaticamente pelo computador.

O uso da matemática intervalar tem sofrido críticas em função de dois problemas principais, os quais se resumem em que, muitas vezes podem resultar intervalos pessimistas, ou seja, demasiadamente grandes, que não possuem muita informação sobre o resultado, gastando-se muito tempo de máquina. Estas duas críticas são facilmente refutadas, uma vez que, com a aritmética avançada, os resultados são produzidos com máxima exatidão. Quanto ao tempo de processamento, depende da forma como foi implementada. Existem várias formas, tanto via software quanto via hardware.

### 2.1.2.1 Aritmética intervalar real

Uma abordagem detalhada sobre aritmética intervalar pode ser encontrada nos livros textos básicos da área, como os de: Moore<sup>2</sup>, Kulisch<sup>3</sup>, Alefeld e Herzberger<sup>4</sup>, Neumaier<sup>5</sup>, ou ainda, o de Claudio<sup>6</sup>. Aqui só serão revistos os conceitos básicos.

Um intervalo real, ou simplesmente um intervalo é um conjunto fechado e limitado de números reais. Sejam  $x_1$  e  $x_2$  dois números reais tais que  $x_1$  é menor ou igual a  $x_2$ . Definimos o intervalo finito  $X$ , o qual será denotado por  $[x_1, x_2]$  ao conjunto descrito por (2.1), onde  $x_1$  é o limite inferior e  $x_2$  o limite superior do intervalo  $X$ . Um intervalo real cobre todos os números reais entre os dois limites. Algumas vezes estes limites são chamados de ínfimo e supremo do intervalo, respectivamente.

$$X := [x_1, x_2] = \{ x \mid x_1 \leq x \leq x_2 \} \quad (2.8)$$

---

<sup>2</sup>[MOO66, MOO79]

<sup>3</sup>[KUL81, KUL83, KUL88]

<sup>4</sup>[ALE83]

<sup>5</sup>[NEU90]

<sup>6</sup>[CLA89]

Desta definição, pode-se ter o caso particular onde  $x_1=x_2$ , o que produz o intervalo degenerado ou intervalo pontual, que é na realidade um conjunto que contém um único número real.

Uma vez definido o intervalo, define-se o universo dos intervalos, onde será desenvolvido toda a teoria dos intervalos. Seja  $X$  um intervalo real, chama-se  $\mathbf{IR}$  o conjunto de todos os intervalos reais e é anotado por:

$$\mathbf{IR} := \{X \mid X \text{ é um intervalo}\} \quad (2.9)$$

Uma vez que intervalos são conjuntos, a igualdade, as relações de pertinência  $\in$ , subconjunto  $\subseteq$ , subconjunto próprio  $\subset$ , superconjunto ou conter  $\supseteq$ , superconjunto próprio  $\supset$  e intersecção  $\cap$  são definidos da forma usual de conjuntos.

Um intervalo  $X$  é dito ser contido no interior de  $Y$ , se  $y_1 < x_1$  e  $x_2 < y_2$ . Neste caso é escrito  $X \overset{\circ}{\subseteq} Y$ , sendo também conhecido como relação interna de inclusão.

Sejam  $X$  e  $Y$  dois intervalos reais quaisquer, definimos a operação aritmética  $\#$  pela fórmula (2.10).

$$X \# Y := \{x \# y \mid x \in X \text{ e } y \in Y\} \in \mathbf{IR}, \text{ onde } \# \in \{+, -, /, *\} \quad (2.10)$$

Observa-se que o resultado é um intervalo e que contém qualquer operação entre os elementos de  $X$  e  $Y$ . Observa-se ainda que  $X/Y$  somente está definido se o Zero não pertence a  $Y$ .

Sejam os intervalos reais  $X$  e  $Y$  de forma:  $X:= [x_1, x_2]$ ,  $Y:= [y_1, y_2]$ . Temos as operações aritméticas definidas como segue:

$$X + Y := [x_1, x_2] + [y_1, y_2] = [x_1+y_1, x_2+y_2] \quad (\text{soma}); \quad (2.11)$$

$$X - Y := [x_1, x_2] - [y_1, y_2] = [x_1-y_2, x_2-y_1] \quad (\text{subtração}); \quad (2.12)$$

$$X * Y := [x_1, x_2] \cdot [y_1, y_2] = [\min\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}, \max\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}] \quad (\text{produto de intervalos}) \quad (2.13)$$

$$X / Y := [x_1, x_2] / [y_1, y_2] = [x_1, x_2] * [1/y_2, 1/y_1] \quad (\text{se } 0 \notin [y_1, y_2]) \quad (\text{divisão de intervalos}) \quad (2.14)$$



As tabelas 2.1 e 2.2 são dadas para facilitar a implementação da multiplicação e divisão de intervalos.

Tab 2.1 Multiplicação de Intervalos

X.Y	$0 \leq y1$	$y1 < 0 < y2$	$y2 \leq 0$
$0 \leq x1$	$[x1y1, x2y2]$	$x2y1, x2y2]$	$[x2y1, x1y2]$
$x1 < 0 < x2$	$[x1y2, x2y2]$	$[\min\{x1y2, x2y1\}, \max\{x1y1, x2y2\}]$	$[x2y1, x1y1]$
$x2 \leq 0$	$[x1y2, x2y1]$	$[x1y2, x1y1]$	$[x2y2, x1y1]$

Tab 2.2 Divisão de Intervalos,  $0 \notin Y$

X/Y	$0 < y1$	$y2 < 0$
$0 \leq x1$	$[x1/y2, x2/y1]$	$[x2/y2, x1/y1]$
$x1 < 0 < x2$	$[x1/y1, x2/y1]$	$[x2/y2, x1/y2]$
$x2 \leq 0$	$[x1/y1, x2/y2]$	$[x2/y1, x1/y2]$

Estas operações gozam da propriedade de fechamento no conjunto dos intervalos, como pode ser verificado em [DIV92]. Observa-se, ainda, que as operações de subtração e divisão de intervalos não são as operações inversas da adição e multiplicação de intervalos, respectivamente. Portanto, não formam uma estrutura do grupo como no caso das operações com reais. A falta destas propriedades para as operações de adição e multiplicação constituem duas das diferenças entre a aritmética intervalar e a aritmética real. Outra diferença está na propriedade distributiva, ou seja,  $A(B+C)$  nos reais é igual a  $AB+AC$ . Na aritmética intervalar ela não é válida, a não ser em casos especiais, como no caso de A ser um valor real. Em geral, tem-se que  $A(B+C) \subseteq AB+AC$ , ou seja, semidistributiva.

Há, por fim, a propriedade de inclusão monotônica nas operações intervalares, onde para A, B, C e D intervalos reais e, se A está contido em B e se C está contido em D, então A operado com C estará contido em B operado com D ( $A*C \subseteq B*D$ ). Isto é verificado para as quatro operações aritméticas intervalares básicas.

Serão consideradas agora algumas das propriedades dos intervalos como conjuntos. Diz-se que dois intervalos, definidos num mesmo conjunto ordenado, são iguais quando seus limites inferiores e superiores são iguais. Define-se a intersecção de dois intervalos X e Y, como o intervalo resultante do máximo dos limites inferiores e do mínimo dos limites superiores. Caso o resultado não seja um intervalo, é devido à intersecção ser vazia, ou seja eles são disjuntos.

A união de intervalos não disjuntos é definida pelo intervalo resultante do mínimo dos limites inferiores e do máximo dos limites superiores. Define-se ainda, a união desconexa ou convexa de dois intervalos X e Y, anotada por  $X \cup Y$ , da mesma forma, só que também é aplicada a intervalos disjuntos.

Seja o intervalo  $X=[x_1, x_2]$ , define-se diâmetro de  $X$ , anotado por  $d(X)$  ou  $w(X)$ , por:  $d(X)=w(X)=x_2-x_1$ . Destaca-se que o diâmetro é sempre maior ou igual a zero. O raio de um intervalo, anotado por  $r(X)$ , é definido por  $r(X):=(x_2-x_1)/2$

Definem-se o menor e maior valor absoluto de  $X$ , anotados por  $\langle X \rangle$  e  $|X|$  respectivamente, o mínimo e o máximo dos valores absolutos dos limites. São funções que associam a cada intervalo um valor real. Matematicamente se tem que  $\langle X \rangle := \min\{|x|/x \in X\}$  e  $|X| = \max\{|x_1|, |x_2|\}$ . A função intervalar valor absoluto é definida como sendo o intervalo cujos extremos são o menor valor absoluto e o maior valor absoluto.

Define-se o ínfimo do vetor  $X$ , como o limite inferior do intervalo. O supremo do intervalo é definido como o limite superior do intervalo. São anotados por:  $\inf(X):=x_1$  e  $\sup(X):=x_2$ . Define-se, ainda, o ponto médio do intervalo, anotado por  $m(X)$  por:  $m(X):=(x_1+x_2)/2$  ou por  $m(X):=\inf(X)+(\sup(X)-\inf(X))/2$ . Daí se tem a relação:

$$\inf(X) \leq m(X) \leq \sup(X) \quad (2.15)$$

Por fim, anota-se o extremo do intervalo e o interior do intervalo por  $X_{\text{lim}}$  e  $X_{\text{int}}$  (ou  $\overset{\circ}{X}$ ). A figura 2.2 ilustra os atributos de um intervalo.

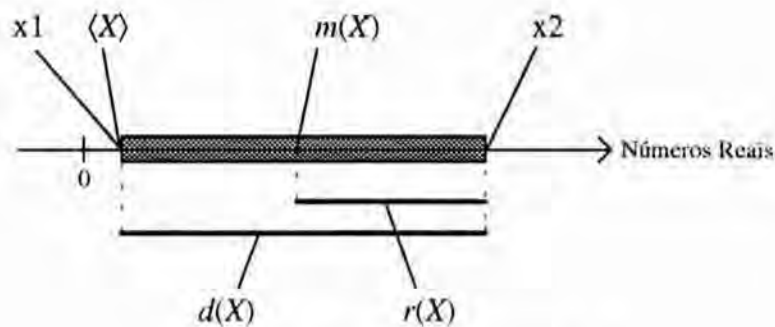


Fig 2.2 Atributos de um intervalo  $X$

Suponha que  $X$  contém um número real  $x$ , cujo valor exato não é conhecido. Para se avaliar a qualidade desta inclusão é definido diâmetro relativo de um intervalo  $X$ . Anotado por  $d_{\text{rel}}$ , e dado por  $d(X)/\langle X \rangle$  se  $0 \notin X$  e  $d(X)$  caso contrário. O diâmetro relativo é um limite superior do erro relativo de  $x$  em relação a um elemento arbitrário de  $X$ .

A distância  $q(X, Y)$  entre dois intervalos  $X$  e  $Y$  é definida como o máximo do valor absoluto das diferenças dos extremos dos intervalos, ou seja,  $\max\{|x_1-y_1|, |x_2-y_2|\}$ . Ela é sempre não negativa, sendo que só será nula se os intervalos forem iguais. A figura 2.3 ilustra a distância entre dois intervalos  $X$  e  $Y$ .

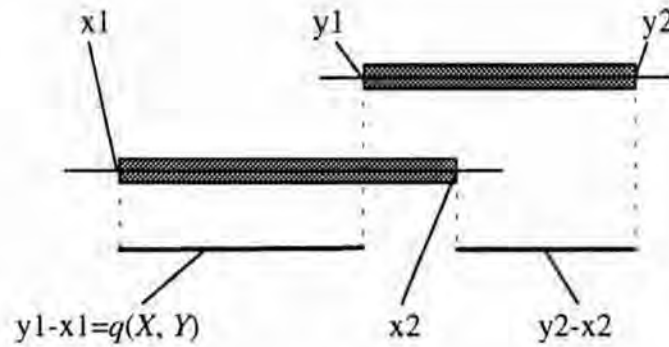


Fig 2.3 Distância entre dois intervalos X e Y

Como  $(\mathbb{R}, q)$  é um espaço métrico, os conceitos de convergência e continuidade podem ser introduzidos da maneira usual. A sequência  $\{X^{(k)}\}$  de intervalos converge para um intervalo se e somente se a sequência de seus limites converge para os limites de X, mais precisamente

$$\lim_{k \rightarrow \infty} X^{(k)} = X \Leftrightarrow \lim_{k \rightarrow \infty} q(X^{(k)}, X) = 0 \Leftrightarrow (\lim_{k \rightarrow \infty} x1^{(k)} = x1, \lim_{k \rightarrow \infty} x2^{(k)} = x2) \quad (2.16)$$

Dada uma função  $f: D \subset \mathbb{R} \rightarrow \mathbb{R}$  representando uma função elementar de valores reais, contínua em todos os intervalos fechados do domínio D. A Extensão Intervalar da função  $f(x)$  com argumentos intervalares é dada por (2.17). Desta forma as funções são inclusões monotônicas e, portanto, podem expressar o intervalo resultante da maioria das funções elementares em termos dos limites dos argumentos.

$$F(x) = \{ z \mid z = f(x), x1 \leq x \leq x2 \} \quad (2.17)$$

Esta função F, chamada extensão intervalar de f, deve ser aquela que se afasta o mínimo possível de  $f(X)$ . O erro obtido no cálculo de  $f(x)$  a partir do intervalo X é facilmente obtido através do diâmetro  $d(F(X)) = y2 - y1$ .

Consistindo na extensão das operações aritméticas, o cálculo de expressões na aritmética intervalar, junto com um conjunto de funções *standards*, forma a aritmética intervalar.

A aritmética intervalar utiliza um arredondamento especial, chamado arredondamento direcionado, o que significa que os resultados são arredondados para o menor e para o maior número de máquina que contém o resultado das operações, obtendo-se com isso um intervalo de máquina, com diâmetro minimal, no qual a solução se situa.

Uma distinção importante na aritmética intervalar é entre a imagem intervalar de uma função e a avaliação intervalar da função. A Imagem intervalar de uma função f, contínua no intervalo X, é definida como o intervalo limitado pelo mínimo da imagem de  $f(x)$  e pelo máximo da imagem de  $f(x)$ , sendo x um elemento do intervalo X.

$$I(f(X)) := [\min \{f(x)\}, \max \{f(x)\}] \text{ (para } x \in X) \quad (2.18)$$

A avaliação intervalar, ou a extensão intervalar de  $f$  à função de valores intervalares  $F$ , é obtida pela substituição de todos os operandos da função por intervalos contidos no domínio da definição de  $f$  e todas as operações por operações intervalares. Ou seja, dada  $f$  contínua em  $X$ , calcula-se a avaliação de  $f(X)$  aplicando as operações aritméticas em  $\mathbf{IR}$ . Observa-se que o resultado dependerá da forma como as operações foram efetuadas, uma vez que na aritmética intervalar a propriedade semi-distributiva é verificada pela inclusão e não pela igualdade como na aritmética real.

$$X * (Y + Z) \subseteq (X * Y) + (X * Z) \text{ (para } X, Y \text{ e } Z \text{ intervalos)} \quad (2.19)$$

A aritmética intervalar de máquina é relacionada com vários aspectos práticos e teóricos da ciência da computação que influenciam outros aspectos, como por exemplo: as estruturas algébricas de espaços numéricos que necessitam computações; resolução de problemas algébricos como sistemas lineares, inversão de matrizes, problemas de autovalores com alta exatidão; desenvolvimento de uma aritmética computacional ótima, isto é, que não exista nenhum número de máquina entre o valor calculado e o resultado exato, diferenciação automática e geração dos coeficientes de Taylor.

### 2.1.2.2 Aritmética intervalar estendida

Introduziu-se a aritmética intervalar real com "dicas" de implementação das regras nas tabelas 2.1 e 2.2. Para estas regras, exclui-se a divisão por zero. Agora, vai-se descrever como remover esta restrição pela definição de uma espécie de aritmética intervalar estendida<sup>7</sup>. É suficiente para este trabalho, estender a definição de intervalos. Estende-se o sistema numérico real pela soma de dois elementos, dois pontos ideais: mais e menos infinito resultando em (2.20).

$$\mathbf{R}^* = \mathbf{R} \cup \{-\infty\} \cup \{+\infty\} \quad (2.20)$$

Permite-se então que o ínfimo e o supremo de um intervalo real sejam estes pontos ideais, portanto o conjunto estendido dos intervalos reais é dado por (2.21). Ou seja,  $\mathbf{IR}^*$  é o conjunto dos intervalos reais calculados por todos os intervalos, cujos limites inferiores e superiores tendem ao infinito.

$$\mathbf{IR}^* = \mathbf{IR} \cup \{[-\infty, r] / r \in \mathbf{R}\} \cup \{[l, +\infty] / l \in \mathbf{R}\} \cup \{[-\infty, +\infty]\} \quad (2.21)$$

Na definição da divisão estendida, retira-se o zero dividindo o intervalo em dois, então, avalia-se a operação quando os extremos tendem a zero pelo lado negativo e pelo lado positivo, resultando na união de intervalos estendidos. Na verdade, o resultado pode ser representado por um ou dois intervalos estendidos. Para intervalos finitos reais, onde o zero pertence ao denominador, a divisão estendida é dada por (2.22).

<sup>7</sup>Detalhes em [ADA93, KUL83]



$$X/Y := \begin{cases} [-\infty, +\infty] & \text{Se } x_1 < 0 < x_2 \text{ ou } X=0 \text{ ou } Y=0 \\ [x_2/y_1, +\infty] & \text{Se } x_2 \leq 0 \text{ e } y_1 < y_2 = 0 \\ [-\infty, x_2/y_2] \cup [x_2/y_1, +\infty] & \text{Se } x_2 \leq 0 \text{ e } y_1 < 0 < y_2 \\ [-\infty, x_2/y_2] & \text{Se } x_2 \leq 0 \text{ e } 0 = y_1 < y_2 \\ [-\infty, x_1/y_1] & \text{Se } 0 \leq x_1 \text{ e } y_1 < y_2 = 0 \\ [-\infty, x_1/y_1] \cup [x_1/y_2, +\infty] & \text{Se } 0 \leq x_1 \text{ e } y_1 < 0 < y_2 \\ [x_1/y_2, +\infty] & \text{Se } 0 \leq x_1 \text{ e } 0 = y_1 < y_2 \end{cases} \quad (2.22)$$

No primeiro caso da definição (2.22) não se pode decidir quando  $X/Y$  com  $x \in X$  e  $y \in Y$  tende para menos ou mais infinito. Então o resultado é todo o eixo real,  $[-\infty, +\infty]$ . Todos os outros casos são calculados, dividindo em dois intervalos que tendem a zero. No algoritmo da divisão estendida, essa muitas vezes é seguida por uma intersecção com algum intervalo finito. O resultado após a intersecção é um conjunto vazio, ou um ou dois intervalos finitos. Neste sentido, a aritmética estendida é uma ferramenta que pode gerar e tratar temporariamente intervalos infinitos.

É necessário, ainda, estender a subtração de um real por um intervalo que tenha pelo menos um extremo infinito. O real pode ser considerado um intervalo degenerado. Esta definição é dada por (2.23).

$$x-Y := \begin{cases} [-\infty, +\infty] & \text{Se } Y = [-\infty, +\infty] \\ [-\infty, x-y_1] & \text{Se } Y = [y_1, +\infty] \\ [x-y_2, +\infty] & \text{Se } Y = [-\infty, y_2] \end{cases} \quad (2.23)$$

### 2.1.2.3 Aritmética intervalar complexa

Neste ítem, são introduzidos intervalos complexos<sup>8</sup>, isto é, intervalos no plano complexo. Serão apenas resumidos os tópicos necessários para um entendimento básico deste capítulo da Matemática Intervalar.

Sejam  $X_{re}, X_{im} \in \mathbf{IR}$ . Então o conjunto descrito por (2.24) é chamado de intervalo complexo, onde  $i$  representa a unidade imaginária. O conjunto dos intervalos complexos é anotado por  $\mathbf{IC}$ . Um intervalo complexo  $X$  é dito ser degenerado ou um ponto intervalar se tanto a parte real  $X_{re}$  quanto a parte imaginária  $X_{im}$  são degenerados. Um intervalo complexo pode ser escrito como um par ordenado  $(X_{re}, X_{im})$  de intervalos reais. Usam-se intervalos retangulares com lados paralelos aos eixos coordenados, mas um intervalo complexo pode ser também definido como um disco no plano complexo, sendo para tanto dado o ponto central (ponto médio) e o seu raio. Esta aritmética é conhecida como aritmética intervalar circular.

$$X := X_{re} + i X_{im} = \{ x = x_{re} + i x_{im} / x_{re} \in X_{re} \text{ e } x_{im} \in X_{im} \} \quad (2.24)$$

<sup>8</sup>Detalhes em [ADA93, KUL83, MOO79]

Sejam  $X$  e  $Y$  intervalos complexos. As relações de igualdade ou inclusão interna são válidas somente se o forem tanto para a parte real quanto para a parte imaginária de seus operandos como mostra a fórmula (2.25).

$$X \# Y \Leftrightarrow (X_{re} \# Y_{re} \text{ e } X_{im} \# Y_{im}) \# \in \{ =, \overset{\circ}{\subseteq}, \subseteq \} \quad (2.25)$$

As operações de reticulados para intersecção e união de dois intervalos complexos podem ser definidas pela redução dos correspondentes operadores para as partes reais e imaginárias, como mostra (2.26).

$$X \# Y = (X_{re} \# Y_{re} + i (X_{im} \# Y_{im})) \# \in \{ \cap, \cup \} \quad (2.26)$$

A figura 2.4 é uma ilustração gráfica de uma intersecção de intervalos.

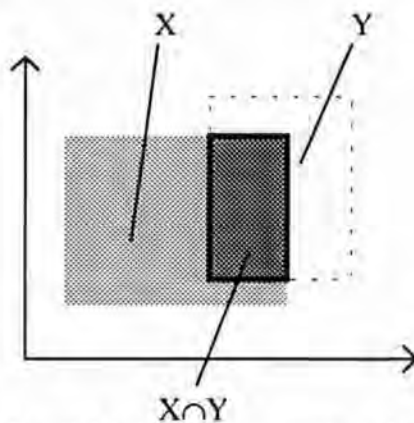


Fig 2.4 Intersecção de intervalos Complexos

Assim como para intervalos reais, pode ser definida a distância em  $\mathbf{IC}$  e, é fácil de verificar que  $\mathbf{IC}$  providencia uma distância apropriada, a qual é um espaço métrico completo, então conceitos de convergência e continuidade também podem ser transferidos para o espaço dos intervalos complexos.

Sejam  $X$  e  $Y$  intervalos complexos, de acordo com as regras padrões de aritmética complexa, definem-se as operações aritméticas complexas.

$$X + Y = (X_{re} + Y_{re}) + i (X_{im} + Y_{im}) \quad (2.27)$$

$$X - Y = (X_{re} - Y_{re}) + i (X_{im} - Y_{im}) \quad (2.28)$$

$$X * Y = (X_{re} * Y_{re} - X_{im} * Y_{im}) + i (X_{re} * Y_{im} - X_{im} * Y_{re}) \quad (2.29)$$

$$X / Y = \frac{(X_{re} * Y_{re} + X_{im} * Y_{im})}{Y_{re}^2 + Y_{im}^2} + i \frac{(X_{im} * Y_{re} - X_{re} * Y_{im})}{Y_{re}^2 + Y_{im}^2} \quad (2.30)$$

A definição da divisão  $X/Y$  está restrita a intervalos em que o zero não pertença a  $Y_{re}^2 + Y_{im}^2$ . A divisão é avaliada usando a função intervalar elementar quadrática para garantir que  $0 \notin Y_{re}^2 + Y_{im}^2$ , para  $Y$  que não contenha o zero.

Existe uma diferença importante entre as definições das operações elementares entre intervalos reais e complexos. A imagem contínua de um intervalo complexo não é, necessariamente, outro intervalo complexo. As operações definidas pela regra geral resultam em um intervalo real. Entretanto, para intervalos complexos o resultado pode não ser um intervalo complexo, isto é, pode não ser um retângulo com lados paralelos aos eixos coordenados. Para se ter um intervalo complexo, tem-se que tomar o menor retângulo que inclua o intervalo com lados paralelos aos eixos. Desta forma, pode-se implementar facilmente as regras para aritmética intervalar complexa. Infelizmente, perde-se informação pela superestimativa do intervalo. Este efeito é conhecido como efeito envolvente (*wrapping*).

A aritmética de intervalos complexos estendidos pode ser definida aplicando-se as definições estendidas na parte real e na parte imaginária.

#### 2.1.2.4 Vetores e matrizes de Intervalos

Um vetor intervalar é um vetor cujos elementos são intervalos. Uma matriz intervalar é uma matriz cujos elementos são intervalos. O conjunto de todos os vetores  $n$ -dimensionais de intervalos reais ou complexos são escritos por  $\mathbf{IR}^n$  e  $\mathbf{IC}^n$ , respectivamente (definidos por (2.31)). Da mesma maneira,  $\mathbf{IR}^{n \times m}$  e  $\mathbf{IC}^{n \times m}$  indicam os conjuntos das matrizes intervalares de reais e complexos de ordem  $n \times m$  (definidas por (2.32)).

$$X := (X_i)_{i=1, \dots, n} = (X_1, \dots, X_n)^T \quad \text{para } X \in \mathbf{IR}^n \text{ e } \mathbf{IC}^n \quad (2.31)$$

e

$$A := (A_{ij})_{i=1, \dots, n \quad j=1, \dots, m} = \begin{bmatrix} A_{11} & \dots & A_{1m} \\ \vdots & & \vdots \\ A_{n1} & \dots & A_{nm} \end{bmatrix} \quad \text{para } A \in \mathbf{IR}^{n \times m} \text{ e } \mathbf{IC}^{n \times m} \quad (2.32)$$

Um vetor de intervalos reais pode ser interpretado como o conjunto de pontos no espaço  $n$ -dimensional limitado por um paralelepípedo com lados paralelos aos eixos coordenados (figura 2.5). Por esta razão, algumas vezes é usado o sinônimo "caixas" para vetores de intervalos.

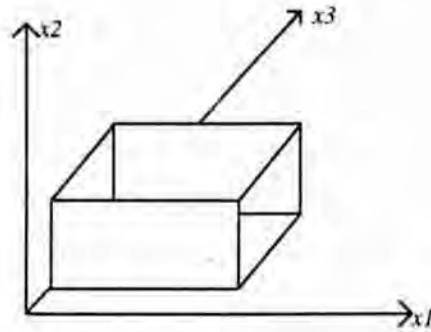


Fig 2.5 Vetor tridimensional de intervalos reais ou caixa

As relações  $=$ ,  $\overset{\circ}{\subseteq}$  e  $\subseteq$  são definidas componente a componente, por exemplo a relação de inclusão interna é definida como  $X \overset{\circ}{\subseteq} Y \Leftrightarrow X_i \overset{\circ}{\subseteq} Y_i$ , para todo  $i$  e  $X, Y \in \mathbf{IR}^n$ , por outro lado a relação de subconjunto próprio é dada por  $X \subset Y \Leftrightarrow X \subseteq Y$  e  $X \neq Y$ .

O ponto médio e o diâmetro de um vetor de intervalos ou de uma matriz de intervalos também são definidos componente a componente, ou seja,  $m(X) = (m(X_i))$  e  $d(A) = (d(A)_{ij})$  para  $X \in \mathbf{IR}^n$  e  $A \in \mathbf{IR}^{n \times m}$ .

A norma máxima é estendida para vetores e matrizes de intervalos reais por (2.33) e (2.34) respectivamente. E, finalmente, introduzem-se notações de diâmetro relativo (2.35) e máximo absoluto (2.36) para estes componentes de vetores de intervalos reais.

$$\|X\|_{\infty} = \max_{1 \leq i \leq n} |X_i| \quad \text{para } X \in \mathbf{IR}^n \quad (2.33)$$

$$\|A\|_{\infty} = \max_{1 \leq i \leq n} \sum_{j=1}^m |A_{ij}| \quad \text{para } A \in \mathbf{IR}^{n \times m} \quad (2.34)$$

$$d_{\infty}(X) = \max_{1 \leq i \leq n} d(X_i) \quad \text{para } X \in \mathbf{IR}^n \quad (2.35)$$

$$d_{rel, \infty}(X) = \max_{1 \leq i \leq n} d_{rel}(X_i) \quad \text{para } X \in \mathbf{IR}^n \quad (2.36)$$

Mais detalhes sobre vetores e matrizes de intervalos são dados no capítulo 4, quando da definição de métodos intervalares para solução de sistemas lineares, ou podem ser encontrados em [NEU90, ALE83].



### 2.1.3 Produto escalar exato ou ótimo

Os produtos escalares  $\nabla$ -,  $\Delta$ - e  $O$ -, são distintos uns dos outros pelo tipo de arredondamento que os finaliza. Sejam  $x := \langle x_1, x_2, \dots, x_n \rangle$  e  $y := \langle y_1, y_2, \dots, y_n \rangle$  vetores de comprimento  $n$  cujos elementos pertencem a  $F$ . Seja  $[\ ] \in \{ \nabla, \Delta, O \}$ . A definição matemática de cada produto escalar<sup>9</sup> é:

$$x[\cdot]y = [\ ](x_1y_1 + x_2y_2 + \dots + x_ny_n) \quad (2.37)$$

Observe que a máxima exatidão é garantida pois todas as operações são realizadas em  $R$  e o arredondamento  $[\ ]$  ocorre apenas em uma ocasião, no final da operação.

Em resumo, é o produto escalar com exatidão máxima juntamente com outras técnicas de aritmética avançada que fornecem a alta exatidão das operações dos espaços. Os produtos escalares mostram uma nova capacidade para análise numérica com respeito ao controle automático do erro, isto é, uma verificação final e correção dos resultados calculados. Pode-se dizer que a aritmética intervalar traz garantia nos cálculos, enquanto que os três produtos escalares proporcionam alta exatidão. Ambas características são desejáveis e não podem ser confundidas.

Os espaços computacionais (de máquina) definem várias operações que devem ser implementadas a fim de servir de suporte a aritmética de alta exatidão. Todas as operações devem estar munidas dos três tipos de arredondamento. As quinze operações mostradas na figura 2.1 são suficientes para a implementação de todos os espaços computacionais.

A aritmética computacional é organizada em três níveis de implementação. O primeiro nível, chamado aritmética básica, trata com a aritmética computacional elementar aumentada pelo produto escalar.

O segundo nível é considerado como aritmética computacional avançada, nele são implementadas as operações dos demais espaços computacionais. O terceiro nível trata a validação de processos, incluindo a capacidade de linguagens fonte para expressarem convenientemente a avaliação de expressões como funções racionais com máxima exatidão.

## 2.2 Verificação Automática do Resultado

O cálculo numérico com a verificação de resultados é de fundamental significância para muitas aplicações; por exemplo, para simulação e modelagem matemática. Modelos que são freqüentemente desenvolvidos por métodos heurísticos podem apenas ser refinados sistematicamente, se os erros computacionais puderem ser excluídos completamente.

<sup>9</sup>Veja em [KUL81, KUL83, BOH91a, ADA93, IMACS91]

Na análise clássica do erro em algoritmos numéricos, o erro em cada operação em ponto-flutuante é estimado. Na verdade, a possibilidade do resultado estar errado não é normalmente considerada. Do ponto de vista matemático, o problema da correção dos resultados é de grande importância para garantir a alta velocidade computacional atingida atualmente. Isto torna possível ao usuário distinguir entre inexatidão nos cálculos e as reais propriedades do modelo.

Em contraste com a velha computação científica, que podia ser caracterizada por multiplicações e divisões intensivas, contadores e contabilistas insistiam com adições e subtrações de longas colunas de números de várias magnitudes. Erros nestas adições e subtrações calculadas intensivamente não eram tolerados, e vários métodos de validação (tal como a contabilidade que era feita duas vezes) eram desenvolvidos para assegurar que as respostas estivessem corretas até nos centavos. No sentido de tornar possível o manuseio de milhões de adições e subtrações com o máximo de exatidão, é evidente que as capacidades da aritmética de ponto-flutuante têm que ser aumentadas das habilidades de um contador paciente. Dadas as possibilidades atuais, não há razão para que isto não possa ser feito em um *chip* simples.

O computador foi inventado para fazer o trabalho complicado para o homem. A evidente discrepância entre o poder computacional e o controle dos erros computacionais sugere também que se coloque o processo de estimativa de erro dentro do computador. Este poder computacional tem tido sucesso em vários problemas básicos de análise numérica e em muitas aplicações. Para fazer isto, o computador tem que ser aritmeticamente mais poderoso que o comum. Na aritmética de ponto-flutuante, a maioria dos erros ocorre em acumulação, isto é, pela execução de uma seqüência de adições e subtrações. Por outro lado, multiplicações e divisões em ponto-flutuante são operações de relativa estabilidade.

É evidente que com o produto escalar ótimo, todas as operações entre números de ponto-flutuante e, em particular, todas as operações em vetores ou matrizes com componentes reais podem ser realizadas exatamente ou com máxima exatidão, isto é, com apenas um arredondamento simples em cada componente.

No sentido de adaptar o computador para o controle automático de erros, sua aritmética deve ser estendida com um outro elemento. Todas as operações com números de ponto-flutuante, que são a adição, a subtração, a divisão, a multiplicação e o produto escalar ótimo de vetores em ponto-flutuante devem ser supridos com os arredondamentos direcionados, isto é, arredondamentos para o número de máquina anterior (para baixo) e posterior (para cima). Uma aritmética intervalar para números reais e complexos em ponto-flutuante, como também para vetores e matrizes com componentes reais e complexos em ponto-flutuante pode ser construída com estas operações. Um intervalo é representado no computador por um par de números em ponto-flutuante. Isto descreve a continuidade dos números reais que está limitada por estes dois números em ponto-flutuante. As operações sobre dois intervalos no computador resultam de operações sobre

dois extremos dos operandos intervalares. Assim, o cálculo do extremo inferior do resultado intervalar é arredondado para baixo, e o cálculo do extremo superior é arredondado para cima. O resultado certamente contém todos os resultados da operação aplicando individualmente os elementos do primeiro e do segundo operando intervalar.

Uma grande vantagem da verificação automática de resultados é que o próprio computador pode rapidamente estabelecer se o cálculo realizado é ou não correto e útil. Neste caso, o programa pode escolher um algoritmo alternativo ou repetir o processo usando uma maior precisão.

Técnicas similares de verificação automática de resultados podem ser aplicadas para muitas outras áreas<sup>10</sup> de problemas algébricos, tal como a solução de sistemas não lineares de equações, o cálculo de raízes, a obtenção de autovalores e de autovetores de matrizes, problemas de otimização, etc. Em particular, a validade e a alta exatidão da avaliação de expressões aritméticas e de funções no computador está incluída. Estas rotinas também funcionam para problemas com dados intervalares.

### 2.2.1 Princípios da verificação numérica

As técnicas mencionadas aqui para o cálculo numérico com verificação automática de resultado utilizam o produto escalar ótimo e a aritmética intervalar como ferramentas essenciais, além da aritmética de ponto-flutuante simples. A aritmética intervalar permite o cálculo de extremos seguros para as soluções de um problema. Obtém-se alta exatidão por meio do produto escalar ótimo. A combinação de ambos os instrumentos é realmente um grande avanço na análise numérica. Uma aplicação ingênua da aritmética intervalar certamente levará a extremos confiáveis. Entretanto, estes podem ser tão grandes que eles são realmente inúteis. Este efeito era observado há vinte anos atrás por muitos analistas numéricos, que freqüentemente criavam rejeições a esta ferramenta útil, necessária e inocente.

Um método para controle do erro, genérico e bastante simples foi proposto por Kulisch, em [KUL81]. O método tem como idéia central equipar o computador com recursos que possibilitem o controle e a validação do processo computacional. Os requisitos básicos são:

- a) Implementação das operações de adição, subtração, multiplicação e divisão sobre os números em ponto-flutuante com exatidão máxima;
- b) Implementação de produto escalar de dois vetores com exatidão máxima;
- c) Implementação das operações aritméticas básicas e do produto escalar no espaço dos intervalos e dos vetores de intervalos com a melhor exatidão possível.

---

<sup>10</sup>ver [ADA93]

A partir destas ferramentas, torna-se possível a utilização de técnicas intervalares de correção residual, as quais têm como característica básica permitir a recuperação da informação perdida no arredondamento, garantindo que as computações sejam realizadas com exatidão máxima.

A combinação desta técnica com a teoria do ponto-fixo resulta no método de verificação automática. Neste caso a verificação da existência e unicidade da solução de problemas algébricos tradicionais, como por exemplo inversão de matrizes e solução de sistemas de equações lineares pode ser realizada pelo próprio computador.

Validação deriva da exatidão máxima com garantia para computações científicas. Validação de computações para uma grande classe de problemas numéricos torna-se possível pela aritmética computacional avançada. Exatidão máxima é obtida em função do produto escalar com um processo numérico especial chamado correção defeituosa. Garantia e limites de erro são obtidos por técnicas intervalares. Este processo todo estabiliza certos algoritmos numéricos, os quais avaliam expressões racionais como a operação aritmética de adição.

Muitos algoritmos de verificação numérica são baseados nos teoremas de ponto-fixo com respeito a conjuntos intervalares. Um dos mais conhecidos e utilizado é o Teorema de Brouwer.

**Teorema de Ponto-Fixo de Brouwer**<sup>11</sup>: Sendo  $f: \mathbf{R}^n \rightarrow \mathbf{R}^n$  um mapeamento contínuo e  $X \subseteq \mathbf{R}^n$  um conjunto fechado, convexo e limitado. Se  $f(X) \subseteq X$ , então  $f$  tem no mínimo um ponto-fixo<sup>12</sup>  $x^*$  em  $X$ .

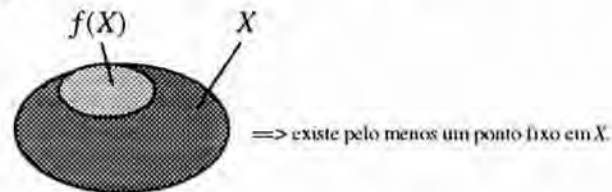


Fig. 2.6 - Teorema de Ponto-Fixo de Brouwer.

Seja  $X \in \mathbf{IR}^n$  um vetor intervalar de máquina. Como uma caixa no espaço  $n$ -dimensional,  $X$  satisfaz as condições do teorema de ponto-fixo de Brouwer. Suponha que se possa encontrar uma caixa com  $f(X) \subseteq X$ . Então,  $X$  contém pelo menos um ponto-fixo  $x^*$  de  $f$ . As suposições anteriores valem se  $f$  for substituído pela avaliação intervalar de ponto-fixo  $f_\phi$ , pois  $f_\phi(X) \subseteq X$  implica  $f(X) \subseteq X$ , desde que  $f_\phi(X)$  é um superconjunto<sup>13</sup> de  $f(X)$ .

<sup>11</sup>[ADA93]

<sup>12</sup>  $x$  é dito um ponto fixo de  $f$ , quando  $x=f(x)$ .

<sup>13</sup> contém



**Teorema de Ponto-Fixo de Brouwer Modificado:** Sendo  $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$  um mapeamento contínuo e  $X \subseteq \mathbb{R}^n$  um conjunto fechado, convexo e limitado. Se  $f(X) \subset X$ , então  $f$  tem um único ponto-fixo  $x^*$  em  $X$ .

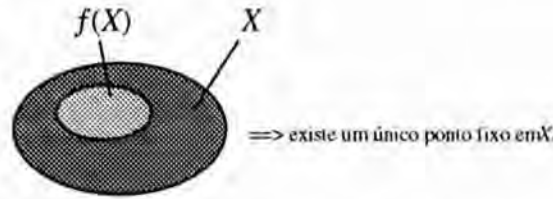


Fig.2.7- Teorema de Ponto-Fixo de Brouwer Modificado.

Observe que  $f(X)$  está no interior de  $X$  sem tocar a fronteira de  $X$ .

Note que esses dois teoremas são muito parecidos e de fácil aplicação em algoritmos, pois só trabalham com os extremos do intervalo.

Uma das aplicações do Teorema de Brouwer é no projeto de algoritmos com verificação dos resultados. Primeiro, deve-se achar uma fórmula de iteração de ponto-fixo  $f(x) = x$  equivalente ao problema original e, toma-se  $f_0$  como uma extensão de máquina da fórmula de iteração intervalar de um método numérico (Newton, Gauss-Seidel, etc). Inicializa-se o algoritmo com alguma aproximação da solução  $X^{(0)}$ ; então, se para alguma iteração, com  $k \geq 0$ , for encontrado  $X^{(k+1)} \subseteq X^{(k)}$ , fica assim provado que o problema original tem no mínimo uma solução  $x^*$  contida em  $X^{(k)}$ .

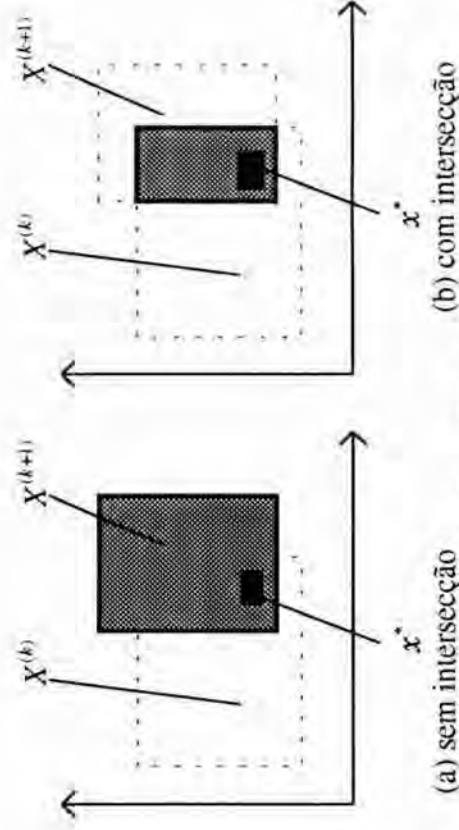
$$X^{(k+1)} = f_0(X^{(k)}), \text{ para } k = 0, 1, 2, \dots \quad (2.38)$$

Unindo a aritmética intervalar aos teoremas, têm-se os Métodos de Inclusão. Com o objetivo de garantir a existência e/ou a unicidade da solução de um algoritmo, podem-se distinguir dois tipos de métodos de inclusão: *a priori* e *a posteriori*, com relação a aproximação inicial.

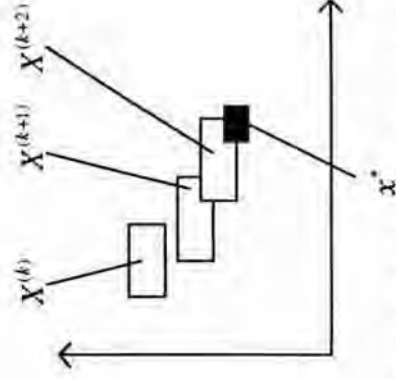
Para o **Método a priori**, a aproximação inicial já inclui o ponto-fixo procurado. Neste caso, a fórmula de iteração (2.38) deve ser modificada por uma fórmula de intersecções sucessivas (2.39), onde  $X^{(k)} \in \mathbb{IR}^n$  é um vetor intervalar de máquina na  $k$ -ésima iteração e  $f_0$  uma extensão de máquina da fórmula de iteração intervalar.

$$X^{(k+1)} = f_0(X^{(k)}) \cap X^{(k)}, \text{ para } k = 0, 1, 2, \dots \quad (2.39)$$

O algoritmo é parado se ocorrerem duas iterações sucessivas de mesmo valor ou se um número máximo de iterações for excedido. Veja na figura 2.8, onde há duas iterações sucessivas. As partes que estão pontilhadas são eliminadas pelo algoritmo, ficando somente a caixa mais escura, que contém o ponto-fixo  $x^*$ . Como é mostrado no segundo gráfico, usando uma variação do método *a priori*, com intersecção a iteração  $X^{(k+1)}$ , é diminuída, garantindo assim uma maior velocidade de convergência do algoritmo, comparado ao método sem intersecção do primeiro gráfico.

Fig. 2.8 - Método de Inclusão *a priori*.

No **Método *a posteriori***, o ponto-fixo procurado não está, necessariamente, contido na aproximação inicial. Aqui, espera-se que as iterações se aproximem cada vez mais do ponto-fixo, até o incluir. Quanto melhor for a aproximação inicial (mais próxima do ponto-fixo), mais rápida será a convergência do algoritmo. Veja a figura 2.9, que mostra o comportamento deste método.

Fig. 2.9 - Método de Inclusão *a posteriori* sem inflação.

Na prática as iterações vão se aproximando do ponto-fixo  $x^*$ , mas muitas vezes não conseguem incluí-lo. Um artifício simples é utilizado para resolver esta limitação. Antes de iniciar uma nova iteração, esta é aumentada em seus extremos por uma inflação<sup>14</sup>. Neste caso, a fórmula de iteração (2.39) é modificada para

$$\left. \begin{aligned} X^{(k)} &= X^{(k)} \bowtie \epsilon \\ X^{(k+1)} &= f_0(X^{(k)}) \end{aligned} \right\} \text{ para } k = 0, 1, 2, \dots \quad (2.40)$$

<sup>14</sup> Fórmula da inflação:  $X \bowtie \epsilon := X + [-\epsilon, +\epsilon].d(X)$  Se  $d(X) \neq 0$ .

A figura 2.10 mostra os efeitos do método *a posteriori* com inflação  $\varepsilon$ .

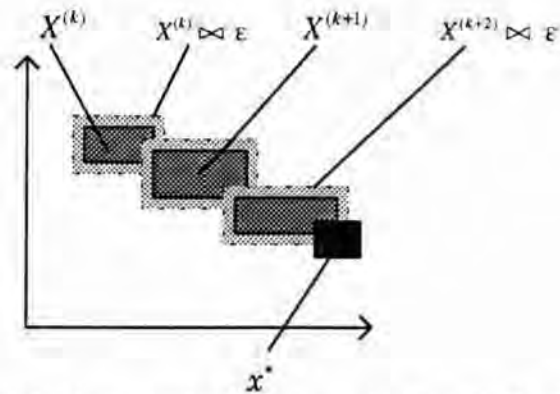


Fig.2.10 - Método de Inclusão *a posteriori* com inflação  $\varepsilon$ .

Métodos que usam os teoremas de ponto-fixo podem ser modificados para provar a unicidade de um ponto-fixo, com o teorema de Brouwer modificado. Para tanto, usa-se a fórmula (2.39) com critério de parada.

$$X^{(k+1)} \stackrel{\circ}{\subseteq} X^{(k)}, \text{ para } k=0, 1, 2, \dots \quad (2.41)$$

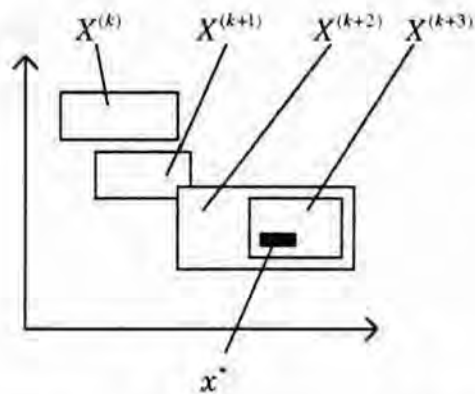


Fig.2.11 - Critério de parada para provar a unicidade de uma solução.

Na iteração  $X^{(k+3)}$  existe apenas uma única solução, pois,  $X^{(k+3)} \stackrel{\circ}{\subseteq} X^{(k+2)}$  (critério de parada).

## 2.2.2 Aplicações da verificação automática

Soluções confiáveis têm sido computadas para um certo número de problemas de engenharia e da computação científica, através de rotinas que além de resolver o problema verificam automaticamente o resultado através de tipos de dados básicos da análise numérica e do suporte para verificação automática fornecido pelas linguagens de programação Pascal-XSC, ACRITH-XSC e C-XSC. Muitos destes problemas quando

resolvidos por métodos pontuais clássicos não produzem solução aceitável. GMM (*Gesellschaft für Angewandte Mathematik und Mechanik*) e IMACS (*International Association for Mathematics and Computers in Simulation*) têm realizado simpósios no tema de "*Computer Arithmetic and Scientific Computation with Automatic Result Verification*" (aritmética computacional e computação científica com verificação automática de resultados) regularmente desde 1980. Muitos eventos sobre este assunto têm acontecido. As aplicações pertencem a vários campos científicos. Como ilustração são mencionados os seguintes exemplos: turbinas de alta velocidade, cálculos de filtros, fluxo de plasma em um canal retangular, vibrações no equipamento de um automóvel, supercondutividade de alta temperatura, infiltração de poluição em um meio líquido, soluções periódicas em química, modelagem geométrica, dimensionamento de esgotos, quadratura verificada em engenharia de transporte químico, direcionamento de pilhas não-lineares, rejeição de certas soluções "caóticas" no problema tri-dimensional de mecânica celestial, desenvolvimento de soluções da equação de Schrödinger em funções de onda, cálculo de fluxos magnetohidrodinâmicos para grandes números de Hartmann, propriedades óticas de células de cristal líquido, simulação de diodos semicondutores, otimização de circuitos VLSI, oscilações rotacionais, métodos geométricos para sistemas CAD, fratura de viga central e robótica.

De acordo com a Teoria Clássica do Controle, faz-se uso de um sistema linear de equações diferenciais ordinárias do tipo  $y'=Ay$ , onde  $A$  representa uma matriz constante. Em uma simulação realística matemática, intervalos devem ser usados para representar os elementos constantes de  $A$ . Nas aplicações, deseja-se obter resultados com estabilidade assintótica de  $y'=Ay$  e o correspondente grau de estabilidade. Para este propósito, quatro métodos semelhantes podem ser desenvolvidos de modo que a matriz de intervalos representada por  $A$  seja diretamente tratada. Conseqüentemente, não há necessidade de uma utilização das características polinomiais e suas raízes, i.e., das quantidades que deveriam ser computadas previamente para uma análise de estabilidade. Um exemplo clássico é o controle de um ônibus urbano eletro-mecânico, por causa de sua variedade de estados operacionais.

A verificação automática do resultado também pode ser usada na investigação numérica do fluxo magnetohidrodinâmico em um duto retangular e na análise de erro do cálculo tradicional da solução derivada. Valores arbitrários dos parâmetros de fluxo são introduzidos, eles são os números de Hartmann e a taxa de condução da parede. O problema típico de perturbação é solucionado e analisado por meio de aritmética intervalar e verificação de métodos de inclusão (E-Methods), suportados pela linguagem de programação para computação científica, como as extensões XSC. Além disso, a análise de erro do cálculo tradicional, como utilizado para fluxo, mostra que existe uma falta de confiabilidade dos resultados numéricos publicados deste problema físico, e isto até mesmo para os números de Hartmann  $M < 1000$ . Estes resultados indicam que a confiabilidade numérica dos resultados deveriam no mínimo ser verificadas por algum cálculo, pelo emprego de um controle de erros através do uso de uma aritmética exata de ponto-flutuante e métodos de inclusão.



## 2.3 Linguagens para Computação Científica

Linguagens de programação tradicionais como Algol, Fortran, Pascal, Módulo e C usam apenas dois tipos de dados numéricos elementares e dispõem somente de operações associadas a estes tipos de dados. Estes tipos de dados elementares são os inteiros e os reais. Mesmo quando operações com números complexos estão disponíveis, elas são baseadas nas operações sobre reais. Todas as operações sobre intervalos devem ser definidas e implementadas pelo usuário na forma de subrotinas. O uso destas operações por unidades numéricas mais elaboradas como vetores, matrizes, números complexos, vetores e matrizes de complexos e intervalos sobre todos estes conjuntos é uma tarefa complicada em si mesma. Cada ocorrência de uma operação em um algoritmo ocasiona uma chamada de rotina, complicada e dispendiosa em termos de consumo de tempo, o que é notacionalmente inconveniente.

Acima de tudo, este método de definir operações avançadas estabelece a definição vertical de aritmética computacional. A fim de tornar mais claro, serão consideradas duas matrizes reais  $a=(a_{ij})$  e  $b=(b_{ij})$ . Em todas as linguagens tradicionais, a multiplicação de  $a$  com  $b$  é programada usando três ninhos de *loops*, como ilustrado por (2.42). Isto mostra que a multiplicação de matrizes é reduzida a operações reais pela forma da definição vertical.

```

For i=1 to n do
  For j=1 to n do
    For k=1 to n do
      s= s+ a[i,k] * b[k,j];

```

(2.42)

Uma forma simples de se eliminar a definição vertical das operações é permitir a notação de operadores para as operações definidas pelo semimorfismo. Linguagens tradicionais admitem tal notação de operadores somente para tipos inteiros e reais. Algumas ainda permitem para complexos. O sinal operacional nas expressões destes tipos são escritos com os símbolos matemáticos usuais, ou seja, +, -, \*, /. Desta forma se introduz, também, a sobrecarga de nomes de operadores, funções e rotinas. Com isto se facilita a leitura de programas e a própria programação.

Portanto, a única maneira razoável e sensível de se evitar esta dificuldade consiste na extensão destas linguagens, de modo que os usuais símbolos matemáticos para o correspondente tipo de dado pré-definido seja fornecido para todas as operações com espaços vetoriais reais e complexos e os correspondentes espaços intervalares. Se  $A$ ,  $B$ ,  $C$  são variáveis que representam matrizes reais, então a multiplicação matricial é denotada simplesmente por  $C := A * B$ . Aqui, o sinal de operação  $*$  chama a nível de máquina uma operação elementar possivelmente realizada no *hardware* que calcula cada componente do produto matricial com a máxima exatidão. Esta "notação de operador" simplifica consideravelmente a programação para praticamente todas as operações aritméticas

simples. Os programas tornam-se mais fáceis de ler, de testar e são assim essencialmente mais confiáveis. Entretanto, acima de tudo, as operações são de máxima exatidão.

### **2.3.1 Características**

Além de um grande número de tipos de dados aritméticos pré-definidos e as correspondentes operações com máxima exatidão, as linguagens-XSC incluem entre outros os seguintes conceitos de linguagem, que não estavam presentes nas versões básicas destas linguagens. Um conceito de módulo, um conceito de operador, funções e operadores com tipos de resultados gerais, sobrecarga de funções, procedimentos, e operadores, sobrecarga do operador de tarefa como também *read* e *write*, *arrays* dinâmicos, acesso a *subarrays*, controle do arredondamento pelo usuário e avaliação exata de expressões. Por exemplo, estes conceitos de linguagem, módulos e operadores para aritmética complexa, aritmética vetorial e matricial, aritmética racional, aritmética de dupla ou múltipla precisão, ou ainda aritmética intervalar podem ser desenvolvidos nestas linguagens. Expressões aritméticas e algoritmos com estes tipos de dados podem ser anotados em notação matemática comum. Isto simplifica muito a programação.

#### **2.3.1.1 Notação matemática das expressões**

As linguagens XSC têm a si incorporadas, o conceito de operador universal e tipo de resultado arbitrário, obtendo-se desta forma uma programação mais fácil por possibilitar que o programador defina funções e operações com tipo de resultado arbitrário.

Elas permitem, também, a sobrecarga de identificadores de rotinas, funções e de operadores. Sobrecarga de funções e rotinas são distinguidas pelo número, ordem e tipo de seus parâmetros. O tipo de resultado não é usado para essa distinção, uma vez que, o próprio programa identifica o tipo de dado resultante, geralmente o maior tipo de dado envolvido na operação.

#### **2.3.1.2 Facilidades de Programação**

Novas características de programação foram introduzidas nas linguagens XSC com o intuito de as tornar mais fáceis de serem utilizadas em computações técnicas e científicas, com a introdução dos tipos de dados abstratos da análise numérica. Consiste em introduzir os espaços numéricos como dos complexos, dos intervalos, dos intervalos complexos e dos correspondentes tipos de vetores e matrizes. Para ilustrá-los, a figura 2.12 contém as definições destes tipos de dados em Pascal.

O conceito de módulo permite ao programador separar grandes programas em módulos, desenvolvê-los e compilá-los separadamente. O controle da sintaxe e da semântica pode ser carregado fora dos limites do módulo; permite também a criação de

novos tipos de dados, a definição das operações e funções que os manipulam tudo dentro de um módulo.

```

TYPE   dimtype = 1..dim;
       complex = record re,im: real;
       interval = record inf,sup: real;
       cinterval = record ire,iim: interval;
       rvector = array [dimtype] of real;
       rmatrix = array [dimtype] of rvector;
       cvector = array [dimtype] of complex;
       cmatrix = array [dimtype] of cvector;
       ivector = array [dimtype] of interval;
       imatrix = array [dimtype] of ivector;
       cvector = array [dimtype] of cinterval;
       cmatrix = array [dimtype] of cvector;
END;
```

Fig. 2.12 Tipos de dados disponíveis em Pascal-XSC

No que diz respeito ao tratamento de *arrays*, não só foi introduzido o conceito de *arrays* dinâmicos, que possibilita ao programador implementar algoritmos independentemente do tamanho dos *arrays* usados mas também o acesso a *subarrays*, como acessar diretamente a linha ou coluna de uma matriz.

O conceito de *string* é introduzido nas linguagens para manusear *strings* de caracteres de tamanho variável.

### 2.3.1.3 Suporte para métodos numéricos com verificação automática do resultado

Para as linguagens XSC, o conjunto de operadores para números reais é estendido pelos operadores com arredondamento direcionado < e > com os símbolos das operações {+,-,\*,/}. Então, os símbolos conjugados denotam as operações com arredondamento direcionado para cima e para baixo.

Elas também têm a si incorporadas a aritmética de máxima exatidão e a aritmética intervalar, com o que é possível desenvolver algoritmos, baseados em inclusões monotônicas, os quais possibilitam a concretização dos métodos numéricos com verificação automática do resultado.

### 2.3.1.4 Alta exatidão

As rotinas proporcionam soluções com alta exatidão, a qual não pode ser obtida por métodos convencionais. Esta alta exatidão significa que a tolerância da solução exata do problema é obtida diferindo somente no último dígito da mantissa.

A implementação de inclusão de algoritmos com alta exatidão requer a avaliação exata do produto escalar. Para este propósito, o novo tipo de dados *dotprecision* foi

introduzido representando um formato de ponto-fixa abrangendo todo o intervalo do produto de ponto-flutuante.

### 2.3.2 Disponibilidade

Já na metade dos anos sessenta, uma extensão em ALGOL foi implementada. Ela possuía, além dos tipos *integer* e *real*, um tipo para intervalos reais com seus correspondentes operadores aritméticos e relacionais. Esta linguagem serviu de suporte para o ALGOL-68. Exceto pela aritmética necessária, o ALGOL-68 tinha todos os conceitos de linguagens necessários para a implementação de algoritmos numéricos com verificação automática de resultados. Infelizmente, o ALGOL-68 não vingou, talvez por causa dos computadores ainda não serem suficientemente poderosos. Linguagens subseqüentes que proporcionavam capacidades de linguagem similares são Ada, C++ e FORTRAN-90.

Existiram algumas tentativas de linguagens, sistemas de software e pacotes nos quais a aritmética intervalar foi implementada. Em máquinas da IBM foram desenvolvidos o sistema TRIPLEX-ALGOL-60, o pacote ACRITH e as extensões PASCAL-SC e FORTRAN-SC. Existe ainda o pacote ARITHMOS que foi desenvolvido para computadores da SIEMENS.

Atualmente, existem pelo menos três desenvolvimentos de linguagens e de seus correspondentes compiladores que contêm todos os conceitos de programação e de aritmética para rotinas numéricas de programação com verificação automática de resultado. Tais linguagens são ACRITH-XSC, Pascal-XSC e C-XSC. O ACRITH-XSC é uma extensão da tão usada linguagem FORTRAN-77, junto com um grande número de rotinas de solução de problemas com verificação automática de resultados. O Pascal-XSC e o C-XSC são, respectivamente, extensões do Pascal e do C, que são linguagens de programação úteis e largamente distribuídas. Os compiladores para o Pascal-XSC e C-XSC podem ser usados em um grande número de máquinas de vários fabricantes.

#### 2.3.2.1 Pascal XSC

O Pascal-XSC<sup>15</sup> é uma extensão da linguagem Pascal para computação científica. Seu nome vem do inglês, **P**ASCAL **e**Xtension for **S**cientific **C**omputation. O Pascal-XSC contém as seguintes características baseadas no Pascal padrão, conceito de operador universal, funções e operadores com tipo de resultado arbitrário, sobrecarga de rotinas, funções e operadores, sobrecarga de atribuição de operadores, sobrecarga de rotinas de I/O - rotinas *read* e *write*, conceito de módulo, *arrays* dinâmicos, acesso a *subarrays*, conceito de *string*, arredondamento controlado, produto escalar ótimo (exato), tipo padrão *dotprecision* (um formato de ponto-fixa abrangendo todo o intervalo do produto de ponto-flutuante), tipos aritméticos adicionais, aritmética de alta exatidão para todos os tipos, funções de alta exatidão, avaliação exata de expressões (*# - expressions*).

<sup>15</sup>[GEO93, HAM93, KLA91, KLA92, DIV94d]



O Pascal-XSC possui aritmética intervalar, aritmética complexa, aritmética intervalar complexa e as correspondentes aritméticas de vetor e matriz.

Módulos de aplicação foram implementados em Pascal-XSC para a resolução de problemas numéricos, tais como: sistemas de equações lineares, inversão de matriz, sistemas de equações não lineares, autovalores e autovetores, avaliação de expressões aritméticas, avaliação de polinômios e de zeros polinomiais, equações integrais, problemas de otimização, entre outros.

O Pascal-XSC possui algumas vantagens: portabilidade do código, produz o mesmo resultado em qualquer plataforma, tem disponível a matemática intervalar e possibilita o desenvolvimento de algoritmos com verificação automática de resultados.

O Pascal-XSC é portátil em várias plataformas, estando disponível para computadores pessoais (PC's), estações de trabalho do tipo Sun e HP, mainframes e supercomputadores. Essa portabilidade é garantida pelo uso de um compilador que traduz para a linguagem ANSI-C. Como um dos principais objetivos do sistema é a portabilidade, o mesmo teve que providenciar uma fácil transferência do compilador e do sistema de execução; teve que permitir a necessária reconfiguração do compilador para cada novo computador; permitiu a portabilidade do código gerado através da compilação cruzada; e providenciou a consistência dos resultados para todas as plataformas.

A produção de resultados idênticos em todas as plataformas deve-se ao Pascal-XSC proporcionar uma completa simulação da aritmética de ponto-flutuante definida pelo padrão binário IEEE-754 ([DIV94]).

O Pascal-XSC é uma linguagem de programação de propósito geral que proporciona condições especiais à implementação de algoritmos numéricos sofisticados, que verificam matematicamente os resultados. Pelo uso dos módulos matemáticos, os algoritmos numéricos que providenciam alta exatidão e verificação automática de resultados podem ser facilmente programados, além disso, o projeto de programas para as Engenharias e para a Computação Científica é simplificado, graças à estrutura modular dos programas, à possibilidade de definição de operadores, à sobrecarga de funções, rotinas e operadores, às funções e operadores com tipos arbitrários de dados e aos *arrays* dinâmicos. Os módulos avançados são descritos no capítulo 6.

Os programas escritos nesta linguagem são de fácil leitura e existe ainda uma grande quantidade de problemas numéricos que podem ser resolvidos pelas bibliotecas de rotinas com verificação automática de resultados.

A descrição da linguagem para o Pascal-XSC tem sido publicada pela Springer Verlag em alemão, russo e inglês.

### 2.3.2.2 C XSC

A linguagem de programação C padrão tem muitos pontos falhos causando dificuldades para sua utilização na programação de algoritmos numéricos. Ela não providencia as estruturas de dados básicas como vetores e matrizes e, não realiza a verificação do intervalo dos índices de *arrays*. Isto pode resultar em erros, os quais são de difícil localização em algoritmos numéricos. Ainda mais, o tratamento de ponteiros e a perda da sobrecarga de operadores em C dificultam a capacidade de escrita e compreensão de programas, ocasionando que o desenvolvimento de programas seja uma tarefa mais dura do que deveria ser. Em C, também não existe controle da exatidão e dos arredondamentos direcionados em operações aritméticas.

A linguagem de programação C++, uma extensão do C orientada a objetos, tornou-se popular nos últimos anos. Ela não providencia melhores facilidades para os problemas, mas seu novo conceito de estrutura abstrata de dados (classes) e o conceito de sobrecarga de operadores e funções criaram a possibilidade de criar ferramentas computacionais que eliminam as desvantagens mencionadas na linguagem C. Isto possibilita a usuários de C e de C++ escrever algoritmos numéricos com resultados confiáveis em um ambiente de programação confortável.

Os aspectos da orientação a objetos da linguagem C++ providenciam ainda características poderosas da linguagem, reduzindo o esforço de programação e a herança da facilidade de escrita/leitura e da confiabilidade dos programas.

O C-XSC<sup>16</sup> é uma extensão do C. O compilador para o C-XSC, assim como o do Pascal-XSC, pode ser usado em um grande número de máquinas de vários fabricantes. Uma descrição da linguagem C-XSC, em inglês, foi publicada pela Springer Verlag.

Com esta estrutura abstrata de dados, com operadores e funções pré-definidas o ambiente de programação da linguagem C-XSC providencia uma interface entre a computação científica e as linguagens de programação C e C++. C-XSC suporta o desenvolvimento de algoritmos com inclusão automática da solução para dado problema matemático em limites verificados. Tais algoritmos resultam em aproximações matemáticas precisas da solução exata.

Entre as características mais importantes do C-XSC estão: aritmética real, complexa, intervalar e intervalar complexa com propriedades matemáticas definidas; vetores e matrizes dinâmicas; subarrays de vetores e matrizes; tipo de dado *dotprecision*; operadores aritméticos pré-definidos com alta exatidão; funções padrões com alta exatidão; aritmética dinâmica de multiprecisão e funções padrões; controle de arredondamento para rotinas de entrada e saída; controle do erro e uma biblioteca de rotinas para solução de problemas com verificação automática de resultados.

---

<sup>16</sup>[KLA93, ADA93]

Em contraste com C e C++, todos os operadores aritméticos pré-definidos, mesmo as operações com vetores e matrizes, proporcionam resultados corretos com até a última casa em C-XSC. Não há necessidade de se aprender as muitas das novas características do C++ para se ser capaz de se utilizar o ambiente de programação do C-XSC para aplicações numéricas. O conhecimento da linguagem C é suficiente para se trabalhar com C-XSC.

Programas C-XSC sempre produzem resultados numéricos compatíveis mesmo em diferentes computadores e com diferentes compiladores C++. Isto significa que C-XSC proporciona ferramentas para conseguir resultados numéricos totalmente compatíveis no sentido da matemática intervalar.

### 2.3.2.3 Linguagens do tipo Fortran

ACRITH-XSC<sup>17</sup> é uma linguagem de programação do tipo Fortran projetada para o desenvolvimento de algoritmos numéricos auto-validáveis. Tais algoritmos produzem resultados com alta exatidão, os quais são verificados se estão corretos pelo próprio computador. Então, não é necessária uma análise do erro manual para estes cálculos.

Com poucas exceções, ACRITH-XSC é uma extensão do Fortran 77, por isto também é chamado de Fortran-XSC. Vários conceitos de linguagem, os quais estão disponíveis no ACRITH-XSC podem ser, também, encontrados de forma similar em Fortran 90. Outras características do ACRITH-XSC têm sido projetadas para propósitos numéricos, como constantes numéricas, conversão de dados e operadores aritméticos com controle de arredondamento; aritmética intervalar e aritmética complexa intervalar; aritmética vetorial/matricial exata; amplo conjunto de funções matemáticas padrões com argumentos pontuais ou intervalares. Para a classe de expressões denominada "expressão produto escalar", o ACRITH-XSC proporciona uma notação especial que garante que estas expressões com estes tipos de dados sejam avaliadas com o último *bit* de exatidão, isto é, não existe nenhum número de máquina entre o resultado calculado e a solução exata.

**ACRITH** foi a primeira versão do ACRITH-XSC. O nome é uma abreviatura de "*High-Accuracy arithmetic subroutine library*". Inicialmente era uma biblioteca de um programa que tornava disponível rotinas VS FORTRAN para a solução de problemas padrões na análise numérica, rotinas interativas para treinamento e novas operações em ponto-flutuante. Estas rotinas usam a técnica de arredondamento direcionado e um produto escalar exato.

Para um número importante de áreas de problemas numéricos, a biblioteca da IBM de alta exatidão ACRITH proporciona vantagens. Estas áreas são: solução de equações lineares, inversão de matrizes, multiplicação de matrizes, solução de problemas algébricos de autovalores, solução de problemas de programação linear, avaliação de funções

---

<sup>17</sup>[ADA93, KUL83]

básicas, avaliação de expressões racionais e de polinômios, cálculo de zeros de polinômios e avaliação de produto escalar. A biblioteca produz um intervalo contendo os limites inferiores e superiores da solução exata do problema especificado, para cada componente da solução.

ACRITH-XSC contém algumas das características do Fortran 90, como expressões e funções de *arrays*, definição de operadores, sobrecarga de operadores, *arrays* dinâmicos e *subarrays*.

ACRITH-XSC por outro lado, possui certas características que o Fortran 90 não tem, estas são ferramentas numéricas especializadas como controle do arredondamento, aritmética intervalar, produto escalar exato e avaliação exata de expressões produto escalar. Estas características são úteis para o desenvolvimento de algoritmos numéricos que proporcionam alta exatidão e verificação automática do resultado.

ACRITH-XSC providencia controle automático do erro, controle de arredondamento e operadores com arredondamento direcionado. Todas as operações são acessíveis pelo seu símbolo usual de operação da álgebra linear. Isto permite que a mesma notação de expressões de vetores e matrizes da matemática seja utilizada na codificação de programas. Esta característica é conhecida como sobrecarga de operadores e funções, ficando a tarefa de identificar a rotina correta para cada tipo de dado ao compilador.

Em ACRITH-XSC, aritméticas de vetores e matrizes são pré-definidas de acordo com as regras da álgebra linear. Todas as operações são acessíveis através de seu símbolo de operador usual. Isto permite a mesma expressão de notação de vetores e matrizes da matemática. Operadores aritméticos e relacionais para vetores e matrizes de componentes reais, complexos, intervalares e intervalares complexos são pré-definidos.

O ACRITH-XSC foi desenvolvido e implementado no Instituto de Matemática Aplicada da Universidade de Karlsruhe com o patrocínio e a colaboração da IBM. Este agora é distribuído como um IBM Program Product. Infelizmente, o uso do ACRITH-XSC é limitado para máquinas com a arquitetura IBM /370 e com o sistema operacional VM CMS.

## 2.4 Considerações finais

As linguagens de programação Pascal-XSC, ACRITH-XSC e C-XSC citados aqui, trazem um avanço essencial. Elas proporcionam uma aritmética computacional universal e permitem o manuseio mais simples das operações semimórficas por meio de símbolos de operadores matemáticos ordinários e, também, em espaços de produtos, tais como os números complexos, assim como para vetores e matrizes de tipos *real*, *complex*, *interval* e *complex interval*.



Infelizmente, as últimas operações não são suportadas pelos padrão de aritmética IEEE 754 mencionadas acima, portanto, têm que ser simuladas em software. Assim, no lugar onde um aumento de velocidade é realmente esperado, na verdade, ocorre uma perda de velocidade. Uma dificuldade adicional vem em consequência dos processadores que oferecem o padrão 754 do IEEE e que não fornecem o produto em dupla precisão, então isto também tem que ser simulado, o que implica uma considerável perda de velocidade. Produtos em dupla precisão são indispensáveis e essenciais para o cálculo da multiplicação vetorial e matricial semimórficas.

Para uma implementação por software da aritmética intervalar avançada é necessário que os tipos sejam encapsulados, ou seja, a existência na linguagem de implementação de mecanismos que garantam a manipulação do tipo exclusivamente pelas operações definidas sobre este.

Por outro lado, a linguagem de implementação deve proporcionar uma transparência ao usuário, de modo que este não necessite saber e nem se preocupar em como a implementação foi realizada, mas somente que ela proporciona cálculos com máxima exatidão, de forma a evitar conflitos de acesso aos dados, em caso de processamento vetorial e paralelo.

Dentre as diversas linguagens de programação, existem algumas dedicadas à implementação de tipos abstratos de dados, mas que não são apropriadas ao processamento numérico. Deve-se escolher uma linguagem que combine estas duas características, ou desenvolver uma extensão de linguagens de programação para a manipulação de intervalos.

A implementação por software da aritmética avançada é bastante simples e flexível, entretanto pode apresentar problemas com relação à eficiência, especialmente quando são implementados os tipos de dados presentes na aritmética intervalar, onde se tem que as operações sobre intervalos consomem, no mínimo, o dobro do tempo de uma computação em uma máquina convencional com números em ponto-flutuante devido ao fato de termos o processamento com os dois limites do intervalo, inferior e superior.

Por outro lado, pelo controle do erro de arredondamento a cada passo de um cálculo, é possível calcular limites garantidos da solução e, então, verificar os resultados numéricos no computador. Inclusões de todo o conjunto de soluções podem ser calculadas pelo uso da aritmética intervalar, como por exemplo no tratamento de problemas que envolvem dados imprecisos ou outro tipo de dados com tolerância, ou ainda, para estudar a influência de certo parâmetro sobre a solução, muito utilizado na análise sensitiva. Logo, a análise intervalar é particularmente útil para o estudo da estabilidade e análise sensitiva. Isto proporciona uma das bases fundamentais para computações numéricas confiáveis.

A resolução IMACS-GAMM<sup>18</sup> sobre aritmética computacional requer que todas operações aritméticas, em particular as operações compostas de computadores vetoriais, tais como: *multiplica-e-soma*, *acumula*, e *multiplica-e-acumula* sejam implementadas de forma que limites garantidos sejam produzidos para os resultados calculados em ponto-flutuante em relação ao resultado exato. Um resultado que difere do resultado matematicamente exato por até um *ulp* (no máximo um arredondamento) é altamente desejável e sempre obtido nas extensões XSC das linguagens.

Uma proposta de aritmética vetorial de ponto-flutuante exata, essencialmente requer que as operações vetoriais tenham as mesmas propriedades matemáticas que são requeridas para operações aritméticas elementares pelo padrão IEEE.

---

<sup>18</sup> No anexo I está a tradução integral e o original está em [ADA93, IMACS91]

### 3 AMBIENTES COMPUTACIONAIS

Neste capítulo são caracterizados os ambientes computacionais onde esta pesquisa se desenvolveu. Na descrição dos ambientes computacionais é especificada a máquina e a linguagem de programação.

#### 3.1 O Cray Y MP e a linguagem Fortran 90

O Supercomputador Cray, utilizado neste trabalho, encontra-se no Centro Nacional de Supercomputação (CESUP/RS). O CESUP instalou e opera o primeiro supercomputador de uso geral da América do Sul. Ele é da família Y-MP com dois processadores vetoriais com as seguintes características:

- velocidade total máxima de 660 *MFlops*;
- palavra de 64 *bits*;
- memória RAM de 256 *Mbytes* e 16 *Gbytes* de disco;
- sistema operacional UNICOS, compatível com *UNIX System V*.

Atualmente as aplicações<sup>1</sup> no CRAY Y-MP estão concentradas nas áreas de: Geofísica; Química computacional; Dinâmica de Fluidos; Análise estrutural; Matemática/ algoritmos; Física do estado sólido; Física (campos/ partículas/ ótica/ astrofísica); Física de Plasma; Computação de Alto Desempenho; Redes Neurais; Engenharia Química e Ciências do Ambiente.

No Cray do CESUP/RS, estão disponíveis os compiladores FORTAN (77 e 90), C e Pascal com subrotinas científicas e matemáticas altamente otimizadas, incluindo Boeing CS, BLAS3, LINPACK, EISPACK, FFT e operações matriciais. Dispõe, também, dos seguintes pacotes de aplicação: MPGS: sistema de visualização de 2 e 3 dimensões, UNICHEM: sistema interativo de química quântica, MOPAC II: analisador de química quântica, GAMESS: analisador atômico e molecular, SPEEDUP: pacote de simulação de processos e plantas químicas, ANSYS: pacote de análise de estruturas, calor e campos, MCS/NASTRAN: análise estrutural e de transferência de calor, MCS/EMAS: análise de campos eletromagnéticos, MCS/DYTRAN: análise dinâmica de sólidos-fluidos, MCS/XL: pacote de visualização de dados, ADAMS: simulação de estruturas mecânicas articuladas, SPICE2 e 3: simulador elétrico, PISCES-2B: simulação de dispositivos semicondutores, PVM/HeNCE: sistema de programação paralela por troca de mensagens, REDUCE: sistema de computação algébrica e FLOTRAN: pacote de análise de fluxo de fluidos por elementos finitos.

---

<sup>1</sup>Segundo fontes do próprio CESUP, nos anais do Seminário de Supercomputação Aplicada, Supercomp 94,

### 3.1.1 Arquitetura do Cray

O *mainframe* contém uma seção de *I/O*, uma seção de comunicação entre processadores, a memória central e um número variável de unidades processadoras. Todas as CPUs no sistema de multiprocessamento compartilham a memória central, a seção de *I/O* e a seção de intercomunicação, como ilustra a figura 3.1.

A **memória central** é compartilhada pelas CPUs e pela seção de *I/O*. Ela é dividida em seções e bancos. Este arranjo admite simultaneidade e gerenciamento de referências da memória. Simultaneidade refere-se a duas ou mais referências sendo feitas ao mesmo tempo e, gerenciamento de referências são uma ou mais referências que iniciam enquanto outra referência está em andamento.

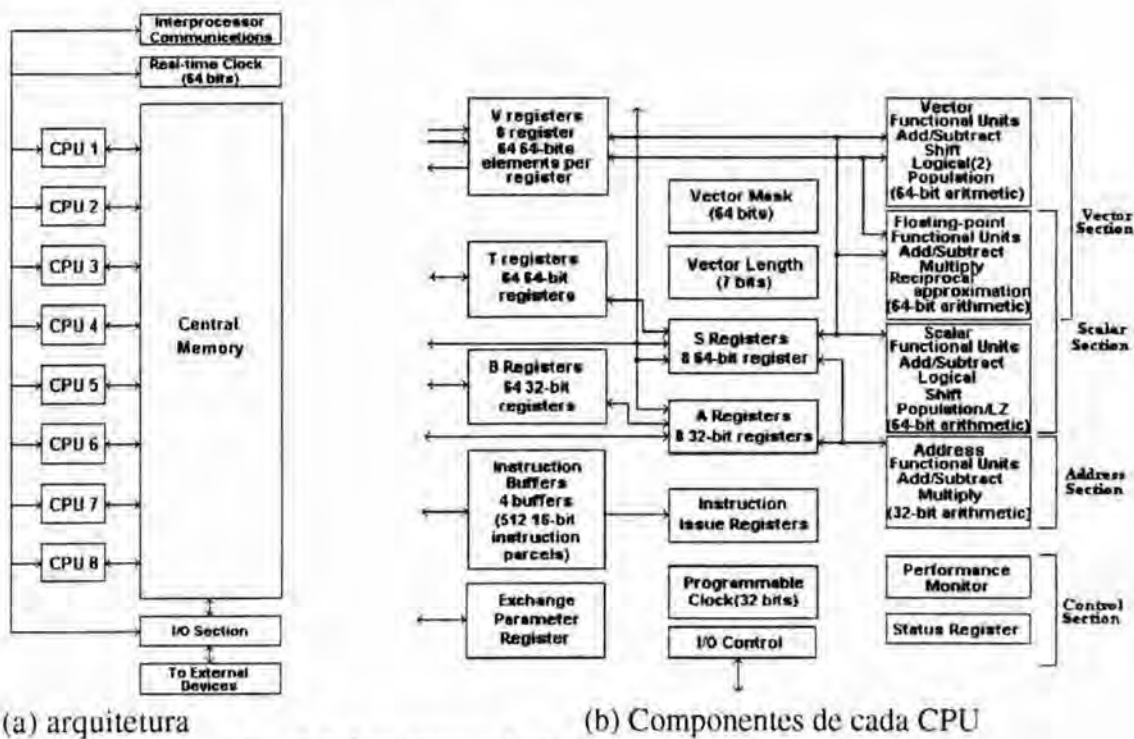


Fig.3.1 Arquitetura do Supercomputador Cray Y-MP

Proteção de dados, correção de erros simples e detecção de erros duplos são realizados na memória central e nos canais de comunicação de dados de ou para a memória. Quando um dado é escrito na memória central, uma verificação dos *bytes* é feita, gerada pela palavra e armazenada. Quando ele for lido da memória central, a checagem e o dado são processados para verificação se algum *bit* foi alterado. Se não ocorreram erros a palavra é transferida sem modificações. Se são detectados erros, os *bits* são analisados pela lógica para ver o número de *bits* alterados. Se somente um *bit* foi modificado, ele é corrigido, voltando ao estado correto e é transferido. Se mais de um *bit* foi alterado, não é possível corrigir a palavra pois o resultado é imprevisível, então é setado o indicador de erro (*flag*) gerando uma interrupção.



A organização da memória central pode ser feita em duas ou quatro seções. Cada uma é constituída de oito subseções. Subseções são ainda subdivididas em bancos separados. A memória é intercalada através de todos os bancos, subseções e seções, resultando que dados consecutivos sejam armazenados em diferentes bancos. Nenhum banco é reusado enquanto o último banco não foi utilizado.

Cada CPU tem seu próprio caminho para cada seção da memória. Cada CPU tem quatro portas de memória (A, B, C e D). Elas são geralmente usadas para funções específicas, duas para leitura (A e B), uma para escrever (C) e uma para instruções dos *buffers* ou para requisições de *I/O* (D). Para que uma CPU acesse a memória tem de ter tanto uma das portas disponível quanto um caminho disponível para a seção de memória.

O ciclo de tempo de acesso ao banco é de cinco períodos (5 CPs). Isto significa que cada referência da memória feita pela CPU torna o banco de memória referenciado indisponível para todas as portas em todas as CPUs por 5 CPs. Ainda mais, cada referência de memória torna uma subseção inteira de memória indisponível a todas as portas da mesma CPU por 5 CPs.

O *mainframe* foi construído sobre uma resolução de *hardware* que minimiza atrasos causados por conflitos de memória e mantém a integridade de toda referência de memória quando conflitos ocorrem. Conflitos de memória ocorrem quando mais de uma referência é feita na mesma área da memória central.

O *subsistema de I/O* providencia a alta velocidade de transferência entre o subsistema de *I/O* (*IOS*) e o *SSD* (*solid state storage device*). Existem três tipos de canais:

- de 1000 *Mbytes/s* - usado para transferir dados entre o *mainframe* e o *SSD*;
- de 200 *Mbytes/s* - usado para transferir dados entre o *mainframe* e o *IOS*;
- de 6 *Mbytes/s* - usado para controlar informações entre o *mainframe* e o *IOS*.

A quantidade de cada tipo de canal varia de acordo com a configuração do sistema e depende da quantidade de CPUs e *I/O clusters*.

O subsistema de *I/O* realiza as seguintes funções: controla todas as transferências de dados entre o *mainframe* ou *SSD* e os dispositivos periféricos, tais como *drives* de disco e computadores *front-end*; converte os dados para o formato dos dispositivos periféricos; detecta e corrige certos tipos de erros nos dados que podem ocorrer durante a transferência.

O *SSD - Solid-State Storage Device* - é um dispositivo de alto desempenho usado para armazenar dados temporários. O *SSD* transfere dados da memória central por canais de 1000 *Mbytes/s*. Estes canais operam abaixo do controlador. O *SSD* pode também ser conectado ao subsistema de *I/O*. O Cray do Centro Nacional de Supercomputação não tem este periférico no momento.

Cada CPU tem uma seção de controle e uma de cálculo. A seção de controle determina a instrução desejada e usa a seção de cálculo, memória e *I/O*. A seção de cálculo consiste de registradores operacionais e unidades funcionais.

A seção de comunicação entre processadores permite que cada CPU sincronize operações com outras CPUs e transmita dados.

O *mainframe* contém um relógio de tempo real, o qual é compartilhado por todas as CPUs. Este relógio consiste de um contador de 64 *bits* que avança uma unidade a cada período do relógio (*clock period* - CP).

Todas as CPUs são idênticas, cada uma tem sua seção de cálculo independente constituída de registradores operacionais, unidades funcionais e instruções de controle da rede, como mostra a figura 3.1.

Cada CPU tem três conjuntos, de oito registradores, denominados de primários porque podem ser acessados diretamente pela memória central e unidades funcionais. Estes registradores são de endereçamento (tipo A), escalares (tipo S) e vetoriais (tipo V). O tamanho dos registradores de endereçamento é de 32 *bits*, os escalares de 64 *bits* e os vetoriais são compostos por 64 elementos, sendo cada elemento de 64 *bits*.

Existem ainda dois conjuntos de 64 registradores intermediários, denominados B e T, os quais são usados para armazenar os valores dos registradores A e S respectivamente. Observa-se que os registradores intermediários B tem o tamanho de 32 *bits*, enquanto que os registradores intermediários T são de 64 *bits*.

Os registradores V servem como fonte e destino para instruções aritméticas e lógicas vetoriais. Elementos sucessivos do registrador V entram em uma unidade funcional em sucessivos CPs como instruções simples. O tamanho efetivo do registrador V para cada operação é controlado por um programa selecionador do registrador VL (*vector length*). O registrador VL tem 7 *bits* que especificam o número de elementos do vetor processados pela instrução. A instrução vetorial só termina quando VL vale zero.

O registrador vetorial *Mask* (VM) permite a seleção lógica de elementos particulares de um vetor. O VM tem 64 *bits*, cada um correspondendo a um elemento de V. Ele é usado em instruções de teste para realização de operações em elementos individuais. Um exemplo de operação vetorial que se utiliza do registrador VM é a instrução lógica de Inserção (*merge*), onde supondo  $V1 = \langle 10101010 \rangle$ ,  $V2 = \langle 11001100 \rangle$  e se quer fazer uma operação de *merge* de V1 e V2 segundo o registrador  $VM = \langle 11110000 \rangle$ , onde o *bit* 1 indica elemento do registrador V1 e o *bit* 0 indica elemento do registrador V2, teremos como resultado  $V3 = \langle 10101100 \rangle$ .

As instruções são realizadas por *hardware* especializado conhecidos por unidades funcionais, as quais implementam um algoritmo ou uma porção do conjunto de instruções. Elas são independentes logicamente e podem operar simultaneamente.

As unidades funcionais são descritas em quatro grupos: endereço, escalar, vetor e ponto-flutuante. Cada um dos três primeiros grupos funciona com um dos três tipos de registradores primários (A, S e V) para dar suporte aos modos de processamento endereçado, escalar e vetorial. O quarto grupo, o de ponto-flutuante aceita tanto operações escalares quanto vetoriais, permitindo que operandos de resultados dos registradores S e V também sejam acessados. A memória central também pode agir como uma unidade funcional para operações vetoriais.

Os registradores e as unidades funcionais, portanto, estão associados a três tipos de processamento: endereçamento, escalar e vetorial.

Processamento de endereçamento opera com informações de controle interiores, como contadores de loops, endereços e índices. Este processamento é feito por registradores de endereçamento (tipo A) e usa unidades funcionais dedicadas de aritmética inteira. Informações de endereçamento vêm da memória central para instruções de valores ou de registradores de controle para registradores de endereçamento, dos quais são distribuídos para várias partes de rede de controle para serem usadas no controle das operações escalares, vetoriais e de *I/O*.

Processamento escalar e vetorial são efetuados sobre dados. Processamento escalar ocorre seqüencialmente e usa um operando ou um par de operandos para produzir um único resultado. Processamento escalar é efetuado usando registradores escalares.

A unidade funcional de endereçamento realiza operações inteiras sobre operandos obtidos de registradores A e envia os resultados para registradores A. Existem duas unidades, uma para realização das operações de adição e subtração e outra para multiplicação inteira. Observa-se que a condição de *overflow* não é detectada nestas operações.

A unidade funcional escalar realiza operações sobre operandos obtidos em registradores S e geralmente envia resultados para registradores S. Podendo em exceção enviar o resultado a um registrador A, no caso de paridade. Há quatro unidades funcionais escalares: a lógica que realiza a manipulação *bit-a-bit* de quantidades obtidas de registradores S; a soma que realiza a soma e subtração inteira usando aritmética de complemento de dois, onde a condição de *overflow* não é detectada; a *shift* que desloca o conteúdo do registrador S ou desloca o conteúdo de dois registradores S concatenados em um único registrador S resultante (observa-se que o primeiro tipo de deslocamento não é circular, ou seja perde-se o *bit* deslocado e coloca-se zero na outra extremidade; o segundo deslocamento é circular) e a paridade que conta o número de *bits* em um registrador S que possui o valor de um operando e então, dependendo da operação desejada, retorna o valor tanto como quantidade quanto como paridade a um registrador A.

A unidade funcional vetorial realiza operações sobre operandos obtidos de um registrador ou dois registradores V, ou ainda, de um registrador V e um registrador S. As



unidades lógicas e de soma requerem dois operandos, enquanto que a de *shift* e a de paridade requerem somente um operando. O resultado é enviado a um registrador V. Sucessivos pares de operandos são transmitidos de cada CP para unidade funcional. O resultado correspondente surge da unidade funcional após  $n$  CPs, onde  $n$  é a unidade de tempo funcional, sendo constante para cada unidade. O registrador VL determina o número de operandos ou par de operandos a ser processado pela unidade funcional.

Há oito unidades funcionais vetoriais, sendo que três operam com escalares também e serão descritas como unidades funcionais de ponto-flutuante. As unidades funcionais vetoriais são: adição e subtração, que realiza as operações de adição e subtração inteira; *shift*, que realiza o deslocamento do conteúdo de um registrador V ou do valor formado de dois elementos consecutivos do registrador V; lógica, que realiza a manipulação *bit-a-bit* de quantidades especificadas para instruções específicas, realizando inserções, compressão indexada e operações lógicas envolvendo o vetor *Mask* (VM), o qual seleciona elementos particulares a serem operados; unicamente lógico, que não efetua as operações complementares da unidade anterior e, por fim, a unidade funcional de paridade.

A unidade funcional de ponto-flutuante é composta por três unidades que realizam a aritmética de ponto-flutuante para operações escalares e vetoriais. Quando uma instrução escalar é desejada, os operandos são obtidos de registradores S e os resultados enviados a registradores S. Para a maioria das operações vetoriais, os operandos são obtidos de um par de registradores vetoriais, ou de um registrador S e um V. Os resultados são enviados para um registrador V. A unidade de adição realiza a adição e subtração de operandos em ponto-flutuante. O resultado final é normalizado mesmo quando os operandos não o eram. São detectadas as condições de *underflow* e *overflow*, mas marcado somente o segundo. A segunda unidade é a de multiplicação, que realiza a multiplicação em plena e em meia precisão de operandos em ponto-flutuante. O produto em meia precisão é arredondado, enquanto que o em plena precisão pode ou não ser arredondado. A terceira unidade é da aproximação recíproca que é utilizada na operação de divisão, através de um cálculo aproximado do inverso do divisor o qual será multiplicado pelo dividendo. Ou seja, a divisão  $A/B$  é feita pelo produto  $A*(1/B)$ , onde  $(1/B)$  é aproximado pela unidade funcional de aproximação recíproca.

A seção de cálculo realiza operações inteiras e em ponto-flutuante. As operações inteiras são efetuadas no modo complemento de dois, enquanto que as quantidades em ponto-flutuante têm a representação sinal-magnitude.

A principal vantagem do processamento vetorial sobre o escalar é a eliminação do tempo de carga (*startup*) de todas as operações. O tempo de carga para operações vetoriais é suficientemente pequeno para que o processamento vetorial seja mais eficiente do que o processamento escalar para vetores contendo poucos elementos.



### 3.1.2 Principais Características

A série do sistema de computadores Cray Y-MP é poderosa e de propósito geral. Consiste em quatro linhas de produtos, cada linha de produto está disponível em vários modelos. Estes modelos permitem que os usuários escolham o número de CPUs, tamanho de memória central, número de clusters de I/O entre outros.

As quatro linhas de produtos são os sistemas de computadores CRAY Y-MP8E, Y-MP8I, Y-MP4E e Y-MP2E. Todos os sistemas são constituídos dos seguintes componentes: um *mainframe*, um sub-sistema de entrada e saída modelo E, denominado IOS, um dispositivo de armazenagem de estado rígido ótimo (SSD), dispositivo de armazenagem com discos e *drives* de fitas, estações de trabalho para operação e manutenção, redes de interface e um poderoso equipamento de suporte.

A rede do supercomputador instalada no CESUP da UFRGS pode ser acessada por todas as redes locais da UFRGS, situadas nos diferentes campus e, também, está ligada a Rede Nacional de Pesquisa (RNP), possibilitando que usuários de outros estados venham a utilizar o supercomputador "remotamente" (*login* remoto).

Processamento paralelo significa, para a Cray, diferentes coisas sendo feitas em diferentes ambientes. Características de processamento paralelo com uma única CPU incluem *pipelineização*, segmentação, independência das unidades funcionais e processamento vetorial. As três primeiras características são inerentes ao *hardware* da máquina, enquanto que o processamento vetorial pode ser manipulado pelo programador para providenciar um melhor desempenho da máquina.

A *pipelineização* é definida como uma operação ou instrução iniciada antes que uma outra operação ou instrução prévia seja completada. A *pipelineização* é conseguida através do pleno uso do *hardware* segmentado. Segmentação refere-se ao processo onde uma operação é dividida em um número finito de passos seqüenciais ou segmentos. A segmentação completa de *hardware* é projetada para implementar esta segmentação efetuando-se um segmento da operação durante um simples CP. Até o início do próximo CP, os resultados parciais obtidos são enviados para o próximo segmento de *hardware* para ser processado o próximo passo da operação. Durante este CP, o segmento anterior de *hardware* pode processar a próxima operação. Se a segmentação não é usada, toda a operação ou instrução tem que terminar antes de iniciar a outra.

A figura 3.2 mostra a segmentação de uma operação vetorial, a qual foi dividida em cinco passos. Têm-se dois registradores vetoriais V1 e V2 como operandos e o registrador V3 como registrador resultante. Observam-se os elementos em cada segmento durante sucessivos CPs. Os resultados passam a surgir após cinco CPs e a partir daí surge um resultado a cada CP.

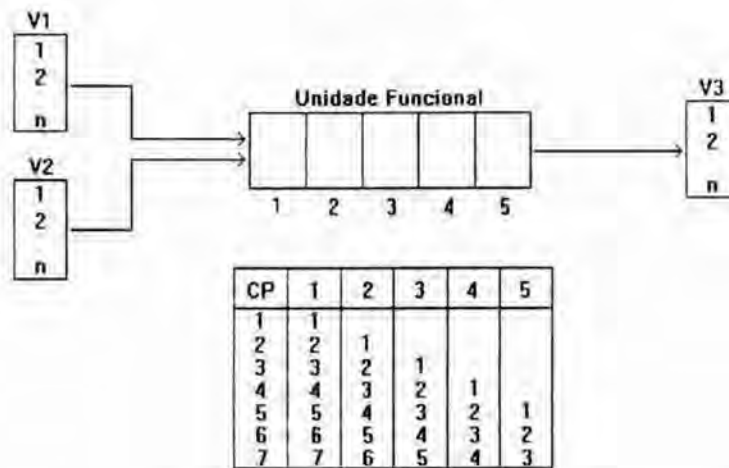


Fig. 3.2 Exemplo de Segmentação e *pipelineização*

Vê-se como um conjunto de elementos é *pipelineizado* através da unidade funcional vetorial segmentada, neste caso ela foi dividida em cinco segmentos. No primeiro CP, o elemento 1 do registrador V1 e o primeiro elemento do registrador V2 entram no primeiro segmento da unidade funcional. Durante o próximo CP, os resultados parciais são movidos para o segundo segmento da unidade funcional e os segundos elementos dos dois registradores entram no primeiro segmento. Este processo continua em cada CP, até que todos os elementos sejam completamente processados.

As unidades funcionais especializadas do *mainframe* garantem as operações aritméticas, lógicas e deslocamentos. A maioria das unidades são totalmente independentes de outras e qualquer número de unidades funcionais pode processar instruções concorrentemente. Esta independência das unidades funcionais admite diferentes operações como multiplicação e adição sendo processadas em paralelo. Um exemplo é a avaliação da expressão  $A=(B+C) \times D \times E$ . Ela pode ser avaliada em três instruções, se os valores de B, C, D e E estiverem carregados nos registradores escalares S. As operações B+C e DxE podem ser feitas ao mesmo tempo, mas como a multiplicação demora mais tempo, ela iniciará primeiro, de forma que as duas terminem ao mesmo tempo, para então realizar o produto dos dois resultados parciais, obtendo o resultado final.

O *mainframe* permite que um registrador reservado a resultados se torne o operando de outra instrução sucessiva. Este processo é chamado de encadeamento e admite um fluxo contínuo de operandos, fluindo através de registradores vetoriais e unidades funcionais. Mesmo quando uma operação vetorial de carga pára, aguardando a resolução de um conflito de acesso de memória, o encadeamento de operações pode ser processado tão logo os dados se tornem disponíveis. Eles ocorrem na forma de instruções compostas, instruções multiplicação-adição e multiplicação-acúmulo. Além de capacitar o pleno uso de *pipelines*, instruções vetoriais também reduzem substancialmente a necessidade de executar ramos de instruções.

Este mecanismo de encadeamento admite encadeamento iniciado em qualquer ponto do fluxo vetorial dos dados resultantes. A quantidade de concorrência em um encadeamento depende do relacionamento entre o tempo de trabalho da instrução encadeada e do fluxo de dados resultantes. Para que ocorra um pleno encadeamento, a instrução de encadeamento tem que ter o tempo de trabalho e estar pronta para usar o elemento zero do resultado ao mesmo tempo que ele chega no registrador V. O encadeamento parcial ocorre se a instrução encadeada usa o elemento após sua chegada.

As instruções que operam com vetores podem ser divididas em quatro grupos:

- a) Vetor-vetor: Os operandos das instruções são obtidos de um ou dois registradores vetoriais V e o resultado é colocado em outro registrador vetorial V;
- b) Vetor-escalar: Os operandos das instruções são obtidos de um operando escalar (registrador S) e outro vetorial (registrador V). O resultado é colocado em outro registrador vetorial V;
- c) Vetor-Memória: Instruções que carregam (lêem) e armazenam (escrevem) elementos para a memória;
- d) Vetores auxiliares: Instruções que se utilizam de registradores auxiliares, os registradores VM e VL.

### 3.1.3 O Compilador Fortran 90

O Fortran ainda é a principal linguagem de programação utilizada na área científica. O Fortran vem evoluindo desde o compilador Fortran IV, para Fortran 77, que incluiu processamento vetorial e paralelo e, enfim, para o Fortran 8x, finalmente conhecido como Fortran 90. Portanto, o Fortran 90 é uma extensão do Fortran padrão. O Fortran 90 contém todo o Fortran 77, ou seja, todos os programas escritos em Fortran 77 ainda são compilados, podendo haver algumas diferenças na forma como serão executados, mas em geral serão compatíveis. O Fortran 90 foi escrito pensando no usuário, do ponto de vista do programador.

Entre as principais características existentes no Fortran 77, podem-se destacar a sintaxe de vetores, *arrays* automáticos, subprogramas recursivos, linha de inclusão (*include*) e os comandos estruturados *do-end* e *do-while*. O Fortran 90, além destas características introduz: nova forma para programas fonte, números complexos com dupla precisão, atribuição de ponteiros, *arrays* alocáveis, estruturas, módulos, rotinas internas, especificações de interface, várias novas funções intrínsecas e novas rotinas de entrada e saída.

O compilador diferencia os programas pela extensão dos nomes dos arquivos. Os programas em Fortran 77 têm a extensão *.f*, enquanto que os programas em Fortran 90 possuem a extensão *.f90*. O Fortran 90 aceita a velha forma de identificar a linha de comando e introduz uma nova forma. Os programas Fortran podem ser escritos de uma forma livre. O padrão determina que não se misture a forma tradicional com a forma livre, já o padrão Cray possibilita a combinação das formas, mas com a diretiva *cdir*.

O uso do caractere **&** no final da linha indica esta continuação. Ele também pode aparecer no início da linha de continuação, apesar disto não ser muito recomendável no caso do uso de sequências (*strings*). Na forma livre uma linha de comando pode ter até 132 caracteres e um comando pode aparecer em qualquer lugar da linha. Pode haver até 39 linhas de continuação. As linhas comentário contém o símbolo de exclamação **!**. Portanto qualquer caractere na linha de comando que seguir o ponto de exclamação será considerado como comentário.

Os identificadores podem ter até 31 caracteres, sendo que devem iniciar por uma letra. Os demais caracteres podem ser letras, dígitos ou o sinal de sublinhado. Todos os caracteres são significativos para diferenciar identificadores. As letras maiúscula e minúscula são equivalentes. Numa linha com mais de um comando, estes devem ser separados por ponto e vírgula **;**.

Os símbolos e os mnemônicos que podem ser utilizados para os operadores relacionais são dados pela figura 3.3, os demais operadores só são caracterizados por mnemônicos: *.EQV.*, *.NEQV.*, *.AND.*, *.OR.*, *.NOT.*, *.TRUE.*, *.FALSE.*. Um exemplo do uso do comando **if** é dado a seguir: **If ( I ==1) then; I=2; else if (i>=2) then; i=1; end if.**

igual	.EQ.	==
menor	.LT.	<
maior	.GT.	>
não igual	.NE.	/=
menor igual	.LE.	<=
maior igual	.GE.	>=

Fig 3.3 Operadores Relacionais

### 3.1.3.1 Tipos-padrões de dados

Os tipos de dados existentes no Fortran 90 são mostrados pela figura 3.4.

<i>integer</i>	inteiro
<i>real</i>	real
<i>double</i>	real em dupla precisão
<i>complex</i>	complexo
<i>logical</i>	lógico (booleano)
<i>character</i>	caracter (ASCII)

Fig. 3.4 Tipos de dados do Fortran 90

Os comandos de especificação dos tipos de dados já existentes são: *real*, *integer*, *character*, *dimension*. Os novos comandos que especificam as atribuições dos dados são: *pointer*, *target*, *allocatable*, *public*, *private*, *intent* e *optional*. Eles podem ser para uma mesma variável, separados por vírgula. Ao final são atribuídos à variável que segue ao separador **::**. Por exemplo, **real, dimension(10), target :: var1, var2**. Isto significa que as



variáveis *var1* e *var2* são reais, de dimensão 10 e *target*. As três declarações são simplificadas em uma.

Novos tipos de dados complexos em dupla precisão e processadores específicos para a mantissa e expoente de tamanho real estão na especificação do Fortran 90. Outra inovação importante é a precisão de variáveis e constantes do tipo real. A forma de especificar a precisão é através das funções: *selected\_real\_kind(n)* e *selected\_int\_kind(n)*. É interessante saber o tipo de número que se está usando, mas é mais útil determinar a precisão necessária para realizar os cálculos. Isto é mais evidente no momento de definição dos parâmetros.

Cada tipo de representação suportada pelo processador é designada por um número inteiro chamado *kind*, que é na verdade o número que determina o tipo de parâmetro. Declarações de tipos de dados são expressas em termos destes parâmetros, que podem ser controlados pelo programa através de constantes cujos valores são fixados no tempo de compilação.

O processador precisa ao menos suportar dois *kinds* com diferentes precisões, que correspondem ao *default* (por definição) tipo real simples e em dupla precisão do Fortran 77. As declarações antigas ainda são aceitas, mesmo sem o tipo de parâmetro *kind*.

Uma nova função intrínseca *selected\_real\_kind(n)* retorna um valor inteiro que corresponde ao tipo de parâmetro *kind*, que providenciará os primeiros *n* dígitos de precisão. A função intrínseca *kind(x)*, por sua vez, retorna o valor correspondente ao tipo de parâmetro *kind* de *x*.

Uma expressão real terá como valor do tipo de parâmetro *kind* o valor do *kind* de seus componentes. Caso eles tenham valores diferentes, o resultado da expressão será o *kind* do operando de maior precisão. O *kind default* de inteiros é 6, reais é 8, *double* 16, *complex* 8, *logical* 8 e *character* um.

Da mesma forma que reais, cada tipo de representação de inteiros suportado pelo processador possui um valor inteiro com o tipo de parâmetro *kind*. Este valor dependerá do processador. Já para variáveis lógicas, só há o *kind default* e este é igual ao *default* real e inteiro, para ser compatível com o Fortran 77. Para o tipo *character*, o *kind* disponível no Fortran 90 do Cray é ASCII.

Objetos e expressões complexas têm um tipo de parâmetro *kind*, o qual é o valor dos seus componentes. Isto implica que tipos de complexos podem ser compostos de dois valores em dupla precisão.

No Fortran 90, pode-se definir tipos derivados. Os tipos derivados lembram os registros em Pascal. Eles são definidos por estruturas que iniciam por *type* e terminam por *end type*, como é mostrado na figura 3.5. O símbolo de porcentagem % é usado para indicar componentes do referido tipo derivado, como por exemplo intervalos de reais, vetores e matrizes de intervalos, que são definidos na figura 3.6 e referenciados na figura 3.7.

```

type nome_do_novo_tipo
    type_spec :: field1, field2, ...
    .
    .
end type nome_do_novo_tipo

```

Fig 3.5 Sintaxe da definição de tipos de dados

Essa é uma forma simples do comando `type`. Na verdade, existem outros parâmetros que podem ser acrescentados para diversos fins, tais como: especificar que os dados serão armazenados em ordem na memória da máquina e determinar os acessos a cada campo da estrutura.

```

type interval
    real :: inf, sup
end type interval
type (interval), dimension(n) :: vector_interval
type (interval), dimension (n,m) :: matrix_interval

```

Fig 3.6 Definição de intervalos reais, vetores e matrizes de intervalos

```

vector_interval(5) % inf= 3.5
vector_interval(5) % sup = 4.6
vector_interval(6)= interval(3.7 , 4.2)

```

Fig 3.7 Referência aos novos tipos em Fortran 90

### 3.1.3.2 Interfaces, Subrotinas e Funções

Além da facilidade de poder criar suas próprias funções (*function*) e subrotinas (*subroutine*), o programador pode ainda definir uma interface para elas, de forma que a notação dos operadores existentes no Fortran 90 seja estendida aos tipos de dados derivados. Por exemplo, pode-se definir a função de soma para o tipo intervalo e, ainda, permitir que se utilize o operador `+` para indicar a soma de intervalos, obtendo desta forma uma sobrecarga de identificadores de subrotinas e funções. Para tanto, define-se a função de soma de intervalos, por exemplo denominada *sadd* como mostra a figura 3.8.

```

function sadd(A,B) result (C)
    type (interval), intent (in) :: A,B
    type (interval) :: C
    C%inf=A%inf+B%inf
    C%sup=A%sup+B%sup
end function sadd

```

Fig 3.8 Função soma de intervalos

Uma vez definida a função, define-se a interface. Esta definição tem o formato parecido com o da definição da função, diferindo apenas por não ter o corpo da função, como mostra a Figura 3.9.

```
interface operator (+)
function sadd(A,B) result (C)
  type (interval), intent (in) :: A,B
  type (interval) :: C
end function sadd
end interface
```

Fig 3.9 Definição da interface para soma de intervalos

Desse modo, quando o compilador encontrar no programa uma expressão  $X+Y$  e, ao verificar que os tipos de  $X$  e  $Y$  são intervalos, utilizará automaticamente a função de soma de intervalos *sadd*. Da mesma forma, pode-se ainda definir a soma de vetores intervalares, soma de matrizes intervalares e soma de intervalos complexos. A todas estas funções se pode atribuir o operador de adição  $+$ . Quando o compilador encontrar o operador  $+$ , escolherá convenientemente a função de soma de acordo com os tipos de dados dos argumentos, obtendo-se assim a sobrecarga de identificadores de funções e subrotinas.

### 3.1.3.3 Módulos

No Fortran 90, pode-se criar módulos. Eles são entidades únicas, que agrupam tanto a definição de um tipo de dado, como as funções e rotinas que os manipulam. São definidos ainda no módulo as interfaces, facilitando a utilização dessas funções, pois no programa, apenas é incluído a linha *use module\_name*, tornando acessível tudo que está contido no módulo. Esta declaração deve ser feita uma por módulo, o compilador não aceita na mesma declaração dois módulos.

Um módulo pode ter partes visíveis fora dele e partes exclusivamente internas ao módulo que não devem ser usadas fora dele. Isso se faz através do comando *private lista*. Assim, tudo que estiver na lista será visível apenas dentro do módulo.

Um módulo pode, ainda, usar outros módulos, criando uma certa hierarquia entre eles, que não poderá ser quebrada. São duas as regras básicas para o uso de módulos: um módulo não pode usar a si mesmo e um módulo não pode usar outro que o utilize.

Uma observação importante é que, se um módulo  $Y$  usa outro módulo  $X$  e, um módulo  $Z$  usa o módulo  $Y$ , as funções do  $X$  não estão acessíveis ao módulo  $Z$ , ou seja, não há transitividade entre os módulos; para usar as funções do módulo  $X$ , o módulo  $Z$  deve incluir o comando *use X*. A estrutura do comando de criação de módulo é dada pela figura 3.10.

```

module nome
    use...
    type...
    interface...
    private...
contains
    function...
    subroutine...
end module

```

Fig. 3.10 Estrutura do comando de criação de módulos

### 3.1.3.4 Compilação e Otimização

O compilador FORTRAN 90, linguagem de programação em que os testes foram implementados, possui uma diretiva de compilação (opção `-O opt, opt1, opt2, opt3`) que permite especificar as características de otimização a serem usadas durante a compilação, sendo 0 para nenhuma otimização até o valor 3 para uma otimização "agressiva". Conforme o nível especificado, 0,1,2 ou 3 implica certos níveis de otimização escalar, *tasking* e vetorização. Com o aumento da performance de execução, pode-se notar aumento no tempo de compilação e tamanho da compilação.

Os valores de *opt* específicos, tais como *vector1, scalar3, loopalign* selecionam características específicas. Os níveis de *opt* devem ser especificados em combinações lógicas. Se forem especificados valores incompatíveis no nível *opt*, o compilador aumentará a *opt* mais baixa para acomodar o nível mínimo necessário. Por exemplo, se for especificado `-O task3, scalar0` o compilador aumentará a o nível de otimização escalar para *scalar3*.

Somente um método pode ser usado para especificar otimização escalar, vetorização ou *tasking*. Os valores gerais de *opt* 0,1,2,3 não podem ser especificados em conjunto com os valores específicos de *opt scalar0, scalar1, scalar2, scalar3, vector0, vector1, vector2, vector3, task0, task1, task2* ou *task3*.

### 3.1.3.5 Arrays e Arrays dinâmicos

Os tipos de *arrays* são:

- *Array* atual - usado no programa principal ou em uma subrotina, tem uma parte da memória alocada para ele;
- *Array dummy* - um argumento *dummy* de uma subrotina, a armazenagem é na parte da memória associada ao *array* atual;



- *Arrays* de tamanho constante - um *array* que não muda de tamanho. Numa declaração de *array* constante os limites das dimensões só podem conter constantes
- *Arrays* de tamanho ajustável - um *array* onde o tamanho é determinado durante a execução;
- *Arrays* de tamanho assumido - É um *array* onde o tamanho é desconhecido, mas é assumido um valor suficientemente grande para conter todas as referências. A dimensão desconhecida é especificada por um asterisco.
- *Array deferred-shape* - é um *array* cujo tamanho é desconhecido, mas se conhece o intervalo onde ele pode variar.

Um *array* automático é um *array* atual, ajustável. Para declarar um *array* desse tipo basta fazer com que o último componente da declaração da dimensão contenha uma ou mais variáveis, funções ou elementos do *array*. Um *array* automático é usado, tipicamente, para dividir (*scratch*) o espaço em um subprograma. A memória para o *array* é alocada no momento que o subprograma é executado e liberada quando do final da execução. *Arrays* automáticos possibilitam ao programador alocar e liberar dinamicamente a memória implícita em uma chamada de subprograma.

Um *array allocatable* é um *array* automático e ainda *deferred-shape*.

Uma característica do Fortran 90 se refere à manipulação de *arrays*, por isso é chamada de seção de *array*. Uma seção de *array* é um grupo de elementos em um *array* que podem ser usados como um operando em uma expressão de *arrays*.

### 3.2 O PC486DX e a linguagem Pascal XSC

Neste item é descrito o ambiente mínimo que suporta a execução do Pascal XSC. Cabe ressaltar que o Pascal XSC é um pré-processador da linguagem C, existindo versões de Pascal XSC para diferentes compiladores C e para diferentes plataformas. O compilador C utilizado na instalação disponível no Instituto de Informática da UFRGS é do C GNU, também conhecido como C DJGPP.

Observa-se ainda que o Compilador Pascal-XSC é distribuído pela empresa alemã Numerik Software GmbH<sup>2</sup> e foi cedido para o Instituto de Informática da UFRGS para fins de estudo e pesquisa. Já o compilador C GNU é de domínio público. Ele está disponível na Rede, podendo ser importado através do utilitário *ftp*.

---

<sup>2</sup>Numerik Software GmbH, POBox 2232 D-76492 Baden-Baden Germany FAX 0049721694418

### 3.2.1 Requisitos de *Hardware*

O compilador Pascal XSC está disponível para várias plataformas, como pode ser visto em [DIV94d]. Em todas elas os cálculos produzem os mesmos resultados, portanto a aritmética que se tornou disponível por esta linguagem é independente da plataforma.

A versão disponível no Instituto de Informática da UFRGS é para equipamento do tipo IBM-PC 486 ou IBM-PC 386 com co-processador aritmético 80387. São necessários 640 KB de memória convencional e, pelo menos, 4 *Megabytes* de memória estendida. Para um melhor desempenho em termos de velocidade de processamento recomenda-se que o equipamento tenha 8 *Megabytes* de memória.

São necessários, ainda, um disco rígido de pelo menos 10 *Megabytes* de espaço disponível, *drive* de 1.2 ou 1.44 *Megabytes*, um monitor monocromático VGA.

O sistema operacional necessário para a instalação é o MS-DOS (ou compatível) versão 5.0 ou posterior.

O sistema Pascal XSC é composto pelos programas de configuração, pelo gerenciador, pelo compilador, pelos módulos padrões e por uma biblioteca de rotinas que compõem o sistema de execução de tempo real. O compilador não contém um gerador de código para uma máquina específica, ele produz um código C, conforme padrão C ANSI, na verdade ele é um pré-processador que converte para C. O código C é que, quando compilado, gera o código objeto para a máquina específica.

### 3.2.2 O Pascal XSC

PASCAL-XSC é uma linguagem de programação de propósito geral que proporciona condições especiais à implementação de algoritmos numéricos sofisticados, que verificam matematicamente os resultados. O novo sistema PASCAL-XSC tem as vantagens de ser portátil a várias plataformas, estando disponível para computadores pessoais (PC), estações de trabalho do tipo Sun e Hp, *mainframes* e supercomputadores. A portabilidade é garantida pelo uso de um compilador que traduz para a linguagem ANSI-C.

Este sistema proporciona uma completa simulação da aritmética de ponto-flutuante definida pelo padrão binário IEEE 754. Graças a isto, programas em PASCAL-XSC produzem resultados idênticos em todas as plataformas.

Pelo uso dos módulos matemáticos do PASCAL-XSC, algoritmos numéricos que providenciam alta exatidão e verificação automática de resultados podem ser facilmente programados. Além disso, a linguagem PASCAL-XSC simplifica o projeto de programas para as Engenharias e para a Computação Científica, graças à estrutura modular dos

programas, à possibilidade de definição de operadores, ao *overloading* de funções, rotinas e operadores, às funções e operadores com tipos arbitrários de dados e aos *arrays* dinâmicos.

Outras características presentes nos módulos da aritmética padrão para os tipos adicionais de dados numéricos incluem operadores e funções elementares com alta exatidão e com avaliação exata de expressões.

Os programas escritos em PASCAL-XSC são de fácil leitura, mesmo se existirem as operações com tipos de dados dos espaços matemáticos avançados, onde os operadores usados para as operações obedecem à notação matemática convencional. Existe ainda, uma grande quantidade de problemas numéricos que podem ser resolvidos pelas bibliotecas de rotinas com verificação automática do resultado. O PASCAL-XSC possui grandes facilidades para o desenvolvimento de tais rotinas.

PASCAL-XSC é uma extensão da linguagem de programação PASCAL para computação científica. Seu nome vem do inglês, **PASCAL eXtension for Scientific Computation**. Ele contém as características do Pascal Padrão; o conceito de operador universal (operador definido pelo usuário); funções e operadores com tipo de resultado arbitrário; *overloading* de rotinas, funções e operadores; *overloading* de atribuição de operadores; *overloading* de rotinas de entrada e saída (*read* e *write* genéricos); conceito de módulos; *arrays* dinâmicos; acesso a *subarrays*; conceito de *string*; arredondamento controlado; produto escalar ótimo (exato); tipo-padrão *dotprecision* (um formato de ponto-fixa abrangendo todo o intervalo do produto de valores em ponto-flutuante); aritmética especial para tipos-padrões de complexos e intervalos; aritmética de alta exatidão para todos os padrões; alta exatidão para funções elementares e avaliação exata de expressões (*#-expressions*).

Uma completa descrição da linguagem PASCAL-XSC e dos módulos aritméticos, assim como uma grande variedade de exemplos são dados em [KLA91] e [KLA92]. As novas características desta linguagem são discutidas a seguir.

### 3.2.2.1 Tipos-padrões de dados, operadores pré-definidos e funções

Além dos tipos de dados inteiro (*integer*) e real (*real*) do Pascal Padrão, existem ainda os tipos de dados numéricos da figura 3.11 disponíveis no PASCAL-XSC e, como se pode observar, os prefixos *r*, *i* e *c* são, respectivamente, abreviaturas de real, intervalo e complexo. Os tipos vetores e matrizes são definidos por *arrays* dinâmicos e podem ser usados como intervalos de índices arbitrários.

Um grande número de operadores são pré-definidos para estes tipos de dados nos módulos aritméticos do PASCAL-XSC. Todos estes operadores proporcionam resultados com máxima exatidão.

rvector	vetor de reais;
rmatrix	matriz real;
complex	tipo complexo;
cvector	vetor de complexos;
cmatrix	matriz complexa;
interval	tipo intervalo;
ivector	vetor de intervalos;
imatrix	matrix de intervalos;
cinterval	tipo intervalo complexo;
civector	vetor de intervalos complexos;
cimatrix	matrix de intervalos complexos;

Fig. 3.11 Tipos de dados numéricos do PASCAL-XSC

Comparando com o Pascal Padrão, existem 11 novos símbolos de operadores. Estes são os operadores  $\circ<$  e  $\circ>$ , onde  $\circ \in \{+, -, *, /$  para as operações com arredondamento direcionado para baixo e para cima e, os operadores  $**$ ,  $+*$  e  $><$  necessários em cálculos intervalares para a intersecção, união e para o teste da desconectividade (intersecção vazia).

Comparado com o Pascal Padrão, o PASCAL-XSC proporciona um extenso conjunto de funções elementares matemáticas. Estas funções estão disponíveis para os tipos *real*, *complex*, *interval* e *cinterval* com um nome genérico e proporciona um resultado com máxima exatidão. As funções para os tipos *complex*, *interval* e *cinterval* são providenciados em módulos aritméticos do PASCAL-XSC.

Além das funções elementares matemáticas, o PASCAL-XSC providencia funções de transferência, enumeradas na figura.3.12, entre tipos de dados necessários para a conversão de tipos de dados numéricos como intervalos e complexos, incluindo tipos escalares e *arrays*.

```

intval (ra,rb: real): interval;
inf (a: interval): real;
sup (a: interval): real;
compl (ra,rb:real): complex;
re (a: complex): real;
im (a: complex): real;

```

Fig.3.12 Funções de transferência



### 3.2.2.2 O Conceito de operador geral

Por um simples exemplo de uma soma de intervalos, as vantagens do conceito de operador geral podem ser demonstradas. Na concepção da definição de operador pelo usuário, existem duas formas de implementar operações entre intervalos como por exemplo a adição de duas variáveis do tipo intervalo. As variáveis intervalares são declaradas como do tipo intervalo, ou seja,

```
type interval = record inf, sup: real;
```

Estas formas seriam a implementação por rotina ou por função. Na implementação por rotina, ou seja, pela declaração de uma rotina, os operadores com arredondamento direcionado ( $+<$  e  $+>$ , os quais não são disponíveis no Pascal padrão) têm de ser programados por uma rotina (procedure) como na figura 3.13.

```
procedure intadd (a,b: interval; var c: interval);
begin
  c.inf := a.inf +< b.inf;;
  c.sup := a.sup +> b.sup;
end;
```

Fig. 3.13 Adição de intervalos por rotinas

Neste caso a expressão matemática  $z := a+b+c+d$ , seria implementada por três comandos de programação, ou seja, três chamadas da rotina da figura 3.13: `intadd(a,b,z); intadd(z,c,z); intadd(z,d,z)`.

No caso da implementação através da declaração de uma função (somente é possível em Pascal XSC, pois este tipo de declaração não é possível no Pascal padrão), como mostra a figura 3.14, a mesma expressão matemática  $z := a+b+c+d$ , seria expressa no uso recursivo da função, ou seja: `z:= intadd(intadd(intadd(a,b),c),d);`.

```
function intadd(a,b: interval): interval;
begin
  intadd.inf:= a.inf +< b.inf;
  intadd.sup:= a.sup +> b.sup;
end;
```

Fig. 3.14 Adição de intervalos por função

Em outros casos, a transcrição da fórmula matemática ficou um pouco complicada. Em comparação, se um operador é implementado em Pascal XSC, como mostra a figura 3.15, a expressão matemática seria programada pelo comando `z:=a+b+c+d;` ou seja, a notação matemática usual.

```

operator + (a,b:interval) intadd: interval;
begin
  intadd.inf:= a.inf +< b.inf;
  intadd.sup:= a.sup +> b.sup;
end;

```

Fig. 3.15 Adição de intervalos pela definição de operador

Além da possibilidade da sobrecarga dos símbolos de operadores, esta forma admite o uso do mesmo nome de operador. A declaração de tais tipos de operadores necessita ser precedida por uma declaração de prioridade.

O conceito de operador desenvolvido em Pascal XSC oferece a possibilidade da definição de um número arbitrário de operadores, ou a sobrecarga de símbolos de operadores ou nomes arbitrários para operadores e a implementação recursiva da definição de operadores.

Em Pascal XSC também é permitida a possibilidade da sobrecarga de atribuição := para permitir a notação natural para atribuições.

Uso da definição de atribuição para complexos é dado pela figura 3.16.

```

var
  c: complex;
  r: real;
operator := (var c:complex; r:real);
begin
  c.re:= r;
  c.im:= 0;
end;

r := 1.5;
c:= r; { número complexo < 1.5 , 0,0>}

```

Fig 3.16 exemplo: uso de complexos

### 3.2.2.3 *Overloading* de subrotinas

O Pascal Padrão providencia as funções matemáticas elementares *sin*, *cos*, *arctan*, *exp*, *ln*, *sqr* e *sqrt* somente para números reais. De forma a implementar a função seno para argumentos intervalos, um novo nome de função como *isin*(..) necessita ser usado, porque o *overloading* ou a sobrecarga do nome *sin* da função elementar seno não é permitida no Pascal Padrão. Entretanto, Pascal XSC admite sobrecarga de nomes de funções e rotinas, através da introdução do conceito de símbolo genérico na linguagem.

Assim os símbolos *sin*, *cos*, *arctan*, *exp*, *ln*, *sqr* e *sqrt* podem ser utilizados não somente para argumentos do tipo real, mas também para intervalos, números complexos e outros tipos. Para distinguir entre diversas funções e rotinas com o mesmo nome, o número e o tipo dos argumentos utilizados são o que determina o método, pois a rotina utilizada será do tipo do argumento. O tipo do resultado, entretanto, não é usado. Por exemplo, a rotina de rotação *rotate* pode ser definida para os tipos real, complexo e intervalo.

```
procedure rotate (var a,b: real);
procedure rotate (var a,b,c: complex);
procedure rotate (var a,b,c: interval);
```

O conceito de overloading também é aplicado às rotinas padrões de leitura (*read*) e escrita (*write*) de uma forma levemente modificada. O primeiro parâmetro da nova declaração da rotina de entrada ou saída necessita ser um parâmetro modificável (precedido pela palavra **var**) do tipo arquivo e, o segundo parâmetro representa a quantidade que deve entrar ou sair. Todos os demais parâmetros são interpretados como formato de especificação. Um exemplo é dado na figura 3.17, onde se tem uma rotina de escrita de complexo.

```
procedure write (var f: text; c: complex; w: integer);
begin
  write (f, '(', c.re:w, ', ', c.im:w, ')');
end;
```

Fig.3.17 Rotina de impressão de complexos

Quando se chama a carga de uma rotina de entrada e saída, o parâmetro de arquivo pode ser omitido, pois corresponde a uma chamada com os arquivos padrões de input ou output. O formato dos parâmetros precisa ser introduzido e separado por vírgulas. Entretanto, vários comandos de entrada ou saída podem ser combinados em um único comando, assim como no Pascal Padrão, como no exemplo a seguir, onde *r* é um real, *c* um complexo e o comando é: *write (r:10, c:5, r/5);*.

#### 3.2.2.4 O conceito de módulo

O Pascal Padrão assume basicamente que um programa consiste em um simples programa texto, o qual precisa ser preparado completamente antes que ele possa ser compilado e executado. Em muitos casos, é mais conveniente preparar um programa em várias partes, chamadas módulos, os quais podem então ser desenvolvidos e compilados independentes uns dos outros. Ainda mais, vários outros programas podem usar os componentes de um módulo sem ter que copiá-lo para dentro de seu código fonte ou recompilá-lo.

Para este propósito, um conceito de módulo foi introduzido no Pascal XSC. Este novo conceito oferece as possibilidades da programação modular, a verificação sintática e análise semântica além dos limites dos módulos e a implementação de pacotes aritméticos como módulos padrões.

Um módulo é introduzido pela palavra-chave **module** seguida por um nome e um ponto e vírgula. Seu corpo é bem similar à estrutura de um programa normal, com a exceção de que a palavra símbolo **global** pode ser usada diretamente na frente das palavras-chave de declaração de constante (**const**), tipo (**type**), variável (**var**), rotina (**procedure**), função (**function**) e operador (**operator**) e diretamente depois da palavra símbolo **use** e do sinal de igual na declaração de tipo.

Portanto, é possível declarar tipos privados assim como tipos não privados. A estrutura interna de um tipo privado não é conhecida fora da declaração do módulo. Objetos deste tipo privado podem somente ser usados e manipulados através de rotinas, funções e operadores definidos pela declaração do módulo.

### 3.2.2.5 Arrays dinâmicos

No Pascal Padrão não há maneira de declarar tipos ou variáveis dinâmicas. A única maneira de gerenciar dinamicamente a memória no Pascal Padrão é através da alocação e liberação de objetos de tamanho fixo, os quais são referenciados por ponteiros (*pointers*).

Por exemplo, pacotes com operações de vetores e matrizes são tipicamente implementadas com dimensões fixas (máxima). Por esta razão, somente parte da memória alocada é usada, se o usuário estiver resolvendo um problema de dimensão menor do que a definida. O conceito de *array* dinâmico acaba com esta limitação.

Este novo conceito pode ser descrito pelas características de ser dinâmico em rotinas e funções, ter alocação automática e liberação das variáveis locais dinâmicas, utilizar economicamente o espaço de armazenamento, ter acesso a linhas e colunas de *arrays* dinâmicos e ter compatibilidade entre os tipos estáticos e dinâmicos de *arrays*.

*Arrays* dinâmicos precisam ser marcados com a palavra-chave **dynamic**. A grande desvantagem de esquemas de *arrays* "pré-formados", disponíveis no Pascal Padrão, é que eles podem ser acessados somente por parâmetros que sejam índices, não sendo permitidos parâmetros que sejam variáveis ou resultados de funções. Portanto, esta característica não é plenamente dinâmica.

No Pascal XSC, *arrays* dinâmicos e estáticos podem ser usados da mesma forma. Entretanto, *arrays* dinâmicos não podem ter componentes de outras estruturas de dados, por exemplo *records*.

A declaração do tipo *array* dinâmico bi-dimensional pode ser feita por:



**type matrix = dynamic array [\*,\*] of real;**

Também é possível definir diferentes tipos de dinâmicos, com correspondentes estruturas sintáticas. Por exemplo, pode ser útil, em algumas situações, identificar os coeficientes de um polinômio como componentes de um vetor ou vice-versa. Como Pascal é uma linguagem estritamente orientada a tipos de dados, esta equivalência de estrutura de *arrays* necessita ser combinada através de uma adaptação prévia.

Além de acessar cada componente da variável, Pascal XSC oferece a possibilidade de acesso a *subarrays*. Se uma componente da variável contém um \* ou um intervalo de uma expressão de índice, refere-se ao *subarray* com um todo ou um específico intervalo de índice na correspondente dimensão.

### 3.2.2.6 Expressões exatas

A teoria da aritmética computacional requer a implementação do produto escalar com somente um arredondamento de acordo com a seguinte definição, dada originalmente em [BOH93].

"Dados dois vetores  $x$  e  $y$  com  $n$  componentes cada, em ponto-flutuante, e um pré-estabelecido modo de arredondamento  $\square$ , o resultado  $s$  da operação produto escalar em ponto-flutuante (aplicado em  $x$  e  $y$ ) é definido por :

$$s := \square(\bar{s}) := \square(x, y) = \square\left(\sum_{i=1}^n x_i * y_i\right), \quad n \geq 1$$

onde todas as operações aritméticas são matematicamente exatas. Então  $s$  pode ser calculado como se fosse um resultado correto intermediário  $\bar{s}$  em precisão infinita e com intervalo do expoente não limitado na primeira rotina e, então, arredondado para o formato desejado de ponto-flutuante de acordo com o modo de arredondamento  $\square$ ."

Portanto o resultado da operação precisa ser o resultado correto do produto escalar com apenas uma aplicação do arredondamento no final.

A implementação de algoritmos de inclusão com verificação automática do resultado ou validação (ver [KAU87, KUL89, KUL88, ULL90]) faz um uso extensivo da avaliação exata do produto escalar. Para avaliar este tipo de expressão foi introduzido um novo tipo de dado *dotprecision*. Variáveis do tipo *dotprecision* podem assumir qualquer valor possível, resultante da avaliação de expressões de produto escalar sem perda de exatidão (como é mostrado em [KUL81, KUL89]). Baseado neste tipo de dados, as assim chamadas **expressões exatas** (*#-expressions*), podem ser formuladas por um símbolo exato ( $\#$ ,  $\#*$ ,  $\#<$ ,  $\#>$ , ou  $\#\#$ ) seguido de uma expressão exata entre parêntesis. A expressão exata precisa ter a forma de expressão do produto escalar na estrutura escalar, vetorial ou matricial, e é avaliada sem qualquer erro de arredondamento. Por causa disto, o resultado de uma expressão exata tem um erro de no máximo 1 *ulp*, isto é, de no máximo uma unidade na última casa da mantissa (*unit in the last mantissa place*).

A sintaxe das expressões exatas é da forma: #<símbolo> (expressão exata). Para obter resultados não arredondados ou corretamente arredondados de uma expressão de produto escalar, o usuário precisa escrever a expressão entre parêntesis precedida pelo símbolo #, podendo, opcionalmente, ser seguido por um símbolo de modo de arredondamento. A tabela 3.1 mostra os possíveis modos de arredondamento com respeito à forma da expressão do produto escalar.

Tabela 3.1 Tabela dos modos de arredondamento para expressões exatas

Símbolo	Forma da expressão	Modo de arredondamento	Simb.Mat
#*	scalar, vector, matrix	simétrico	□
#<	scalar, vector, matrix	para baixo	▽
#>	scalar, vector, matrix	para cima	△
##	scalar, vector, matrix	menor intervalo contido	◇
#	somente scalar	exato sem arredondamento	

Na prática, expressões de produto escalar podem conter um grande número de termos resultando numa notação implícita muito incômoda. Para aliviar esta dificuldade matemática, o símbolo somatório  $\sum$  é usado. Se por exemplo A e B são matrizes n-dimensionais, então a avaliação de  $d := \sum A_{ik} \cdot B_{kj}$ , para  $k=1$  até n, representa uma expressão de produto escalar. O Pascal XSC providencia uma notação equivalente para este propósito, o comando **sum**, sendo a expressão correspondente dada a seguir, onde d é uma variável do tipo *dotprecision*.

$$d := \#(\text{for } k:=1 \text{ to } n \text{ sum } (A[i,k]*B[k,j])),$$

Expressões produto escalar ou expressões exatas são usadas principalmente no cálculo do resíduo (ou erro). No caso de sistemas de equações lineares  $Ax = b$ , onde  $A \in \mathbf{R}^{n \times n}$ ,  $x, b \in \mathbf{R}^n$ ,  $Ay \approx b$  é tomado como exemplo. Então a inclusão do erro é dada por  $\hat{\Delta}(b - Ay)$ , o qual pode ser calculado em Pascal XSC pela fórmula **##(b-Ay)**, com apenas uma operação intervalar de arredondamento para cada componente do vetor intervalar resultante. Para se terem inclusões verificadas em sistemas de equações lineares é necessário avaliar a expressão do erro  $\hat{\Delta}(E - RA)$ , onde  $R \approx A^{-1}$  e E é a matriz identidade. Em Pascal XSC esta expressão pode ser programada por **##(id(A)-R\*A)**; onde uma matriz intervalar é calculada com somente um arredondamento por componente. A função **id(.)** é definida no módulo da aritmética real de vetores e matrizes e produz uma matriz identidade do mesmo tipo dos argumentos.

### 3.2.2.7 O conceito de *string*

As ferramentas para tratamento de seqüências (*string*) de caracteres no Pascal Padrão não admitem um processamento de texto conveniente. Por esta razão, o conceito de seqüência foi integrado na definição da linguagem Pascal XSC, a qual admite um tratamento conveniente de informações textuais, e usa o conceito de operador, mesmo em computação simbólica. No novo tipo de dado *string*, o usuário pode trabalhar com seqüências de até MAXINT caracteres. Quando estiver declarando variáveis do tipo *string*, o usuário pode especificar o tamanho máximo da seqüência menor que MAXINT. Portanto uma seqüência *s* que pode ter até 40 caracteres é declarada por: **var s: string[40];**

As operações padrões disponíveis ao tipo *string* são: concatenação; tamanho atual; conversões *string* para *real*, *string* para *integer*, *real* para *string* e *integer* para *string*; retirada de *substring*; posição da primeira aparição; operadores relacionais <=, <, >=, >, <>, = e *in*.

## 4 RESOLUÇÃO DE SISTEMAS LINEARES

### 4.1 Aplicabilidade

Um problema de grande interesse prático, que aparece, por exemplo, em cálculo de estruturas, redes elétricas e solução de equações diferenciais, é o da resolução numérica de um sistema linear de ordem  $n$ .

São dados alguns exemplos com o objetivo de ilustrar a aplicação de sistemas lineares na solução de vários problemas, os quais são modelados e recaem na simples solução de sistemas. Aproveitam-se estes exemplos para revisar conceitos e dificuldades pertinentes ao assunto.

Como primeiro exemplo<sup>1</sup>, considera-se um certo produto a ser transportado, que é fornecido em dois tipos de embalagens diferentes, constando para ambas o peso bruto de 400 gramas. A empresa transportadora quer saber o peso exato de cada tipo. Ao pesar as duas embalagens em uma balança, cuja precisão vai até 100 gramas, não se verificou diferença significativa; então foi idealizada a seguinte situação: ao colocar onze embalagens do primeiro tipo e cinco do segundo tipo, lê-se o peso de 6.4 kg, a seguir, retiram-se duas do primeiro tipo e uma do segundo tipo, resultando no peso lido de 5.2 kg. Esta situação pode ser expressa pelo seguinte sistema de equações, onde se adotará  $x_1$  para o peso da primeira embalagem e  $x_2$  para o da segunda. Estes valores serão obtidos resolvendo o sistema:

$$\begin{cases} 11x_1 + 5x_2 = 6.4 \\ 9x_1 + 4x_2 = 5.2 \end{cases}$$

Percebe-se ao encontrar os valores  $x_1 = x_2 = 0.4$ , que a precisão não é suficiente para a determinação correta dos pesos, então o processo é repetido em uma balança de precisão de até 10 gramas. Os valores lidos nas pesagens, nos dois casos, são respectivamente 6.45 kg e 5.16 kg. Resolvendo-se o sistema modificado

$$\begin{cases} 11x_1 + 5x_2 = 6.45 \\ 9x_1 + 4x_2 = 5.16 \end{cases}$$

obtem-se como resultado os pesos  $x_1 = 0$  kg e  $x_2 = 1.29$  kg. Observa-se que há uma descalibragem nesta segunda balança, de cerca de 0.05kg, mas isto gerou valores absurdamente diferentes. Com este exemplo, constata-se o problema da instabilidade do problema ocasionado pela quase singularidade, pois as retas que representam as equações são quase coincidentes (paralelas). Este sistema, na verdade, é mal condicionado.

---

<sup>1</sup>Retirado do livro Curso de cálculo numérico de Vitoriano Ruas de Barros Santos, identificado por [SAN76].



Outro exemplo do uso de sistemas de equações está no cálculo das correntes elétricas pelas Leis de Kirchoff (1847). As leis de Kirchoff são de fundamental importância na resolução dos circuitos elétricos. Seja  $E$  a força eletromotriz,  $I$  a intensidade da corrente elétrica e  $R$  a resistência, os lemas seguem os enunciados:

- A soma algébrica das intensidades das correntes que concorrem em um ponto é nula:  $\sum I_i = 0$ .
- A soma algébrica das forças eletromotrizes e das quedas ohmônicas de potência (produtos das resistências pelas intensidades das correntes) é nula:  $\sum E_i + \sum I_i R_i = 0$ .

Para efetuar estas somas algébricas é necessário estabelecer corretamente o sinal correspondente da corrente elétrica. Por convenção, supor-se-á que a corrente vai do pólo negativo da fonte de força eletromotriz ao pólo positivo, através do circuito externo.

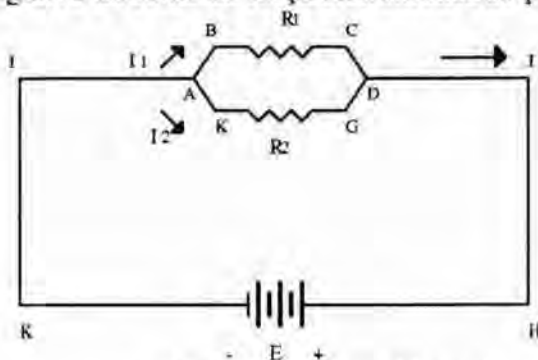


Fig 4.1 Circuito elétrico I

No caso da figura 1.1, supondo serem conhecidos  $E$  e as resistências  $R_1$  e  $R_2$ , para se determinar as intensidades  $I_1$  e  $I_2$  das correntes que circulam, respectivamente, pelos condutores  $ABCD$  e  $AFGD$  e a intensidade  $I$  devem-se aplicar as duas leis de Kirchoff. No nó  $A$ , deve-se ter:  $I - I_1 - I_2 = 0$ , pois  $I$  chega em  $A$  e  $I_1$  e  $I_2$  saem de  $A$ .

No circuito  $ABCDHKA$ , a segunda lei permite escrever que  $-I_1 R_1 + E = 0$  e no circuito  $AFGDHKA$ , obtém-se de forma análoga  $-I_2 R_2 + E = 0$ .

Portanto, tem-se:

$$I = I_1 + I_2 = \frac{E}{R_1} + \frac{E}{R_2} = E \left( \frac{1}{R_1} + \frac{1}{R_2} \right)$$

De um modo geral, chega-se a um sistema de equações lineares. Para melhor ilustrar considere o exemplo do circuito elétrico retirado de [CLA89], exemplo 3.6, p.65.

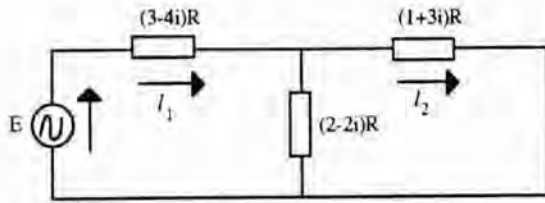


Fig 4.2 Circuito Elétrico II

Usando as leis de Kirchoff , solução a é:

$$E-(3-4i)RI_1-(2-2i)R(I_1-I_2) = 0$$

$$(2-2i)R(I_2-I_1)+(1+3i)RI_2 = 0$$

fazendo  $x_1 = I_1 / (E/R)$  e  $x_2 = I_2 / (E/R)$  obtém-se

$$\begin{bmatrix} 5-6i & -2+2i \\ -2+2i & 3+i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

ou ainda, descomplexificando:

$$\begin{bmatrix} 5 & -2 & 6 & -6 \\ -2 & 3 & -2 & -1 \\ -6 & 2 & 5 & -2 \\ 2 & 1 & -2 & 3 \end{bmatrix} \begin{bmatrix} x_{1r} \\ x_{2r} \\ x_{1i} \\ x_{2i} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Este sistema é, então, resolvido para se obter a solução.

## 4.2 Formalização do problema

Um sistema de equações lineares pode ser escrito na forma:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \dots + a_{3n}x_n = b_3 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \dots + a_{nn}x_n = b_n \end{cases} \quad (4.1)$$

onde  $n$  é o número de incógnitas do sistema, também denominado de ordem do sistema.

Todo sistema de equações pode ser reescrito na forma de uma matriz de coeficientes  $A$  multiplicada pelo vetor  $X$  (das incógnitas), resultando no vetor dos termos independentes  $B$ , ou seja  $AX = B$ , onde:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \quad (4.2)$$

Um problema típico de resolução de equações é: dado um sistema de equações lineares de ordem  $n \times n$ , quer se achar os valores das incógnitas, que multiplicados por  $A$ , produzem o vetor  $B$ .

Uma solução para um sistema linear é um conjunto de valores das  $n$  incógnitas que satisfaça a todas as equações. Um sistema linear pode ser classificado quanto ao número de soluções em: **compatível**, quando apresenta solução e **incompatível** caso contrário.

O sistema é dito **homogêneo** quando o vetor dos termos independentes é nulo, ou seja,  $b_i = 0$ , para todo  $i=1,2, \dots, n$ . Todo sistema homogêneo é compatível, pois admite pelo menos a solução trivial, que tem como vetor solução  $X$  o vetor nulo.

Os sistemas compatíveis podem ainda ser classificados em **determinados**, quando apresentam uma única solução e **indeterminados**, caso contrário. Ou seja, se o conjunto solução é único, o sistema é determinado ou não singular. Se existirem uma infinidade de tais conjuntos, o sistema é indeterminado.

Essas considerações sobre unicidade ou não da solução devem se limitar ao campo teórico, pois como se está lidando com problemas numéricos, tem-se aproximações e arredondamentos que podem conduzir a soluções diferentes e, em muitos casos, inteiramente erradas, como no caso de problemas instáveis ou mal condicionados.

#### 4.2.1 Revisão de Matrizes e Álgebra matricial

Os tipos de matrizes mais usuais são:

- a) Matriz retangular: uma matriz  $m \times n$  é dita retangular quando  $m \neq n$ ;

b) Matriz quadrada: uma matriz  $m \times n$  é dita quadrada quando  $m=n$ , neste caso, diz-se que a matriz é de dimensão  $n$ , não sendo necessário dizer  $n \times n$ ;

c) Matriz coluna: quando a matriz só possui uma coluna, ou seja  $n=1$ . Também chamada de vetor;

d) Matriz triangular: uma matriz triangular é um caso especial de matriz quadrada, na qual todos os elementos de um lado da diagonal principal são nulos. Se forem nulos os elementos abaixo da diagonal, a matriz é denominada triangular superior; se forem nulos os elementos acima da diagonal, a matriz é chamada de triangular inferior;

e) Matriz diagonal: é uma matriz quadrada que só possui elementos não nulos na diagonal principal;

f) Matriz tridiagonal: é uma matriz quadrada que possui os elementos não nulos na diagonal principal e na primeira diagonal acima e abaixo dela;

g) Matriz simétrica: é uma matriz quadrada cujos valores são simétricos com relação à diagonal principal, ou seja  $a_{ij} = a_{ji}$ ;

h) Matriz densa: uma matriz onde a maioria dos seus elementos são não nulos;

i) Matriz esparsa: uma matriz onde cerca de dois terços dos elementos são nulos. Um caso especial de matriz esparsa é do tipo banda, onde os elementos não nulos se concentram em certas bandas.

j) Matriz conjugada: dada uma matriz complexa, define-se a matriz conjugada como a matriz composta pelo conjugado de cada elemento complexo  $a_{ij}$ .

k) Matriz Hermitiana: é definida como sendo a transposta da matriz conjugada de  $A$  e anotada por  $A^H$ .

l) Matriz de Hilbert de ordem  $n$ : são matrizes onde todos os elementos são menores ou iguais a um. Uma matriz de Hilbert de ordem  $n$  é calculada da forma  $H_n = (h_{ij}) = [1/(i+j-1)]$  para  $i=1, \dots, n$  e  $j=1, \dots, n$ .

Dadas as matrizes reais  $A=(a_{ij})$  e  $B=(b_{ij})$  de ordem  $m \times n$ , define-se a soma das matrizes  $C=A+B$  pelas somas dos elementos  $a_{ij}$  e  $b_{ij}$  correspondentes de  $A$  e  $B$ , gerando o elemento  $c_{ij}$  de  $C$ , ou seja:  $c_{ij} = a_{ij} + b_{ij}$ , onde  $i=1, \dots, m$  e  $j=1, \dots, n$ .

O produto de duas matrizes, anotado por  $C=A.B$  é definido em função do somatório dos produtos dos elementos das linhas de  $A$  pelos elementos da coluna de  $B$ , ou seja os elementos da matriz produto  $C$ , são definidos por:  $c_{ij} = \sum_{k=1, \dots, n} a_{ik}.b_{kj}$ , onde  $k=1, \dots, n$ ,  $i=1, \dots, m$ ,  $j=1, \dots, n$ .



Observa-se que, num produto de matrizes, o número de colunas da primeira matriz tem que ser igual ao número de linhas da segunda matriz e a matriz resultante  $C$  terá a ordem correspondente ao número de linhas da primeira e o número de colunas da segunda, por exemplo, se  $3 \times 5$  é a ordem de  $A$  e  $5 \times 6$  a ordem de  $B$ , então o produto é válido e a matriz resultante  $C$  terá a ordem  $3 \times 6$ . O elemento  $c_{2,3}$  corresponde ao somatório do produto dos elementos da segunda linha de  $A$  com os elementos da terceira coluna de  $B$ . O produto de matrizes não é comutativo.

O produto de uma matriz por um escalar (um número real) é definido como sendo a matriz resultante  $C$ , cujos elementos são o produto do real pelos elementos de  $A$ , ou seja,  $C=(c_{ij})=rA=(r.a_{ij})$ , para  $i=1,\dots,m$  e  $j=1,\dots,n$ . O mesmo é válido para operações de adição e subtração de escalares.

A divisão de matrizes não é uma operação legítima na álgebra matricial, entretanto o conceito de matriz inversa guarda certa semelhança com a notação de divisão. A inversa da matriz quadrada  $A$  é definida como a matriz  $B$ , tal que o produto de  $A$  por  $B$  e vice versa, produz a matriz identidade, ou seja,  $A.B=B.A=I$ .

Potências de matrizes são definidas como casos particulares de produtos de fatores iguais. Entretanto, cabe ressaltar que  $A$  elevado a zero resulta na identidade;  $A$  elevado a um é a própria  $A$ ,  $A^2$  é  $A.A$  e, por fim,  $A^n$  é igual a um produto de  $n$  fatores iguais a  $A$ . Estas operações entre matrizes reais são estendíveis a matrizes de intervalos; para tanto se utilizam de matrizes intervalares operadas por operações intervalares.

Os métodos numéricos para a solução de Sistemas de Equações Lineares mais conhecidos podem ser chamados de métodos pontuais, por operarem com números em ponto-flutuantes. Existem outros métodos que se utilizam de intervalos, sendo chamados de métodos intervalares. Cada um destes métodos possui vantagens e desvantagens, em função do tamanho do sistema, tipo de matriz dos coeficientes e do número de operações necessárias para sua realização.

Uma das grandes dificuldades na resolução de sistemas é o problema da instabilidade. Esta é ocasionada pelas limitações e pelos erros de aproximações que estão ligados aos algoritmos usados e aos problemas em estudo, podendo levar a situações indesejáveis. Alguns problemas são sensíveis a pequenas mudanças nos dados de entrada, como por exemplo no sistema de equações:  $\{x + 2y = 3$  e  $\{0.499x + 1.001y = 1.5$ , que tem por solução  $x=y=1.0000$ . Mas se a segunda equação for substituída por:  $\{0.5x + 1.0y = 1.5$ , a solução passa a ser  $x = 3.0$  e  $y = 0.0$  o que evidencia que uma pequena variação nos coeficientes resulta numa solução bem diferente da inicial.

Diz-se, por definição, que um problema matemático é instável, mal condicionado ou mal posto se sua solução for sensível a pequenas mudanças nos dados de entrada. Caso pequenas alterações nos dados de entrada sejam acompanhados de pequenas variações na solução, o problema é dito estável ou bem condicionado, ou ainda, bem posto.

Para se falar de continuidade, deve-se ser capaz de medir distâncias entre elementos do conjunto, que no caso são vetores e matrizes. Seja a métrica uma das normas de matrizes abaixo:

$\|A\|_C := A$  maior soma de coluna de  $A$ , tomados os elementos em módulo

$\|A\|_L := A$  maior soma de linha de  $A$ , tomados os elementos em módulo.

$\|A\|_E := A$  raiz quadrada da soma de todos os quadrados dos elementos da matriz

Sejam os sistemas  $AX = B$  e  $AX' = B'$  (perturbado). Então,  $B-B' = AX-AX'=A(X-X')$ , e  $X-X' = A^{-1}(B-B') \Rightarrow |X-X'| \leq \|A^{-1}\| \cdot |B-B'|$ , mas como:  $AX=B \Rightarrow |B| \leq \|A\| \cdot |X|$ , então implica que  $1/|X| \leq \|A\|/|B|$  e portanto,  $|X-X'|/|X| \leq \|A\| \cdot \|A^{-1}\| \cdot (|B-B'|/|B'|)$ .

Logo o erro relativo em  $X$ , devido à incerteza na determinação de  $B$  é proporcional ao erro relativo em  $B$ . Temos ainda que o coeficiente de propagação é  $\|A\| \cdot \|A^{-1}\|$ .

Define-se, portanto, a grandeza  $\|A\| \cdot \|A^{-1}\|$ , denominada *condição* ou *condicionamento* da matriz  $A$  e anotada por  $COND(A)$ , como sendo:  $COND(A) = \|A\| \cdot \|A^{-1}\|$  e pode-se observar que: quanto maior for o  $COND(A)$ , pior condicionada é a matriz  $A$ . Não é verdade que o condicionamento seja dependente do determinante, pois podem haver casos em que o  $\det(A)$  é aproximadamente zero, cuja matriz não é mal condicionada;  $COND(A) = \|A\| \cdot \|A^{-1}\| \geq \|A \cdot A^{-1}\| = \|I\| \geq 1$ . Se  $COND(A) \leq 10^j$ , então até  $j$  algarismos significativos de exatidão poderão ser perdidos na solução do sistema, devido ao mal condicionamento da matriz.

Na resolução de um sistema de equações, pode-se, então, encontrar problemas que tornam o método utilizado ineficiente, produzindo resultados incorretos. Como exemplo deste tipo de problema tem-se o número de operações que são necessárias para a resolução do sistema de equações. Outro problema a destacar é a proximidade da singularidade dos sistemas lineares. Caso a matriz seja mal-condicionada ou instável, quaisquer erros, como por exemplo os de arredondamentos, podem gerar um resultado incorreto e esta instabilidade ou estabilidade deve ser analisada a fim de evitar surpresas durante ou no final da computação.

Os métodos numéricos para a solução de Sistemas de Equações Lineares podem ser pontuais ou intervalares. Os pontuais manipulam números reais ou complexos, podendo ser diretos, iterativos ou compactos. Os métodos intervalares manipulam intervalos de números reais ou complexos, podendo ser diretos ou iterativos. Cada um destes métodos possui vantagens e desvantagens, em função do tamanho do sistema, tipo de matriz dos coeficientes e do número de operações necessárias para sua realização.

Na resolução de um sistema de equações, podem-se encontrar problemas que tornam o método utilizado ineficiente produzindo resultados incorretos. Como exemplo deste tipo de problema tem-se o número de operações que são necessárias para a

resolução do sistema de equações. Há métodos, como por exemplo os de Eliminação de Gauss, que serão eficientes apenas para sistemas de pequeno e médio porte.

Para fins de estudo, realizar-se-á uma divisão dos métodos numéricos em métodos intervalares e métodos pontuais. Serão descritos os métodos, suas características e as condições para se ter algoritmos numéricos com verificação automática do resultado. Esta metodologia identifica quando o sistema não possui solução (identificando se a matriz é singular) ou fornece a solução na forma de um intervalo.

Referências bibliográficas, para cada um dos métodos descritos a seguir; foram levantadas entre o acervo bibliográfico das bibliotecas da Informática e da Matemática da UFRGS e publicadas em [DIV90] e, posteriormente, publicada em [HOL95] com uma versão atualizada. Em cada método, são dadas apenas as obras clássicas como referência.

### **4.3 Métodos Pontuais**

Os métodos pontuais manipulam números reais ou complexos, quando implementados em computadores os dados são pontos flutuantes. Estes métodos podem ser métodos diretos, iterativos ou compactos, sendo que, também, cada um possui vantagens e desvantagens, dependendo do tipo de sistemas de equações e do tipo da matriz dos coeficientes.

Entre os métodos pontuais diretos serão descritos o método de Cramer, a Família dos Métodos de Eliminação de Gauss e o método de Gauss-Jordam. Com relação aos iterativos serão apresentados o método de Gauss-Jacobi, o de Gauss-Seidel e o método da Sobre-relaxação. Os métodos compactos apresentados serão o método de Banachiewicz, o método de Cholesky e o método de Cholesky para sistemas simétricos.

#### **4.3.1 Métodos Diretos**

Um método dito direto é aquele na qual a solução exata  $X$  é obtida realizando-se um número finito de operações aritméticas. Exemplos de métodos diretos pontuais são o método de Cramer e a família dos Métodos de Eliminação de Gauss (MEG).

O método de Cramer é um método teórico eficaz, mas na prática ineficiente, devido à grande quantidade de operações de adição e multiplicação necessárias para o cálculo do vetor solução  $X$ .

Os métodos diretos mais conhecidos e mais usados para resolução de um sistema de equações denso de porte pequeno a médio são os métodos de Eliminação de Gauss, ou algoritmo de Gauss.

#### 4.3.1.1 Família dos métodos de Eliminação de Gauss.

Os métodos diretos mais conhecidos e mais usados para resolução de um SELA denso de porte pequeno a médio são os métodos de Eliminação de Gauss ou algoritmo de Gauss. Por sistema de pequeno porte, entende-se uma ordem de até 30; para médio porte podem-se ter sistemas de ordem 50.

Estes métodos buscam calcular a solução exata do sistema através da realização de um número finito de operações aritméticas nos reais. Eles são caracterizados por duas etapas: a triangularização e retrossubstituição. A etapa da triangularização, caracteriza-se por transformar a matriz em triangular superior (todos os elementos abaixo da diagonal principal são nulos), usando apenas operações aritméticas. A etapa da retrossubstituição é a etapa que determina os valores das componentes do vetor solução  $X$ .

Com o uso da aritmética de ponto-flutuante pelo computador, ocorrem erros que se propagam para o resultado, afetando a qualidade do mesmo. Na tentativa de minimizar os erros foram desenvolvidas várias versões que são relacionadas na figura 4.3.

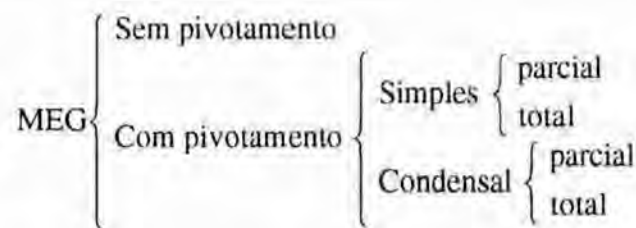


Fig.4.3 - Família dos métodos de Eliminação de Gauss

A técnica do pivotamento é uma alternativa de utilizar sempre o maior valor como denominador, produzindo um quociente, em módulo entre 0 e 1, que é a zona de maior concentração de números de máquina no sistema de ponto-flutuante, ocasionando menor perda de dígitos significativos.

No caso de matrizes reais, a matriz inversa e a solução de sistemas de equações lineares deverão ser encontradas, os métodos de eliminação de Gauss, com seus posicionamentos parciais e totais, é o método, geralmente, mais recomendado.

Os métodos de eliminação de Gauss, com todas as operações realizadas na aritmética complexa são, provavelmente, os mais eficientes para a inversão de matrizes complexas e para a solução de sistemas de equações lineares complexas.

Maiores detalhes sobre estes métodos podem ser encontrados em [CLA83, CLA89, VAN78, RAL60, RAL65].



**a) Sem pivotamento.** Esta versão do MEG que não utiliza pivotamento, é a versão mais simples e parecida com a estudada na álgebra, conhecida como "Método das Operações Elementares". Também não se efetuam trocas de linhas, a não ser que existam zeros na diagonal principal. Neste caso, uma troca para retirar o zero da diagonal se faz necessária.

MEG - Sem pivotamento possui duas etapas: a triangularização e a retrossubstituição. Na triangularização, as operações para transformar a matriz A em uma matriz triangular são da forma:

$$LEZ = \frac{-EZ}{ED} \times LD + LEZ \quad (4.3)$$

onde: EZ - Elemento a ser zerado;  
ED - Elemento da diagonal principal, da mesma coluna que EZ;  
LEZ - Linha que contém o elemento a ser zerado EZ;  
LD - Linha que contém o elemento da diagonal ED.

**b) MEG- Com pivotamento simples.** A técnica com pivotamento simples é uma tentativa de minimizar os erros de arredondamento nas operações durante a triangularização da matriz e, conseqüentemente, minimizar a propagação destes para o resultado do sistema. O pivotamento simples consiste em nomear um PIVO, um elemento escolhido, que ocupará a posição na diagonal principal e servirá de base nas operações elementares para zerar os demais elementos de sua coluna. Este elemento sempre é escolhido, considerando o elemento da diagonal principal para baixo e que seja o MAIOR em módulo. Para que o PIVO ocupe a diagonal, troca de linhas podem ser necessárias.

$$PIVO(j) = \max_{i=j \text{ até } N} \{|a_{ij}|\} \quad (4.4)$$

**c) MEG- Com pivotamento total.** O pivotamento total é uma técnica que assegura que os elementos que ocupam a posição da diagonal sejam maiores, em módulo, que os demais elementos de linhas e colunas, e para que isto ocorra, muitas vezes são necessárias trocas de linhas e colunas. Um detalhe muito importante deve ser observado:

- Uma troca de coluna em A ocasiona troca de linha no vetor X, uma vez que cada coluna de A corresponde aos coeficientes de uma das incógnitas de X, trocando a ordem das colunas ocasiona a troca de ordem das incógnitas (muda a ordem das linhas). O vetor dos termos independentes B não se altera.

#### 4.3.1.2 Método de Gauss-Jordan

O método de Gauss-Jordam apresenta duas variações básicas. Uma que é muito utilizada para calcular a inversa de uma matriz e, então, resolver o sistema. A outra é na verdade o MEG com pivotamento total, ou seja, diagonaliza a matriz A, obtendo a solução X sem a necessidade de retrossubstituição.

**a) Gauss-Jordan (com inversa).** Esta versão de método é muito utilizada para calcular matrizes inversas. O processo parte da concatenação de A com a matriz identidade I, transformando a matriz A em identidade através de operações elementares. Ao final do processo, onde se tinha a matriz identidade concatenada, tem-se a matriz inversa. Uma vez de posse da matriz inversa  $A^{-1}$  de A, pela álgebra pode-se resolver o sistema por (4.5), ou seja, o vetor solução X pode ser calculado, simplesmente multiplicando a inversa  $A^{-1}$  pelo vetor de termos independentes B.

$$\text{Se } AX = B, \text{ então } X = A^{-1} B \quad (4.5)$$

**b) Gauss-Jordan (pivotalamento total).** É outra variação do método de Gauss-Jordan, quando não se dispõe da matriz inversa, ou não se quer utilizá-la. Parte-se, então, da matriz A concatenada com o vetor dos termos independentes B. Transforma-se, por operações elementares, a matriz A na matriz identidade. Como todas as operações elementares realizadas sobre A também foram efetuadas sobre B, ao final do processo, tem-se, no lugar dos termos independentes B, o vetor solução X.

Maiores detalhes sobre estes métodos podem ser encontrados em [CLA83, CLA89, VAN78, RAL60, RAL65].

### 4.3.2 Métodos Iterativos

Um método é dito iterativo quando a solução X é obtida como o limite de uma sequência de aproximações sucessivas  $X_1, X_2, \dots, X_m$ . Os métodos iterativos podem ser considerados como uma generalização do método de ponto-fixo para equações. No lugar de uma equação tem-se um sistema de equações  $AX=B$ , de ordem n, para o qual se obtém uma "função vetorial de iteração  $G(X)$ " isolando uma variável em cada uma das equações. Os valores iniciais são utilizados para calcular a primeira iteração e, estes, para os cálculos da segunda iteração. Portanto, os valores das iterações são calculados pelos valores da iteração anterior, entretanto, há a necessidade de se estabelecer um critério de parada adequado. A convergência nestes métodos é garantida quando a matriz dos coeficientes é diagonalmente dominante.

#### 4.3.2.1 Método de Gauss-Jacobi

Este método utiliza uma "função vetorial de iteração"  $G(X)$ , para o cálculo das iterações. Os valores iniciais são utilizados para calcular toda a primeira iteração e, estes, para os cálculos da segunda iteração. Portanto, os valores das iterações são calculados pelos valores da iteração anterior, ou seja:

$$X^{<k+1>} = G(X^{<k>}), \text{ onde } G(X) \text{ é da forma:}$$

$$G(X^{<k+1>}) = \begin{cases} x_1^{<k+1>} = (b_1 - a_{12}x_2^{<k>} - a_{13}x_3^{<k>} - \dots - a_{1n}x_n^{<k>}) / a_{11} \\ \dots & \dots \\ x_n^{<k+1>} = (b_n - a_{n1}x_1^{<k>} - a_{n2}x_2^{<k>} - \dots - a_{nn}x_n^{<k>} - 1) / a_{nn} \end{cases}$$

O método de Gauss-Jacobi também é conhecido como método da substituição simultânea, pois todos os valores são atualizados ou substituídos ao mesmo tempo, no final do cálculo de cada iteração.

Uma desvantagem deste método, é que necessita-se duplicar o vetor solução, um para o valor recente calculado, outro para o valor da iteração anterior. Isto não é só um uso ineficiente do espaço, como também ocasiona uma convergência mais lenta.

#### 4.3.2.2 Método de Gauss-Seidel

O método parte de um valor inicial arbitrário ("um chute inicial"), pois a escolha para o valor inicial desses métodos não é crítica.

Contudo, há a necessidade de se estabelecer um critério de parada adequado. As aproximações vão sendo calculadas, utilizando-se os resultados das iterações anteriores.

Formulação do método

$$x_i^{k+1} = \frac{1}{a_{ii}} (y_i - a_{i1}x_1^{k+1} - a_{i2}x_2^{k+1} \dots - a_{i,i-1}x_{i-1}^{k+1})$$

onde  $x_1, x_2, x_3, \dots, x_n$  são valores conhecidos.

Os índices superiores referem-se ao número da iteração.

Os  $a_{ii}$  não devem ser nulos.

O método de Gauss-Seidel e o método de Jacobi servem como uma medida junto à qual outros métodos iterativos podem ser comparados. Maiores detalhes sobre estes métodos podem ser encontrados em [CAR69, CLA89, VAN78, RAL65].

#### 4.3.2.3 Método da sobre-relaxação

A base para o método da sobre-relaxação é uma técnica que reduz sucessivamente o resíduo. É calculado pelo valor da diferença entre a aproximação e o valor real. O método da sobre-relaxação está baseado na extrapolação linear usando duas aproximações sucessivas. No método da sobre-relaxação os novos valores são calculados por

$$x_i^{(k+1)} = x_i^{(k)} + w(x_i^{r(k+1)} - x_i^{(k)}) \quad (4.6)$$

onde  $x_i^{r(k+1)}$  é o valor calculado pelo método de Gauss-Seidel e  $w$  é o fator ou relaxação, o qual satisfaz :  $0 \leq w \leq 2$ .

Se  $w=1$ , a técnica se reduz ao método de Gauss-Seidel. A escolha do valor de  $w$  influenciará a convergência. Uma alternativa útil para a escolha do valor de  $w$  é arbitrar um valor entre 1 e 2, observando o desempenho da convergência do processo.

Maiores detalhes sobre estes métodos podem ser encontrados em [ALB73, STA67, RAL65].

### 4.3.3 Métodos Compactos

Os métodos compactos são equivalentes ao método de Eliminação de Gauss. Neles, a matriz  $A$  é decomposta em duas matrizes triangulares  $L$  e  $U$ , onde  $L$  é triangular inferior e  $U$  triangular superior. Isto reduz o problema  $AX=B$  a dois sistemas simples  $LY=B$  e  $UX=Y$ .

Maiores detalhes sobre estes métodos podem ser encontrados em [ALB73, CLA89, RAL65].

#### 4.3.3.1 Método de Banachievicz

O método de Banachievicz, também é conhecido como método de decomposição LU, pois a matriz  $A$  é decomposta nas matrizes triangulares  $L$  e  $U$ , tais que  $L * U=A$ , onde:

$$L = \begin{bmatrix} L_{11} & 0 & \dots & 0 \\ L_{21} & L_{22} & \dots & 0 \\ \dots & \dots & \dots & 0 \\ L_{n1} & L_{n2} & \dots & L_{nn} \end{bmatrix} \quad U = \begin{bmatrix} 1 & U_{12} & \dots & U_{1n} \\ 0 & 1 & \dots & U_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.7)$$

A resolução do sistema  $AX=B$ , se resume à resolução dos sistemas:  $LY=B$  e  $UX=Y$ , que podem ser calculados facilmente por retrossubstituição.

#### 4.3.3.2 Método de Cholesky

O método de Cholesky é um método que não requer muita memória para sua implementação e é econômico em termos de tempo de computação. A base da utilização do método é a matriz dos coeficientes  $A$  concatenada com o vetor  $B$ , que forma a matriz estendida  $AB$ , e pode ser reduzida a um sistema onde a matriz seja triangular superior  $U$  da forma:



$$U = \begin{bmatrix} 1 & U_{12} & \dots & U_{1n} & U_{1n+1} \\ 0 & 1 & \dots & U_{2n} & U_{2n+2} \\ 0 & 0 & \dots & U_{3n} & U_{3n+1} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & U_{nn+1} \end{bmatrix} \quad (4.8)$$

Se esta forma pode ser achada, então a solução do sistema pode ser facilmente calculada por retrossubstituição, a partir da última equação.

O método de Choleski para sistemas simétricos é provavelmente o mais eficiente. O pivotamento não é necessário. O único comentário contrário é que a matriz quadrada (n-1) deverá ser computada. Este método requer que todos os elementos da diagonal principal sejam maiores que zero, caso contrário erros podem ser produzidos.

#### 4.3.3.3 Método de Cholesky para sistemas simétricos

Se a matriz A for uma matriz real simétrica, o algoritmo de Cholesky pode ser simplificado, obtendo uma fatoração de A na forma:  $A = L.L'$ , onde  $L'$  é a transposta de L, sendo L da forma:

$$L = \begin{bmatrix} L_{11} & 0 & \dots & 0 \\ L_{21} & L_{22} & \dots & 0 \\ \dots & \dots & \dots & 0 \\ L_{n1} & L_{n2} & \dots & L_{nn} \end{bmatrix} \quad (4.9)$$

Do sistema  $AX=B$ , substituindo A pela decomposição,  $AX = L(L'X) = B$

Foram revistos aqui os métodos pontuais diretos, iterativos e compactos com o objetivo de servir de base e facilitar a compreensão dos métodos intervalares. Algumas vezes, utilizam-se métodos híbridos, onde uma solução aproximada inicial é calculada por métodos pontuais. Esta é, então, melhorada por métodos intervalares.

## 4.4 Métodos Intervalares

A matemática intervalar tem sido utilizada em diversas áreas, para resolver sistemas de equações lineares e não lineares, equações diferenciais ordinárias e parciais, equações integrais e problemas de otimização - como pode ser encontrado em [HAM93]. Para cada uma dessas classes de problemas numéricos, o emprego da matemática intervalar tem sido acompanhado pelo desenvolvimento de novas técnicas, as quais vão além da mera substituição dos coeficientes reais por intervalos e do uso de operações intervalares. A extensão ingênua, por assim dizer, de métodos pontuais em métodos intervalares tem mostrado ser ineficiente, pois não há convergência, como foi observado na extensão ingênua do método de Newton para o cálculo de raízes reais (como demonstrado em [DIV94] e será revisto no item seguinte).

As técnicas para o cálculo numérico com verificação automática do resultado utilizam o produto escalar ótimo e a aritmética intervalar como ferramentas essenciais, além da aritmética de ponto-flutuante simples. A aritmética intervalar permite o cálculo de extremos seguros para as soluções de um problema. Obtém-se alta exatidão por meio do produto escalar ótimo. A combinação destas, proporciona um grande avanço na análise numérica. Uma aplicação ingênua apenas da aritmética intervalar levará a extremos confiáveis. Entretanto, estes podem ser tão grandes que eles são realmente inúteis. Este efeito já era observado há quase vinte anos atrás por analistas numéricos, que muitas vezes criaram rejeições a esta técnica.

Este item tem por objetivo selecionar os métodos intervalares para a resolução de sistemas de equações lineares encontrados no levantamento bibliográfico realizado, identificando características comuns, visando estabelecer parâmetros para a classificação dos métodos intervalares e, se possível, caracterizar técnicas de desenvolvimento comuns nesses métodos. Não é objetivo desta pesquisa a caracterização dos métodos intervalares. Alguns métodos foram descritos como ilustração, visando os testes do uso efetivo da matemática intervalar no supercomputador Cray. O estudo dos métodos, o desenvolvimento de seus algoritmos e a implementação dos mesmos serão desenvolvidos em uma nova etapa de pesquisa (possivelmente como trabalho de pesquisa de aluno de mestrado do PGCC).

### 4.4.1 Resolução Intervalar de equações algébricas

Uma equação algébrica, para fins desta pesquisa, será considerada como um caso particular de sistema de equações, onde a ordem do sistema é um, ou seja, um sistema com uma única equação. A resolução de equações por métodos pontuais pode ser feita por vários tipos de métodos, entre estes se podem destacar os métodos de quebra, os métodos iterativos e os métodos híbridos - como classificados por Diverio em [DIV86].

A resolução de equações por métodos de quebra, como o conhecido método da Bisseção, necessita de um *intervalo* inicial que contenha um zero da equação. Este intervalo é "quebrado" por alguma regra como, por exemplo, pela metade, de forma a

diminuir o *intervalo*, até que se esteja suficientemente próximo do zero da equação. Por fim, tem-se um *intervalo* que contém a raiz cujos extremos diferem apenas nas últimas casas da mantissa. Observa-se que o conceito de *intervalo* é utilizado, mas sem a utilização de propriedades e operações intervalares.

Os métodos híbridos, como o método híbrido C (definido por Claudio, D.M em [CLA89] e, também descrito em [DIV90b]), também utiliza a idéia de *intervalo* que contém uma raiz, o qual é refinado através de operações compostas sobre os extremos que envolvem retas tangentes e secantes interseccionando o eixo  $x$ . Mais uma vez é utilizado o conceito de *intervalo*, sem operações intervalares.

O método iterativo mais conhecido é, sem dúvida, o método de Newton-Raphson, também conhecido como método da tangente, por causa de sua interpretação geométrica visualizar que o valor da nova aproximação  $x_{k+1}$  do zero da equação é tomado como o valor do ponto onde a reta tangente ao ponto  $(x_k, f(x_k))$  intersecciona o eixo  $x$ . Algebricamente o método parte de um valor inicial que, aplicado a uma função de iteração, gera a nova aproximação. Pelo aplicar sucessivo dos valores produzidos pela função de iteração, tem-se uma seqüência de valores que, muitas vezes, é convergente. Quando converge, o limite é uma aproximação de um dos zeros da equação. A função de iteração do método de Newton é dada pela fórmula (4.10), onde se tem que o valor atual é produzido pelo valor anterior menos o quociente do valor da função no ponto anterior pela derivada da função no ponto anterior.

$$x_{k+1} := x_k - f(x_k) / f'(x_k) \quad (4.10)$$

Aparentemente, o método de Newton não se utiliza do conceito de intervalo. Mas, como método iterativo que é, a convergência é determinada pela verificação de certas propriedades em uma vizinhança, a qual pode ser vista como um intervalo. O método de Newton é estendido a raízes múltiplas e complexas e pode ser estendido a cálculos intervalares.

Uma generalização ingênua do método de Newton para o uso da aritmética intervalar seria converter os valores reais por intervalos e as avaliações da função e da derivada por extensões intervalares da mesma. Ou seja,  $X_0 \subseteq X$  e para  $f$  e  $f'$  sejam  $F$  e  $F'$  avaliações intervalares com  $0 \notin F'(X_0)$ . Seja  $\alpha$  um zero de  $F(X)$ ,  $\alpha$  é única no intervalo.

$$X_{k+1} := X_k - F(X_k) / F'(X_k) \quad (4.11)$$

Este método é divergente para todo intervalo inicial  $X_0 \subseteq X$ , com  $\alpha \in X$ ,  $X_0 \neq \alpha$ , pois se for verificado o tamanho do intervalo, pelo diâmetro, nota-se que o intervalo cresce a cada iteração, ou seja  $d(X_{k+1}) \geq d(X_k)$ , logo a seqüência de intervalos  $\{ X_k \}$  é divergente.

Para ilustrar esta passagem, a seguir é considerada uma das versões do método de Newton Intervalar, conhecido como método de Newton Intervalar Simplificado.

Sejam  $X$  e  $X_0$  intervalos, se  $f \in F(X)$  e  $X_0 \subseteq X$ .  $\alpha$  é raiz de  $f$  determinada em  $X$ , e se  $\alpha$  é única;  $M$  é a avaliação intervalar  $F'$  em  $X_0$ . Define-se o operador Newtoniano pela fórmula 4.12, onde  $N_i$  e  $M$  são intervalos.

$$N_i(x) = x - f(x)/M, \text{ onde } 0 \notin M. \quad (4.12)$$

O método de Newton Intervalar simplificado é dado por:

$$X_{k+1} := N_i(X_k) \cap X_k \text{ para } k=1,2,\dots,n \quad (4.13)$$

Observa-se que a propriedade que se  $x_0 \in X_0$  e  $N_i(x_0) \cap X_0 = \emptyset$ , então não existe raiz real de  $f$  em  $X_0$ . Observa-se, ainda, que as variações na definição do operador Newtoniano é que definem as versões do método de Newton intervalar. Maiores informações deste método, bem como de outras versões são encontradas em [DIV91b].

#### 4.4.2 Resolução intervalar de sistemas lineares

Outra aplicação de intervalos é na resolução de sistemas de equações lineares, onde o uso de intervalos pode garantir a existência e unicidade da solução ou identificar se as matrizes envolvidas são ou não singulares.

Os sistemas podem envolver matrizes de coeficientes densas ou esparsas. Observa-se os métodos diretos pontuais para a solução dos sistemas lineares são mais adequados a sistemas cuja matriz de coeficientes é densa, enquanto que sistemas cuja matriz de coeficientes é esparsa, do tipo banda ou tridiagonal, são resolvidos mais eficientemente por métodos iterativos. Existe uma grande variedade de métodos, cujo desempenho é melhor para determinado tipo de sistema, cuja matriz possui determinada propriedade, como por exemplo simétrica e definida positiva. Entretanto, devido à instabilidade (ou estabilidade) do problema, a erros de arredondamentos, principalmente resultantes do produto escalar, presente nas operações de multiplicação de vetores, de matrizes e de matrizes por vetores e ao efeito acumulativo, resultados incorretos ou errôneos são produzidos.

A extensão intervalar destes métodos não é muito simples e o cálculo da solução por métodos intervalares pode ser dispendioso, uma vez que se está tratando com vetores e matrizes de intervalos. Uma alternativa econômica e bastante utilizada é calcular uma aproximação pontual e, a partir desta, melhorá-la através de métodos intervalares.

Entre os métodos intervalares levantados, foram identificadas características que permitiram a reunião dos métodos em três classes ou grupos de métodos. O primeiro grupo consiste em métodos baseados em operações algébricas matriciais intervalares, sendo a versão intervalar de métodos diretos. Neles a solução intervalar é calculada através de propriedades ou de operações intervalares.



O segundo grupo é baseado em refinamentos. São métodos híbridos, onde uma aproximação da solução exata é calculada por algum método pontual. Então a solução é refinada através do uso da matriz inversa real ou intervalar, produzindo aproximações da solução intervalar contendo a solução exata, ou seja inclusões, desta forma se têm limites confiáveis de forma não dispendiosa. O refino da solução pode ser feito através do refinamento do erro de uma aproximação ou do erro da matriz inversa

Por fim, têm-se os métodos intervalares baseados em iterações. São versões intervalares dos métodos pontuais, onde o conceito de iteração também está baseado em inclusões. A convergência destes métodos é definida por teoremas de ponto-fixo vistos no capítulo dois. Os métodos de inclusão podem ser *a priori* ou *a posteriori*, ambos com ou sem inflação. A inclusão pode ser obtida através da intersecção da nova aproximação com a aproximação anterior.

#### 4.4.2.1 Métodos intervalares baseados em operações algébricas

Antes de entrar nos métodos baseados em operações algébricas intervalares, é necessário rever as definições de vetores e matrizes de intervalos e das operações entre estes tipos de dados.

Como foi visto no capítulo 2, um vetor intervalar é um vetor cujos elementos são intervalos. Uma matriz de intervalos é uma matriz cujos elementos são intervalos. Os intervalos podem ser de reais ou complexos, mas para fins desta pesquisa só serão considerados vetores e matrizes de intervalos reais. Se os elementos do vetor ou matriz forem intervalos pontuais (degenerados), tem-se um vetor ou matriz de intervalo pontual. Os diferentes tipos de matrizes também são facilmente estendíveis a matrizes intervalares, como as matrizes triangulares, densas e esparsas.

As operações de adição e subtração de vetores e matrizes de intervalos são definidas como a adição ou subtração intervalar dos componentes correspondentes das parcelas, gerando o elemento correspondente, de mesma posição no vetor ou matriz intervalar resultante.

O produto *escalar* de vetores de intervalos é definido da mesma forma que o produto escalar real. A diferença reside no fato que o resultado não é um real, e sim um intervalo resultante do somatório dos produtos dos elementos intervalares dos vetores.

A extensão do produto de matrizes intervalares mantém a restrição de que o número de colunas da primeira matriz tem de ser igual ao número de linhas da segunda matriz e, a matriz resultante terá a dimensão correspondente ao número de linhas da primeira e o número de colunas da segunda. O produto de duas matrizes intervalares determina que cada elemento da matriz-produto seja calculado pelo somatório dos produtos dos elementos das linhas da primeira pelos elementos da coluna da segunda, onde os elementos são intervalos e as operações são intervalares.

A divisão não era uma operação válida entre matrizes reais, portanto também não é uma operação válida entre matrizes de intervalos. Todas as operações de escalares com vetores e matrizes de reais são estendíveis a vetores e matrizes de intervalos. São ainda definidas as mesmas operações entre intervalos com vetores e matrizes de intervalos. A relação destas operações constam no capítulo cinco, na descrição do módulo *mpi* da biblioteca *libavi.a*.

Os métodos intervalares baseados em propriedades e operações algébricas tendem a ser uma extensão dos métodos diretos pontuais de resolução de sistemas. Eles são estendidos a métodos intervalares pelo uso de operações e argumentos intervalares. Um exemplo é dado a seguir, onde o sistema de ordem dois é mal condicionado. Este exemplo foi resolvido em [DIV94b] e é transcrito a seguir, com o objetivo de ilustrar o uso de intervalos para gerar a solução do sistema.

Exemplo: Considere o seguinte sistema mal-condicionado:

$$\begin{cases} 2.000x_1 + 3.001x_2 = 1.000 \\ 0.6667x_1 + 1.000x_2 = 0.3333 \end{cases}$$

Se os cálculos forem feitos usando aritmética intervalar arredondada com  $n$  dígitos decimais, para  $n=4, 5, 6, 7, 8$  e  $9$ . Pode-se eliminar o termo  $x_1$  na segunda equação para se obter  $(1.000 - (0.6667 / 2.000)(3.001))x_2 = 0.3333 - (0.6667 / 2.000)(1.000)$ . Obtendo-se para  $x_2$  os seguintes resultados:

- para  $n=4$  :  $0.6667/2.000 \in [0.3333, 0.3334]$   
 $(0.6667/2.000)(3.001) \in [1.000, 1.001]$ ,  
 $0.3333 - (0.6667/2.000)(1.000) \in [-0.0010, 0]$ ,  
 $(1.000 - (0.6667/2.000)(3.001)) \in [-0.0010, 0]$ ,  
 $x_2 \in [0, \infty)$  (sem limite em  $x_2$ )
- para  $n=5$  :  $0.667/2.000 \in [0.33335, 0.33335]$ ,  
 $(0.6667/2.000)(3.001) \in [1.0003, 1.0004]$ ,  
 $0.3333 - (0.6667/2.000)(1.000) \in [-0.00005, -0.00005]$ ,  
 $(1.000 - (0.6667/2.000)(3.001)) \in [-0.00040, -0.00030]$ ,  
 $x_2 \in [0.12500, 0.16667]$ ;
- para  $n=6$  :  $x_2 \in [0.128205, 0.131579]$ ;
- para  $n=7$  :  $x_2 \in [0.1302083, 0.1305484]$ ;
- para  $n=8$  :  $x_2 \in [0.13041210, 0.13044613]$ ;
- para  $n=9$  :  $x_2 \in [0.130429111, 0.130429112]$ .

O repentino aumento na exatidão que ocorre entre os  $n=8$  e  $n=9$  é explicado pelo fato de que para  $n>8$  o erro de arredondamento restante é gerado na divisão final por  $x_2$ . Neste exemplo, o tamanho do intervalo calculado diminui na ordem de  $10^{-1}$ , à medida que  $n$  cresce, para  $n>5$ .

A seguir serão apresentados os métodos encontrados no levantamento bibliográfico. Eles não são apresentados de uma maneira formal. São feitos alguns comentários sobre o conteúdo dos artigos levantados.

a) **Método de Hansen**, é um método que foi encontrado em [ALE83, HAN65, HAN67, HAN69, DIV94a, DIV94b] e que se utiliza de propriedades dos intervalos para calcular a solução. Ele estuda o comportamento de inequações em cada quadrante para produzir a solução composta, que pode sofrer o *efeito envolvente*, ou seja, pegar o menor intervalo retangular que contenha a solução.

Considere o sistema de equações  $Ax = b$  onde  $b = (b_i)$  é um vetor real de ordem  $n$  e  $A = (a_{ij})$  é uma matriz real não singular de ordem  $n$ . O vetor solução  $x$  é o vetor  $A^{-1}b$ . Suponha, contudo, que  $A$  e  $b$  são sujeitos a erros. Suponha que se sabe somente que  $a_{ij} \in \alpha'_{ij} = [a_{ij}^1, a_{ij}^2]$  e  $b_i \in b'_i = [b_i^1, b_i^2]$  para  $i, j=1, \dots, n$ . Denote  $A' = (\alpha'_{ij})$  e  $b' = (b'_i)$ , deseja-se saber o conjunto de soluções  $X = \{x: Ax = b, A \in A', b \in b'\}$  para a equação  $A'x = b'$ .

Hansen e Smith<sup>2</sup> discutiram métodos para computar um vetor  $x^l$  contendo  $X$ . Um vetor  $x^l = [x^1, x^2]$  define uma região em um espaço  $n$ -dimensional limitado pelos planos  $x_i = x_i^1$  e  $x_i = x_i^2$  ( $i = 1, \dots, n$ ). Em geral, o conjunto  $X$  não é uma região limitada pelos planos paralelos aos eixos coordenados e então o menor vetor intervalar  $x^l$  não é igual a  $X$ . (Pelo menor vetor intervalar, entende-se o vetor de quem os elementos são todos os menores possíveis).

Contudo o menor  $x^l$  é de interesse. Será mostrado como obter ambos os limites (superiores e inferiores) do menor  $x^l \supset X$ . Assume-se que pelo menos um elemento de  $A'$  ou  $b'$  é um intervalo de tamanho diferente de 0.

Agora consideremos o conjunto  $X$ . Pode-se reescrever  $A'x = b'$  por

$$\sum_{j=1}^n \alpha'_{ij} x_j = b'_i \quad (i = 1, \dots, n). \quad (4.14)$$

---

<sup>2</sup>em [HAN65, HAN67]

Suponha que alguns pontos fixos  $x \in X$  localizam-se no quadrante positivo. Para este  $x$ , (4.6) pode ser escrito como

$$\sum_{j=1}^n a_{ij}^1 x_j, \sum_{j=1}^n a_{ij}^2 x_j = b_i' \quad (i=1, \dots, n) \quad (4.15)$$

Existe  $A \in A'$  e  $b \in b'$  tal que  $Ax=b$ , para este  $x$  fixo, contanto que os intervalos à esquerda e à direita de (4.7) cruzem-se para cada  $i=1, \dots, n$ . Isto é,  $x \in X$  (se  $x_j \geq 0$  para  $j=1, \dots, n$ ) sob condição que:

$$\sum_{j=1}^n a_{ij}^1 x_j \leq b_i^2, \sum_{j=1}^n a_{ij}^2 x_j \geq b_i^1 \quad (i=1, \dots, n). \quad (4.16)$$

Seja  $x_j \leq 0$  para algum valor de  $j$ , então  $a_{ij}' x_j = [a_{ij}^2 x_j, a_{ij}^1 x_j]$ . Considere as equações

$$[2, 3]x_1 + [0, 1]x_2 = [0, 120]$$

$$[1, 2]x_1 + [2, 3]x_2 = [60, 240]$$

No primeiro quadrante, onde  $x_1 \geq 0, x_2 \geq 0$ , pode-se reescrever os membros esquerdos como  $[2x_1, 3x_1 + x_2]$  e  $[x_1 + 2x_2, 2x_1 + 3x_2]$ , respectivamente. As intersecções

$$[2x_1, 3x_1 + x_2] \cap [0, 120]$$

$$[x_1 + 2x_2, 2x_1 + 3x_2] \cap [60, 240]$$

não devem ser vazias e então

$$2x_1 \leq 120 \quad 3x_1 + x_2 \geq 0$$

$$x_1 + 2x_2 \leq 240 \quad 2x_1 + 3x_2 \geq 60$$

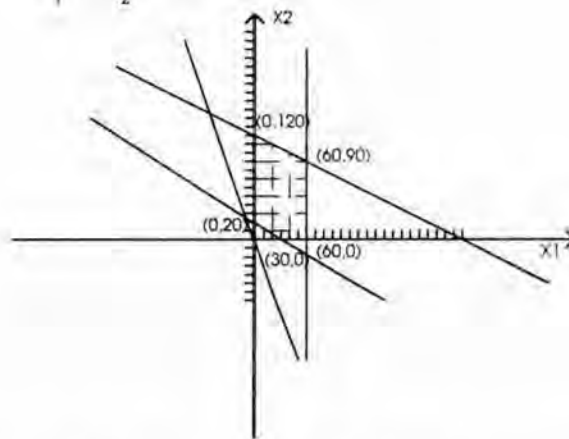


Fig.4.4 Conjunto solução  $X$  no primeiro quadrante



Das inequações pode-se determinar (no primeiro quadrante) que  $X$  é um polígono com vértices nos pontos  $(30,0)$ ,  $(60,0)$ ,  $(60,90)$ ,  $(0,120)$  e  $(0,20)$ .

Repetindo este processo para os outros 3 quadrantes, tem-se: no segundo quadrante, onde  $x_1 \leq 0$  e  $x_2 \geq 0$ ,

$$\begin{aligned} [3, 2]x_1 + [0, 1]x_2 &= [0, 120] \\ [2, 1]x_1 + [2, 3]x_2 &= [60, 240] \end{aligned} \quad e$$

$$\begin{aligned} 3x_1 \leq 120, \quad 2x_1 + x_2 \geq 0 \\ 2x_1 + 2x_2 \leq 240, \quad x_1 + 3x_2 \geq 60 \end{aligned}$$

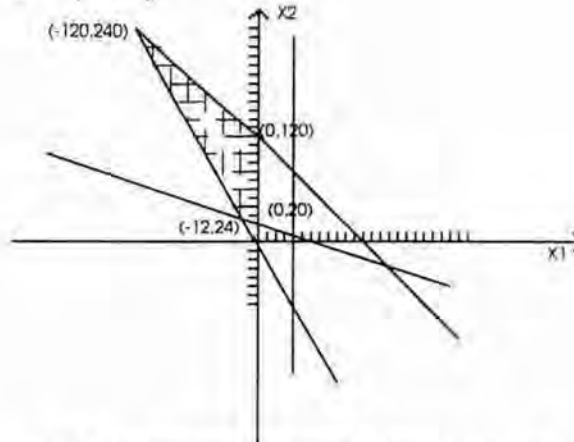


Fig.4.5 Conjunto solução  $X$  no segundo quadrante

No terceiro quadrante, onde  $x_1 \leq 0$  e  $x_2 \leq 0$

$$\begin{aligned} [3, 2]x_1 + [1, 0]x_2 &= [0, 120] \\ [2, 1]x_1 + [3, 2]x_2 &= [60, 240] \end{aligned}$$

$$\begin{aligned} 3x_1 + x_2 \leq 120, \quad 2x_2 \geq 0 \\ 2x_1 + 3x_2 \leq 240, \quad x_1 + 2x_2 \geq 60 \end{aligned}$$

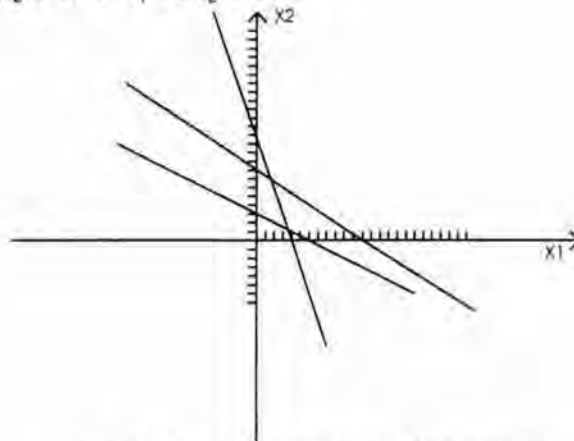


Fig.4.6 Conjunto solução  $X$  no terceiro quadrante

No quarto quadrante, onde  $x_1 \geq 0$  e  $x_2 \leq 0$

$$[2, 3]x_1 + [1, 0]x_2 = [0, 120]$$

$$[1, 2]x_1 + [3, 2]x_2 = [60, 240]$$

$$2x_1 + x_2 \leq 120, \quad 3x_1 \geq 0$$

$$x_1 + 3x_2 \leq 240, \quad 2x_1 + 2x_2 \geq 60$$

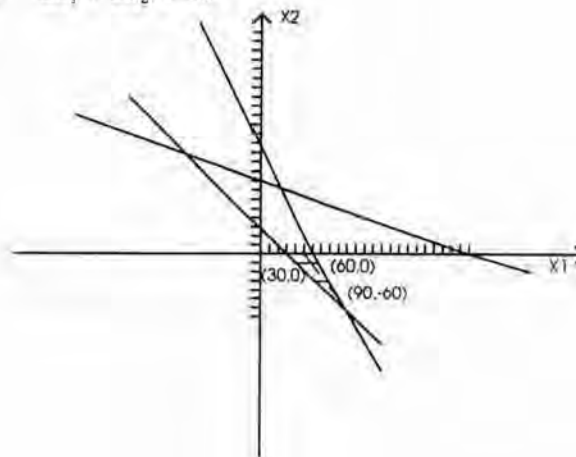


Fig.4.7 Conjunto solução X no quarto quadrante

Mesmo neste caso de duas dimensões, o conjunto X não é particularmente fácil de se representar. Obviamente, em dimensões maiores, X é difícil de representar em geral. Pode-se, contudo, facilmente representar o menor paralelepípedo contendo X tendo lados paralelos aos eixos coordenados. No exemplo acima, este paralelepípedo é limitado pelos planos  $x_1 = -120$ ,  $x_1 = 90$ ,  $x_2 = -60$ ,  $x_2 = 240$ . Pode-se então representá-lo por

$$x^I = \begin{bmatrix} [-120, 90] \\ [-60, 240] \end{bmatrix}$$

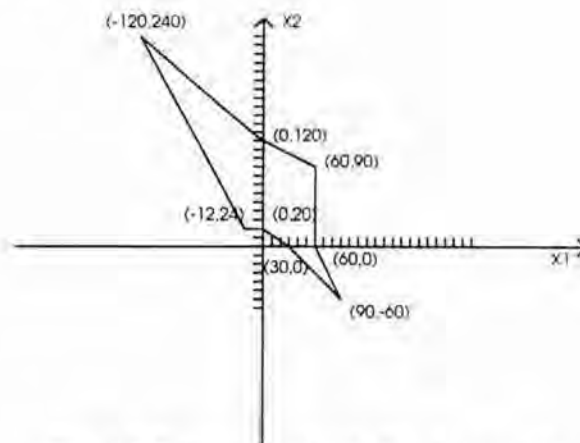


Fig.4.8 Conjunto solução X - área total

Existem várias versões nas quais se buscou melhorar este método, como descrito por Hansen, em [HAN67], onde são apresentadas cinco versões para calcular o vetor intervalar da solução. No entanto, a descrição destas versões ultrapassa o objetivo desta pesquisa.

b) **Método de Eliminação de Gauss para sistemas com coeficientes intervalares** (matrizes e vetores intervalares), estes métodos foram encontrados em [ALE83, NEU90, MAY91]. O método foi descrito pela primeira vez em [ALE83], depois Neumaier (em [NEU90]) acrescentou questões que estavam em aberto. Por fim, em [MAY91], são comentados velhos aspectos e dados novos aspectos sobre o método. A estrutura do método, onde a matriz  $A$  é concatenada com o vetor dos termos independentes para ser triangularizada através de operações elementares é, em linhas gerais, mantida.

A seguir é descrito o Método de Eliminação de Gauss Intervalar. A matriz dos coeficientes e o vetor dos termos independentes são intervalares. Deve-se salientar que, para facilitar a diferenciação dos tipos de dados intervalares dos pontuais, utilizar-se-á um subscrito  $I$  nas matrizes e vetores intervalares, cujo componentes são intervalos. Estes serão representados por letras maiúsculas, matrizes e vetores pontuais são representados por letras em negrito.

Seja  $A^I$  uma matriz intervalar e  $b^I$  um vetor intervalar, assume-se que existe a inversa  $A^{-1}$  para todo  $A \in A^I$ . Deseja-se encontrar o conjunto  $\{X | AX=B, A \in A^I \text{ e } B \in B^I\}$ . Este vetor intervalar solução pode ser determinado pelo algoritmo de Gauss para sistemas de equações lineares com coeficientes intervalares. A estrutura do método de eliminação de Gauss é mantida, ou seja, a matriz  $A^I$  é concatenada ao vetor dos termos independentes  $B^I$  produzindo uma matriz de ordem  $n \times n+1$ , da como em (4.17)

$$\begin{array}{cccc} A_{11} & \cdots & A_{1n} & B_1 \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1} & \cdots & A_{nn} & B_n \end{array} \quad (4.17)$$

Aplicando as fórmulas descritas em (4.18) sobre a matriz descrita em (4.17), assumindo que  $0 \notin A_{11}$ , resulta na nova matriz de coeficientes descrita em (4.19).

$$\begin{array}{ll} A_{1j} = A_{1j}, & 1 \leq j \leq n, \\ B_1 = B_1, \\ A_{ij} = A_{ij} - A_{ij}(A_{11}/A_{11}), & 2 \leq i, j \leq n, \\ B_i = B_i - B_1(A_{i1}/A_{11}), & 2 \leq i \leq n, \\ A_{i1} = 0, & 2 \leq i \leq n \end{array} \quad (4.18)$$

$$\begin{array}{cccc}
 A'_{11} & A'_{12} & \cdots & A'_{1n} & B'_1 \\
 0 & A'_{22} & \cdots & A'_{2n} & B'_2 \\
 \vdots & \vdots & & \vdots & \vdots \\
 0 & A'_{n2} & \cdots & A'_{nn} & B'_n
 \end{array} \quad (4.19)$$

Então se mostra que  $\{X \mid AX=B, A \in A^I \text{ e } B \in B^I\} \subseteq \{Y \mid A'Y=B', A' \in A^I \text{ e } B' \in B^I\}$  é válido. Assume-se  $A \in A^I$  e  $B \in B^I$  e que  $AX=B$ . A matriz  $A'=(A'_{ij})$  e o vetor  $b'=(B'_i)$ , são calculados a partir de (4.20). De forma que os sistemas sejam semelhantes, ou seja, os sistemas  $A'Y=B'$  e  $AX=B$  têm a mesma solução.

$$\begin{array}{ll}
 A'_{ij} = A_{ij}, & 1 \leq j \leq n, \\
 B'_1 = B_1, \\
 A'_{ij} = A_{ij} - A_{ij}(A_{ii}/A_{ii}), & 2 \leq i, j \leq n, \\
 B'_i = B_i - B_i(A_{ii}/A_{ii}), & 2 \leq i \leq n, \\
 A'_{ii} = 0, & 2 \leq i \leq n
 \end{array} \quad (4.20)$$

Se este passo é aplicado  $n-1$  vezes, então a matriz concatenada de ordem  $n \times n+1$  descrita em (4.17) é transformada em uma matriz intervalar triangular superior, como ilustrado em (4.21), cujo sistema linear é semelhante ao inicial e para qual é válida a relação de inclusão:  $\{x \mid Ax = b, A \in A^I, b \in b^I\} \subseteq \{\tilde{x} \mid \tilde{A}\tilde{x} = \tilde{b}, \tilde{A} \in \tilde{A}^I, \tilde{b} \in \tilde{b}^I\}$ .

$$\begin{array}{cccc}
 \tilde{A}_{11} & \tilde{A}_{12} & \cdots & \tilde{A}_{1n} & \tilde{B}_1 \\
 & \tilde{A}_{22} & & & \\
 & & & \vdots & \vdots \\
 & & & \tilde{A}_{nn} & \tilde{B}_n
 \end{array} \quad (4.21)$$

Usando as fórmulas (4.22), que correspondem à retrossubstituição, obtém-se o vetor intervalar  $x^I = (X_i)$  satisfazendo  $\{x \mid Ax = b, A \in A^I, b \in b^I\} \subseteq x^I$ .

$$X_i = \left( \tilde{B}_i - \sum_{j=i+1}^n \tilde{A}_{ij} X_j \right) / \tilde{A}_{ii}, \quad 1 \leq i \leq n-1, \quad X_n = \tilde{B}_n / \tilde{A}_{nn}, \quad (4.22)$$

Se  $A=(a_{ij})$  é uma matriz não singular, então o algoritmo de Gauss é viável para cada vetor intervalar  $b^I$ . Poderá então ser necessário trocar colunas durante o processo de eliminação. Isto é equivalente à multiplicação da matriz  $A$  com uma permutação da matriz no lado esquerdo do processo de eliminação.



Para serem aplicadas, as fórmulas do algoritmo de eliminação Gauss intervalar exigem que o zero não pertença aos elementos intervalares da diagonal principal, ou seja  $0 \notin A_{ii}$ , para todo  $i$ , variando  $1 \leq i \leq n-1$ , em todos os  $n-1$  passos da triangularização. Se para algum elemento intervalar da diagonal principal ( $A_{ii}$ ), o zero estiver contido ( $0 \in A_{ii}$ ), então, analogamente ao método pontual, trocas de linhas ou colunas serão necessários para retirar da diagonal principal o elemento intervalar que contém o zero. Isto é sempre possível se for assumido inicialmente que  $A^{-1}$  exista para todo  $A \in A^I$ . Se não for possível realizar este passo é porque os todos membros da coluna contém zero e, então isto implicará que a matriz original  $A^I$  contém uma matriz singular  $A$ , ou seja, não tem inversa, o que contradiz a suposição inicial.

c) **Cálculo da matriz inversa**, sobre o cálculo da matriz inversa foram encontrados diferentes tópicos. Ele pode ser calculada por métodos pontuais e, então, expandida para intervalos, pode ser calculada por métodos iterativos, por métodos de decomposição LU, onde a matriz  $A$  é igual ao produto de duas matrizes triangulares  $L$  e  $U$ , como visto na parte de métodos pontuais. Então, se  $A=LU$ ,  $R=(LU)^{-1}=U^{-1}L^{-1}=A^{-1}$ . E, ainda sobre o refinamento intervalar da inversa, onde o resíduo é determinado e acrescentado a matriz para calcular a solução. Estes tópicos foram encontrados em [HAM93, ROH88, RUM83, ALE83].

Para calcular a aproximação da matriz inversa de  $A$ , pode ser utilizado o algoritmo com pivotamento parcial. Seja  $A^*=PA$  a matriz que se origina de  $A$  por mudança de linha tal que uma fatorização  $A^*=LU$  com uma matriz triangular inferior  $L$  e uma matriz triangular superior  $U$  existam. Se forem normalizadas as entradas da diagonal de  $L$  para unidade e se forem salvos os índices para as linhas pivotadas em um vetor  $p$ , inicializado por  $p=(1, 2, \dots, n)^T$ , o procedimento da decomposição é o dado por (4.23).

$$\begin{aligned}
 v_k &= a_{ki} - \sum_{j=1}^{i-1} l_{kj}a_{ji}, \quad k = i, \dots, n; \\
 |v_j| &= \max |v_k|, \quad i \leq k \leq n \\
 \text{se } j \neq i : p_i &\leftrightarrow p_j, \quad v_i \leftrightarrow v_j, \quad a_i \leftrightarrow a_j, \quad l_i \leftrightarrow l_j, \quad u_i \leftrightarrow u_j; \quad i=1, \dots, n. \\
 u_{ii} &= v_i \\
 u_{ik} &= a_{ik} - \sum_{j=1}^{i-1} l_{ij}u_{jk} \\
 l_{ki} &= v_k / v_i
 \end{aligned} \tag{4.23}$$

Aqui, a seta dupla ( $\leftrightarrow$ ) indica que os valores dos elementos especificados ou linhas foram trocados. Uma vez que a decomposição LU é avaliada, a inversa aproximada  $R$  é calculada, coluna por coluna, usando substituição simples progressiva ou retroativa. Seja  $e^{(k)}$  o  $k$ -ésimo vetor unitário. Os sistemas  $Ly = Pe^{(k)}$  e  $Ux = y$  serão resolvidos por (4.24) e (4.25), respectivamente. O vetor  $x$  é uma aproximação para  $k$ -ésima coluna de  $R$ .

$$y_i = e_{p_i}^{(k)} - \sum_{j=1}^{i-1} l_{ij}y_j, \quad i=1, \dots, n \tag{4.24}$$

$$x_i = (1/u_{ii}) (y_i - \sum_{j=i+1}^n u_{ij}x_j), \quad i = n, n-1, \dots, 1. \quad (4.25)$$

#### 4.4.2.2 Métodos intervalares baseados em refinamento

Antes de descrever os métodos intervalares deste grupo, será caracterizado o conceito de refinamento. O refinamento é uma técnica que possibilita que se tenha uma medida de exatidão da resposta e outra para se avaliar o sistema quanto ao seu condicionamento. A medida de exatidão da resposta é obtida comparando duas aproximações sucessivas; já a análise do condicionamento é feita através da evolução da medida de exatidão das aproximações sucessivas. Se elas não crescerem, pode ser por causa do mal condicionamento da matriz.

A técnica dos refinamentos compreende dois passos: obtenção de uma primeira aproximação  $x_1$  da solução exata  $x$  e refinamento sucessivo da aproximação da solução  $x_k$ , gerando nova aproximação  $x_{k+1}$ , até que se obtenha  $x$  como limite da seqüência  $(x_k)$ , sob certas condições de convergência. A geração das aproximações é baseada no cálculo do valor do erro contido na solução aproximada. Este erro pode ser somado à aproximação gerando nova aproximação, ou então, ser refinado, uma vez que no cálculo deste também existem erros de arredondamentos embutidos nos cálculos.

Na figura 4.9, item *a*, é ilustrado o vetor  $x$  multiplicado pela matriz  $A$ , gerando o vetor solução  $b$ , ou seja o sistema  $Ax=b$ . A aproximação inicial  $x_1$  do vetor solução  $x$ , contém certo erro devido às operações aritméticas de máquina e aos arredondamentos. A qualidade da aproximação é verificada substituindo-se  $x_1$  no sistema, ou seja, multiplicando a matriz dos coeficientes  $A$  pela aproximação, produzindo a aproximação  $b_1$  do resultado, que é ilustrado pelo item *b* da figura 4.9.

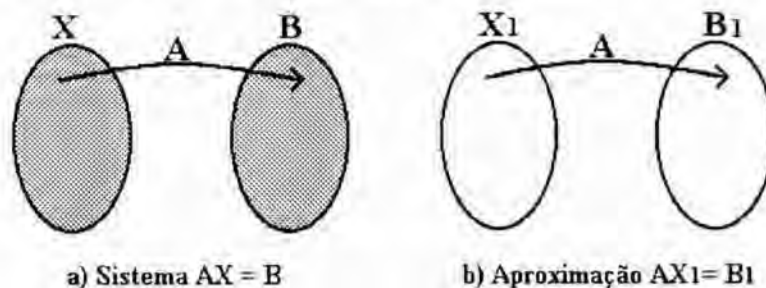


Fig 4.9 Ilustração do sistema e da aproximação

Sobrepondo os dois itens, como na figura 4.10, podem-se visualizar dois erros; o erro no resultado denominado resíduo  $r_1$  e o erro na solução  $z_1$ . Estes erros são importantes pois, se determinados, possibilitam melhorar a solução aproximada e, até mesmo, determinar a solução exata do sistema em alguns casos.

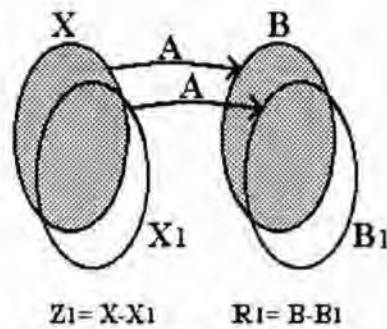


Fig 4.10 Dois erros na solução de um sistema

O erro no resultado ou resíduo  $r_1$ , é facilmente determinado, pois  $r_1 = b - b_1 = b - Ax_1$ . A determinação do erro contido na solução  $z_1$ , que é a diferença entre a solução exata e a aproximação, é mais trabalhosa; pois, como não se tem a solução exata, é necessário resolver o sistema  $Az_1 = r_1$ . Mas na resolução deste sistema erros também são produzidos resultando em aproximações para o erro na solução.

Este erro pode ser somado à aproximação da solução gerando nova aproximação e assim sucessivamente, até se ter uma aproximação suficientemente próxima da solução exata. Isto se constitui em um processo iterativo de refinamento da solução onde cada iteração é chamada de refinamento, gerando uma nova aproximação, a qual poderá convergir ou não para a solução exata. Isto é determinado pela comparação de aproximações sucessivas, como por exemplo pelo número de algarismos significativos corretos<sup>3</sup>.

O erro ao invés de ser somado pode ser refinado, melhorando sua qualidade. Aplica-se o refinamento para determinar o valor mais exato do erro na solução. Isto é muito utilizado nos métodos intervalares.

Na verdade, a técnica de refinamento é uma das bases para o desenvolvimento de métodos intervalares. Neste tipo de métodos intervalares a solução inicial  $x_1$  é determinada por algum método eficiente pontual. A solução é então transformada em um intervalo inflacionado, para então ser refinado. A seguir será considerada, como exemplo, a solução de sistemas densos.

No caso de sistemas densos, tem-se o sistema de equações lineares  $Ax = b$ . A matriz  $A$  pode ser de números reais (**MR**) ou de intervalos reais (**IMR**),  $x$  e  $b$  podem ser vetores reais ou de intervalos reais. Será considerado inicialmente o caso onde a matriz  $A$  é de números reais e os vetores também são de números reais.

Seja  $Ax = b$  um sistema real de equações com  $A \in \mathbb{R}^{n \times n}$  e  $b, x \in \mathbb{R}^n$ . Encontrar a solução do sistema  $Ax = b$  é equivalente a encontrar um zero de  $f(x) = Ax - b$ . Portanto, o método de Newton dá o seguinte esquema de iteração de ponto-fixa

<sup>3</sup>ASC(xk,xk+1)=- (0.3+log(eps+abs((xk+1-xk)/xk+1))) definido em [CLA83]

$$x^{(k+1)} = x^{(k)} - A^{-1} (A x^{(k)} - b), \quad k=0,1,\dots \quad (4.26)$$

Aqui,  $x^{(0)}$  é algum valor inicial arbitrário. Em geral, a inversa de  $A$  não é conhecida. Assim em vez de (4.26), será utilizada a fórmula de iteração do método tipo Newton (4.27), onde  $R \approx A^{-1}$ , é uma aproximação da inversa de  $A$ .

$$x^{(k+1)} = x^{(k)} - R(A x^{(k)} - b), \quad k = 0,1,\dots \quad (4.27)$$

Substituindo as iterações reais  $x^{(k)}$  por vetores intervalares  $[x]^{(k)} \in \mathbb{IR}^n$  e, se existe um índice  $k$  com  $[x]^{(k+1)} \subseteq [x]^{(k)}$ , então, pelo teorema do ponto-fixado de Brouwer, (4.27) tem-se pelo menos um ponto-fixado  $x \in [x]^{(k)}$ . Suponha que  $R$  seja regular. Então este ponto-fixado também é solução de  $Ax=b$ . Portanto, se for considerado o diâmetro de  $[x]^{(k+1)}$ , obtém-se  $d([x]^{(k+1)}) = d([x]^{(k)}) + d(R(A[x]^{(k)} - b)) \geq d([x]^{(k)})$ . Assim, em geral a relação de subconjunto não será satisfeita. Por esta razão, modifica-se o segundo membro de (4.27) para

$$[x]^{(k+1)} = Rb + (I - RA) [x]^{(k)}, \quad k=0,1,\dots, \quad (4.28)$$

onde  $I$  denota a matriz identidade  $n \times n$ . Rump (em [RUM83]) demonstrou que se na equação (4.28) existe um índice  $k$  onde,  $[x]^{(k+1)} \subseteq [x]^{(k)}$ , então as matrizes  $R$  e  $A$  são regulares e existe uma solução única  $x$  do sistema  $Ax=b$  com  $x \in [x]^{(k+1)}$ . E ainda, este resultado é válido para qualquer matriz  $R$ .

Refinamento é um princípio numérico no qual uma solução aproximada  $\bar{x}$  de  $Ax=b$  pode ser melhorada, resolvendo o sistema  $Ay = r1$ , onde  $r1 = b - A\bar{x}$  é o resíduo de  $A\bar{x}$ . Desde  $y = A^{-1}(b - A\bar{x}) = x - \bar{x}$ , a solução exata de  $Ax = b$  é dada por  $x = \bar{x} + y$ .

$$y^{(k+1)} = R(b - A\bar{x}) + (I - RA) y^{(k)}, \quad k = 0,1,\dots \quad (4.29)$$

De acordo com os resultados de Rump, a equação residual  $Ay = r1$  tem uma única solução  $y \in [y]^{(k+1)}$  se for possível encontrar um índice  $k$  satisfazendo  $[y]^{(k+1)} \subseteq [y]^{(k)}$  para o esquema de iteração intervalar correspondente e  $y = x - \bar{x} \in [y]^{(k+1)}$ ; tem-se então, uma inclusão verificada da única solução de  $Ax = b$  dada por  $\bar{x} + [y]^{(k+1)}$ .

Estes resultados continuam válidos se forem substituídas as expressões exatas para  $z := R(b - A\bar{x})$  e  $C := (I - RA)$  em (4.29) por extensões intervalares. Portanto, para evitar efeitos de sobrestimação, é aconselhável avaliar  $b - A\bar{x}$  e  $I - RA$  sem qualquer arredondamento intermediário.

Rump<sup>4</sup> propôs um algoritmo para resolver sistemas lineares em [RUM83]. O algoritmo da figura 4.11, foi implementado em Pascal-SC e mostrou-se eficiente na resolução de sistemas lineares. Este algoritmo é uma adaptação do algoritmo original do

<sup>4</sup>em [KUL83]



Rump, pois em vez de só refinar o resíduo e no final somar com a aproximação inicial como foi proposto por Rump, está sendo refinada a solução passo a passo. Este comentário refere-se ao passo 4 do algoritmo da figura 4.11, onde é acrescido ao resíduo  $z$  o valor da aproximação inicial  $x_0$ . No caso do algoritmo original de Rump, a aproximação inicial é somada ao vetor resíduo  $x$  no final, ou seja no passo 8, em caso de convergência.

Deve-se observar que dado o sistema de ponto-flutuante  $f(b, n, l, u)$ , então,  $eps := b^{(1-n)}$  com  $l$  e  $l+eps$  sendo números consecutivos de  $F$ . Este fator  $eps$  pode ser visto como uma inflação de  $x$ , isto é, ocorrência de erro.

- 1) Calcule aproximação da matriz  $R$  inversa de  $A$  pelo seu método preferido;
- 2) Cálculo do erro existente na aproximação da matriz inversa de  $A$ ,  $E = I - RA$ ;
- 3) Cálculo da aproximação inicial  $x_0$ ,  $x_0 = R.b$ ;
- 4) Cálculo do resíduo,  $z := b - Ax_0$ ,  $z := Rz + x_0$ ;
- 5) Intervalo Inicial,  $X := [z, z]$ ;
- 6)  $k := 0$ ;
- 7) REPEAT
  - $Y := X.[1-eps, 1+eps]$ ; (onde  $eps := 10^{(-10)}$ )
  - $k := k + 1$ ;
  - $X := z + E.Y$ ;
- UNTIL
  - $X \subseteq \overset{\circ}{Y}$  ou  $k = \text{Limit}$  (10 iterações)
- 8) SE  $X \subseteq \overset{\circ}{Y}$ 
  - ENTÃO Existe  $\alpha$  em  $X$ , tal que  $b - A\alpha = 0$ ;
  - SENÃO Não convergiu!  $A$  e  $R$  podem ser singulares.

Fig 4.11 Algoritmo Resolução de Selas intervalares.

Este mesmo algoritmo pode ser modificado para ser aplicado a sistemas de equações lineares, onde a matriz  $A$  é uma matriz intervalar e  $b$  um vetor intervalar. A modificação inclui o cálculo da aproximação da Matriz  $R$ , inversa da matriz  $m(A)$ , que é composta pelos pontos médios de cada elemento intervalar. No passo 2 do algoritmo (figura 4.11) - no cálculo do erro contido na inversa e no passo 4 - onde é calculado o resíduo é utilizada a matriz intervalar  $A$ , o vetor intervalar  $b$  é usado no cálculo da aproximação inicial e no resíduo.

A mesma abordagem pode ser utilizada para sistemas esparsos, onde, por exemplo, a matriz dos coeficientes é do tipo banda (ou tridiagonal). Entretanto, como a inversa de uma matriz banda é, em geral, uma matriz densa, o proposto algoritmo consumiria muito tempo, tornando-se ineficiente. Outra abordagem para o cálculo da matriz inversa  $R$  da matriz  $A$  é a utilização da técnica de decomposição. Decompõe-se a matriz  $A$  em duas matrizes triangulares, uma inferior ( $L$ ) e outra superior ( $U$ ), ou seja,  $A = LU \in MR$ . Tem-se então que a inversa  $R$  é igual a:

$$R = (LU)^{(-1)} = U^{(-1)} \cdot L^{(-1)} = A^{(-1)} \quad (4.30)$$

observa-se que os cálculos são aproximados, não se têm os valores reais, exatos. Têm-se então que  $\mathbf{R} = \mathbf{U}^{(-1)} \cdot \mathbf{L}^{(-1)}$ , que pode ser substituído na condição (4.31).

$$\mathbf{R}(\mathbf{b}-\mathbf{Ax}_0) + \{\mathbf{I}-\mathbf{RA}\} \cdot \mathbf{X} \subseteq \overset{\circ}{\mathbf{X}} \quad (4.31)$$

observa-se ainda, que por serem matrizes triangulares, o sistema pode ser resolvido por retro-substituição, que será denotada pelo operador "\*".

Então a condição (4.31) pode ser reescrita por:

$$\mathbf{U}^{(-1)} * (\mathbf{L}^{(-1)} * (\mathbf{b}-\mathbf{Ax}_0)) + \{\mathbf{I}-\mathbf{U}^{(-1)} * (\mathbf{L}^{(-1)} * \mathbf{A})\} \cdot \mathbf{X} \in \overset{\circ}{\mathbf{X}} \quad (4.32)$$

com esta condição pode-se transformar o algoritmo anterior, substituindo o cálculo da inversa  $\mathbf{R}$  e os produtos da inversa pelo cálculo das matrizes triangulares e pelas retro-substituições. Estas modificações são mostradas e discutidas em [RUM83].

#### 4.4.2.3 Métodos intervalares baseados em iteração

Os métodos intervalares baseados em iterações foram encontrados em: [ALE83], no geral ele descreve os métodos de relaxação; [RUM83], que introduz as fórmulas das versões intervalares dos métodos de Gauss-Seidel, Gauss-Jacobi e o de Sobre-relaxação; [SCH87] que introduz vários procedimentos iterativos; e [NEU90] que trata mais especificamente da versão intervalar do método de Gauss Seidel. Finalmente, em [RUM94] são tratados métodos intervalares para matrizes esparsas e do tipo banda. A seguir são apresentadas as fórmulas dos métodos iterativos.

Seja  $\mathbf{A} \in \mathbf{MR}$ ,  $\mathbf{b} \in \mathbf{VR}$ , O processo de iteração é baseado na fórmula (4.33)

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{B}^{(-1)} (\mathbf{b}-\mathbf{AX}_k) \quad (4.33)$$

para dado vetor inicial  $\mathbf{x}_0 \in \mathbf{VR}$  e uma matriz de iteração  $\mathbf{B} \in \mathbf{MR}$ , então (4.33) é convergente se, e somente se, o raio espectral de  $\mathbf{I} - \mathbf{B}^{(-1)}\mathbf{A}$  for menor do que um.

Seja  $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ , onde  $\mathbf{L}$ ,  $\mathbf{U}$ ,  $\mathbf{D} \in \mathbf{MR}$  são matrizes inferior, superior e diagonal, respectivamente. Então, dependendo da forma como a matriz  $\mathbf{B}$  é definida se têm os métodos de Jacobi (4.34), de Gauss-Seidel (4.35) e o da Sobre-relaxação (4.36).

$$\text{Se } \mathbf{B} = \mathbf{D}, \text{ tem-se de (4.33) que: } \mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{D}^{(-1)}(\mathbf{b}-\mathbf{AX}_k) \quad (4.34)$$

$$\text{Se } \mathbf{B} = (\mathbf{D}+\mathbf{L}), \text{ tem-se de (4.33) que: } \mathbf{X}_{k+1} = \mathbf{X}_k + (\mathbf{D}+\mathbf{L})^{(-1)}(\mathbf{b}-\mathbf{AX}_k) \quad (4.35)$$

$$\text{Se } \mathbf{B} = w^{(-1)}(\mathbf{D}+w\mathbf{L}), \text{ tem-se de (4.33) que: } \mathbf{X}_{k+1} = \mathbf{X}_k + w(\mathbf{D}+w\mathbf{L})^{(-1)}(\mathbf{b}-\mathbf{X}_k) \quad (4.36)$$

Os métodos são bem definidos se os elementos da diagonal principal de  $A$  não são nulos.

Os refinamentos são considerados, por alguns autores, como processos iterativos por causa da geração dos refinamentos sucessivos. Aqui eles foram considerados como uma classe de métodos específica, apesar desta característica.

#### 4.5 Considerações finais

Muitas vezes, há utilização de uma combinação de métodos pontuais e intervalares. É calculada uma aproximação da solução real para, então, utilizar este valor como ponto de partida de métodos intervalares. Com isto ganha-se tempo, pois fazer todos os cálculos com aritmética intervalar pode ser muito dispendioso. Ganha-se, também, qualidade e confiabilidade, pois o resultado está incluído dentro do intervalo solução, portanto se têm limites confiáveis e, pode-se inclusive, dependendo do ambiente onde os cálculos forem feitos, se ter a verificação da corretude, automaticamente pelo computador.

Em geral, segundo Rump, em [RUM83], existe um fator de quatro no custo computacional de algoritmos intervalares comparados em relação a algoritmos de ponto-flutuante. Em comparação, ganha-se a verificação automática da não singularidade da matriz dada e, portanto, a "solubilidade" do sistema é demonstrada pelo algoritmo sem qualquer esforço da parte do usuário e a solução possui a garantia dos limites confiáveis.

Rump também observou que num sistema de ponto-flutuante de base decimal, com 12 casas na mantissa (de precisão) é possível resolver com máxima exatidão sistemas lineares com matrizes de coeficientes com condicionamento de até  $10^8$ . Em [CLA89] era observado que num sistema com condicionamento na ordem de  $10^8$  poderia-se perder até 8 dígitos. Observa-se que com máxima exatidão se tem este ganho. Ele observou, ainda, que sistemas extremamente mal condicionados não são solucionáveis, devido à sensibilidade aos arredondamentos.

Outras considerações sobre os métodos intervalares foram feitas no capítulo sete, onde alguns métodos foram implementados para validar o uso da matemática intervalar no supercomputador Cray.

Um estudo mais específico sobre estes métodos intervalares está sendo desenvolvido, sob orientação de Diverio, T.A por Hölblig, C.A como parte de sua pesquisa de mestrado. Alguns resultados foram publicados no seu trabalho individual (TI-472, [HOL95]).

## 5 PROJETO E ESPECIFICAÇÃO DA BIBLIOTECA INTERVALAR

Neste capítulo é proposta uma aritmética de alto desempenho. Ela consiste em uma aritmética capaz de operar com espaços computacionais avançados, como por exemplo dos intervalos, dos complexos e dos vetores e matrizes complexos ou intervalares produzindo de forma eficiente resultados confiáveis.

O ambiente computacional para onde esta proposta se enquadra é o dos supercomputadores. Na formalização da proposta da aritmética de alto desempenho são revistos tópicos referentes à matemática intervalar de máquina e sobre processamento vetorial, pois estes tópicos são necessários para sua definição e caracterização.

Inicialmente será recordada a pesquisa proposta, a qual abordou a idéia de desenvolvimento de uma biblioteca de rotinas intervalares no ambiente de supercomputadores vetoriais, para se demonstrar o uso efetivo da matemática intervalar neste ambiente. São citados: a justificativa, os objetivos e a metodologia desenvolvida na proposta de Tese.

A seguir é caracterizada a aritmética de alto desempenho. Para tanto, foi desenvolvido um estudo sobre as necessidades, as quais são brevemente relatadas aqui. Como protótipo desta aritmética de alto desempenho, foi desenvolvido um estudo, uma especificação e, posteriormente, implementada uma biblioteca de rotinas intervalares no supercomputador Cray Y-MP2E, denominada *libavi.a*. O nome *libavi.a* significa biblioteca (*lib*) composta da aritmética vetorial intervalar (*avi*). O sufixo *.a* é o sufixo padrão do UNIX, mantido nas bibliotecas no Cray.

Para a especificação desta biblioteca de rotinas intervalares, foi necessário o estudo da aritmética computacional, especialmente do padrão de aritmética binária de ponto-flutuante denominado IEEE 754, o qual estabeleceu as operações aritméticas com máxima exatidão, ou seja, que o resultado das operações aritméticas diferem de no máximo meia unidade do último dígito da mantissa. Para isto, são necessárias diferentes formas de controlar os arredondamentos. Entre estas formas, estão os conhecidos arredondamentos direcionados que gozam de certas propriedades, que possibilitam a definição de operações pela regra geral, a qual garante uma estrutura algébrica mínima no conjunto de ponto-flutuante com estas operações aritméticas.

Como o ambiente vetorial, a aritmética do supercomputador Cray Y-MP2E com a linguagem de programação Fortran 90, não é conforme o padrão da IEEE 754, não só pelo tamanho da palavra, mas também pela forma como os arredondamentos e operações aritméticas em ponto-flutuante são efetuadas, foi necessário desenvolver rotinas que implementassem tais operações.

A seguir, verificaram-se as limitações do processamento vetorial na realização do produto escalar e de somas acumuladas. As diferenças nos resultados foram identificadas como resultantes da forma como os cálculos são efetuados. As diferenças de resultados



foram encontradas não só em relação a máquinas diferentes, mas também entre o processamento seqüencial e vetorial. Para solucionar esta limitação seriam necessário acumuladores longos, de precisão maior do que a disponível no Cray ou, então, a simulação por software da aritmética inteira de precisão infinita. Infelizmente isto ultrapassou os objetivos desta pesquisa, apesar de estabelecer certas limitações à biblioteca de rotinas intervalares quanto à aritmética de alta exatidão e ao desenvolvimento de algoritmos que verificam automaticamente o resultado.

Ainda referente a questões de vetorização e processamento vetorial, que aumenta enormemente a velocidade de processamento das operações, foram pesquisadas as características dos compiladores Fortran 77 e 90, principalmente quanto à vetorização automática. No Fortran 77, havia várias limitações no que diz respeito à definição de novos tipos de dados e de operações que os manipulassem, como por exemplo para a definição de intervalos, que são necessários dois valores para representar o limite inferior e superior do intervalo. Funções em Fortran 77 só permitem um único parâmetro atribuído como resultado. Já em Fortran 90, estas limitações não ocorrem, pois o conceito de módulo já está incorporado à linguagem, o que facilitou o desenvolvimento da biblioteca.

Com a escolha da linguagem Fortran 90, perdeu-se um pouco no potencial de vetorização automática do compilador, pois este ainda está em fase de desenvolvimento pela Cray. Observa-se que o ganho computacional com o uso do Cray está associado à velocidade de processamento, à otimização e vetorização automática do compilador e ao tamanho da memória que permite a solução de problemas de grande porte.

O Fortran 90 possibilitou à biblioteca de rotinas intervalares ter a característica de ser modular. No primeiro módulo estão: a definição de intervalos em ponto-flutuante, as operações aritméticas e entre conjuntos, as funções elementares intervalares e os operadores relacionais entre intervalos. No segundo módulo foram definidos os vetores e matrizes de intervalos, bem como as operações e funções que os manipulam. O terceiro módulo é composto de intervalos de complexos. As operações elementares e os operadores relacionais para complexos também estão disponíveis. Por fim, tem-se o módulo de aplicações, o qual é composto de rotinas que implementam métodos intervalares para a solução de equações algébricas e resolução de sistemas de equações lineares, métodos estes, baseados em refinamentos e em iterações.

Em relação ao tempo de execução, estes métodos podem ser mais demorados do que os métodos reais disponíveis nas bibliotecas matemáticas e científicas da Cray, mas se sabe que os resultados produzidos são confiáveis, pois são verificados automaticamente.

As rotinas escolhidas para compor a biblioteca, foram selecionadas a partir de estudos de outras bibliotecas disponíveis na UFRGS, mas principalmente das rotinas disponíveis na extensão da linguagem Pascal para computação científica (Pascal XSC), a qual tem a si incorporadas todas as características da aritmética de alta exatidão e da verificação automática do resultado.

Alguns dos testes desenvolvidos para avaliar a biblioteca *libavia*, também foram executados em Pascal XSC (em PCs) com o objetivo de se estabelecer comparações em termos da qualidade dos resultados. A avaliação quanto ao tempo de execução foi desenvolvida tomando-se por base os tempos do processamento seqüencial e vetorial.

## 5.1 Formalização da proposta

Nos capítulos anteriores foram expostas algumas dificuldades encontradas ao se resolverem problemas numéricos em computadores. Estas dificuldades se concentram na qualidade do resultado e no porte do problema a ser resolvido. Estudos vêm sendo desenvolvidos para solucionar estas duas barreiras. Estes estudos referem-se à matemática intervalar com a aritmética intervalar e o uso de supercomputadores com processamento vetorial.

A proposta de Tese refere-se, inicialmente, ao desenvolvimento de uma ferramenta denominada aritmética vetorial intervalar *libavia*, na qual está disponível a aritmética intervalar juntamente com o processamento vetorial, ou seja, as operações intervalares podem ser executadas vetorialmente.

A idéia do estudo é mostrada pela figura 5.1, onde, na horizontal, têm-se o avanço da computação e, na vertical, da matemática. O objetivo é se ter diretamente algoritmos numéricos intervalares onde os cálculos são feitos com o auxílio de operadores vetoriais.

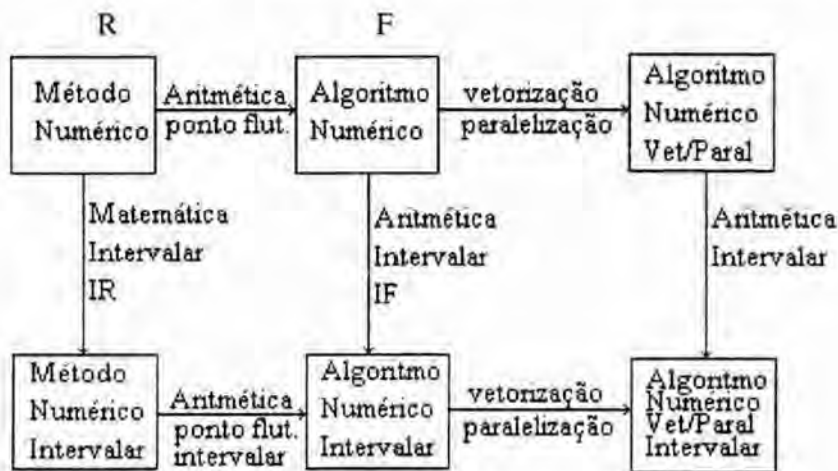


Fig. 5.1 Aritmética Vetorial Intervalar

Os métodos numéricos são projetados nos números reais ( $\mathbb{R}$ ) e, muitas vezes, são fórmulas matemáticas infinitas ou métodos contínuos. Os métodos numéricos intervalares, são baseados na matemática intervalar, onde os intervalos ( $\mathbb{IR}$ ) gozam de propriedades algébricas (como números) e propriedades topológicas (como conjuntos).

A passagem do método numérico real, como do método intervalar, para o algoritmo ou programa em uma máquina digital é feita sobre um sistema de ponto-flutuante ( $F$ ), onde números tem precisão finita, as operações básicas não possuem as mesmas propriedades das operações reais e, portanto, têm-se de trabalhar com aproximações e não com valores exatos.

A passagem de algoritmos numéricos para algoritmos numéricos vetoriais é, muitas vezes, feita automaticamente por compiladores, como o compilador Fortran da Cray, que proporciona a otimização escalar e vetorização automática para vários tipos de ninhos de laços (gerados pelo comando *DO*). O Fortran consagrou-se na vetorização automática devido ao fato de que o comando básico de repetição disponível é o *DO* e a estrutura de dados disponível é o vetor (matriz), que são vetorizáveis. A paralelização não tem o mesmo sucesso devido ao fato de que processar em paralelo não é uma extensão natural do processamento seqüencial, exigindo muitas vezes que se pense em paralelo, que se tenha estruturas de dados que facilitem a paralelização.

A transformação de algoritmos numéricos para algoritmos intervalares deu seu passo fundamental com o desenvolvimento da biblioteca ACRITH. Daí começaram a surgir extensões de linguagem como o Pascal-SC e o Fortran-SC, nas quais estavam disponíveis a aritmética intervalar. Infelizmente estas extensões são bastante limitadas, pois são suportadas em sistemas computacionais de pequeno porte, como computadores pessoais do tipo PC. Devido a isto o porte de problemas matemáticos capazes de serem resolvidos é bastante limitado, como por exemplo, a dimensão máxima de sistemas de equações lineares é de quinze equações com quinze variáveis. Atualmente, existem novas extensões como o Pascal-XSC e C-XSC, as quais não têm estas limitações.

Ainda em relação ao porte de problemas, os problemas que envolvem larga escala de computação, passaram a ser resolvidos com o surgimento dos supercomputadores, os quais não só dispõem de mais memória e de uma grande velocidade de processamento, mas possibilitam o processamento vetorial e paralelo de programas.

Para compensar a perda de exatidão na passagem do real para o sistema de ponto-flutuante, foram desenvolvidos sistemas cada vez mais eficientes, como o padrão da IEEE 754. Foram também aperfeiçoadas bibliotecas de rotinas matemáticas como a NUMERALS, BLAS, LINPACK, IMSL, EISPACK, NAG, NATS, etc, que se consagraram por resolver eficientemente um grande conjunto de problemas. (Detalhes sobre estas bibliotecas podem ser encontradas em [DIV87]) Estas bibliotecas estão disponíveis para uma grande quantidade de sistemas de computadores. Atualmente, elas foram incorporadas a sistemas de supercomputação, resultando em versões otimizadas e vetorizadas, proporcionando uma redução no tempo de processamento.

O grande desafio para se obter algoritmos numéricos vetoriais (paralelos) intervalares reside na concorrência com estes sistemas quanto à qualidade do resultado (exatidão) e quanto ao tempo de processamento.

Desenvolveram-se estudos da viabilidade do uso do supercomputador Cray, disponível no Centro Nacional de Supercomputação (CESUP) na UFRGS (desde maio de 1992), para implementar a aritmética vetorial intervalar para a solução de problemas numéricos. Com esta pesquisa iniciou-se o estudo de metodologias de conversão de: algoritmos seqüenciais para versões vetoriais (e se possível, paralelas); algoritmos numéricos da aritmética real (ponto-flutuante) para a aritmética intervalar. Estes muitas vezes são algoritmos adaptados ou mesmo novos algoritmos que garantem uma seqüência de intervalos convergente.

A aritmética de alto desempenho, implementada na biblioteca de rotinas intervalares (*libavi.a*) é projetada para viabilizar o uso de supercomputadores na solução de problemas físicos, químicos, das engenharias entre outros problemas que necessitem alta exatidão. Para tanto, além da aritmética vetorial intervalar (operações, funções e avaliação de expressões), é necessário providenciar bibliotecas que tornem disponíveis, a estes usuários, os métodos intervalares para solução de seus problemas.

Através destes estudos se concluiu que o avanço prático do uso da matemática intervalar na resolução de problemas reais está, em parte, limitado pela inexistência de ferramentas computacionais que possibilitem um uso efetivo de intervalos. Observou-se que algumas das ferramentas existentes são limitadas, pois o porte dos problemas que se pode resolver é pequeno. Isto dificulta a disseminação prática e o uso de intervalos nas engenharias e em outras áreas.

Para a validação prática do uso da matemática intervalar são necessários: inicialmente, um computador de grande porte, capaz de processar problemas de larga escala; a existência de ferramentas computacionais que tornem a aritmética intervalar disponível em tal máquina; a existência de um conjunto mínimo de rotinas numéricas que viabilizem o desenvolvimento e implementação de algoritmos numéricos intervalares.

A validação então pode ser feita através da resolução prática de um número de problemas em que a aritmética de ponto-flutuante se mostre deficiente, como problemas mal condicionados ou instáveis.

### **5.1.1 Justificativa e Motivação**

A justificativa desta pesquisa pode ser iniciada mencionando-se a grande vantagem que se proporcionará na resolução de problemas da física, química e engenharias, onde os problemas são de grande porte e necessitam rapidez e grande exatidão na solução. A aritmética vetorial intervalar é projetada para isto.



Por outro lado, através desta ferramenta, será possível constatar praticamente a eficiência da matemática intervalar na resolução de problemas reais, através da resolução de sistemas de equações lineares por métodos intervalares em máquinas vetoriais.

Por fim, destaca-se que a existência desta ferramenta computacional, poderá incentivar e permitir o desenvolvimento de estudos de conversão de métodos reais para intervalares em diversas áreas.

Podem-se destacar, ainda, fatores que não são justificativas, mas que tornam viável a pesquisa, como a disponibilidade do supercomputador Cray Y-MP2E no Centro Nacional de Supercomputação na UFRGS, o interesse comum pelo tema dos grupos de Matemática Computacional e Processamento Paralelo do CPGCC da UFRGS e a crescente divulgação e uso de processamento vetorial e paralelo no Brasil.

### 5.1.2 Objetivos

O objetivo geral desta pesquisa foi o estudo da viabilidade prática da matemática intervalar na resolução de problemas, através da resolução de sistemas de equações lineares, para tanto foi necessário o desenvolvimento de uma ferramenta computacional, denominada *libavia*, que reúne tanto a aritmética intervalar quanto o processamento vetorial no Cray. Esta biblioteca de rotinas intervalares contém cerca de 290 rotinas, elas estão organizadas em quatro módulos, cada um tratando com um conjunto de rotinas. Estes conjuntos de rotinas são: rotinas básicas sobre intervalos reais, rotinas que manipulam vetores e matrizes de intervalos reais, rotinas que manipulam intervalos complexos e rotinas intervalares da álgebra.

Entre os objetivos inicialmente específicos podem-se destacar:

O desenvolvimento de uma ferramenta computacional, que tornasse disponível a aritmética intervalar no supercomputador Cray Y-MP2E; e a otimização da biblioteca de rotinas básicas intervalares e de matrizes e vetores intervalares explorando as características vetoriais;

A definição e implementação de uma biblioteca de rotinas numéricas para álgebra linear, que manipulasse vetores e matrizes intervalares e que permitisse a resolução de sistemas de equações lineares; e o estudo de métodos intervalares de resolução de sistemas de equações lineares, desenvolvimento de algoritmos e implementação no supercomputador;

A resolução de exemplos práticos de problemas que envolvam sistemas de equações lineares por métodos numéricos e intervalares, para que fossem feitas comparações quanto a tempo de processamento, exatidão, convergência e controle de instabilidades;

O estudo das limitações, de melhorias e alternativas de implementação das rotinas que compõem os quatro módulos da biblioteca de rotinas intervalares.

Estes objetivos foram desenvolvidos e satisfatoriamente concretizados com a implementação da biblioteca de rotinas intervalares *libavi.a*, em Fortran 90 no supercomputador Cray. Foram, ainda desenvolvidos estudos sobre métodos de resolução de sistemas de equações lineares que possibilitaram a especificação da biblioteca científica intervalar *libselint.a*, abrangendo este capítulo.

### 5.1.3 Metodologia

O desenvolvimento da pesquisa ocorreu em etapas. A **primeira etapa** consistiu na definição e especificação da biblioteca de rotinas intervalares, constituída de quatro módulos; na implementação e documentação do módulo básico, composto por rotinas de transferência, de operações aritméticas intervalares; de operadores relacionais (funções lógicas), de operações entre conjuntos, de funções elementares básicas e rotinas de entrada e saída de dados.

A **segunda etapa** consistiu na implementação e documentação dos módulos *mvi* (vetores e matrizes intervalares), *ci* (intervalos complexos) e *aplic* (rotinas da álgebra linear). Como cada função tem sua extensão intervalar, foi necessário tomar cuidados especiais, para que o domínio de definição das funções fosse observado.

A **terceira etapa** consistiu no estudo dos métodos intervalares para solução de sistemas de equações lineares. Ressalta-se que na notação adotada, o sistema de ordem  $n$  corresponde a uma equação linear. Os objetivos deste estudo foram a identificação de metodologias de conversão de métodos reais em métodos intervalares e a seleção de métodos para a implementação na biblioteca científica intervalar para solução de problemas práticos. Os testes desenvolvidos foram reunidos na biblioteca *libselint.a*.

Na **etapa de resolução de problemas práticos** foram selecionados problemas práticos, os quais foram resolvidos com o auxílio da biblioteca de rotinas intervalares e em Pascal XSC, que possui uma aritmética de alta exatidão. Os resultados foram confrontados visando a verificação da biblioteca. Foram resolvidos alguns exemplos de problemas instáveis, onde a aritmética de ponto-flutuante deixava a desejar em termos de qualidade. Foram escolhidos, também, problemas de maior porte, para que os efeitos da vetorização fossem visualizados e comparados.

Na **etapa da análise das limitações** foram analisadas as limitações da ferramenta computacional e dos efeitos que elas tiveram no resultado dos problemas resolvidos. O resultado desta análise foi empregado para o desenvolvimento das propostas de melhorias e de alternativas de implementação desta biblioteca. Algumas destas propostas começaram a ser desenvolvidas por alunos do mestrado do PGCC.

Finalmente, a **etapa da análise do uso da matemática intervalar** na resolução de problemas práticos, através da biblioteca *libavi.a*, desenvolvida no Cray. Nesta análise, foram identificados os pontos positivos e negativos do uso da aritmética intervalar.

## 5.2 Aritmética de Alto Desempenho

Para que se possa obter uma aritmética de alto desempenho, esta deve ser estendida ainda com outros elementos. Todas as operações com números de ponto-flutuante devem ser supridas de arredondamentos direcionados, isto é, arredondamentos para baixo(para o número de máquina anterior), para cima(para o número de máquina posterior) e simétrico(para o número de máquina mais próximo). Uma aritmética intervalar para números reais e complexos em ponto-flutuante pode ser construída com estas operações.

As operações sobre dois intervalos no computador resultam de operações sobre dois extremos apropriadamente escolhidos dos operandos intervalares, o cálculo do extremo inferior é arredondado para baixo e o cálculo do extremo superior para cima. Dessa forma, o resultado certamente contém todos os resultados da operação aplicada individualmente aos elementos do primeiro e do segundo operandos.

### 5.2.1 A Necessidade

A necessidade de ultrapassar as limitações de espaço e tempo levaram à construção de supercomputadores com características de *hardware* que possibilitam o processamento de alto desempenho (vetorial e paralelo). Com isto, tornou-se possível a solução de problemas que necessitavam de uma grande quantidade de cálculos, conhecidos como *problemas de computação de larga escala*. Houve, portanto, uma preocupação com a *quantidade* de operações que poderiam ser executadas e não com a *qualidade* destas operações em ponto-flutuante. Atualmente, há uma necessidade de maior confiabilidade nos métodos numéricos e dos cálculos. A solução para esta questão seria a obtenção de uma aritmética de alto desempenho, rápida e altamente confiável (com garantia de exatidão).

### 5.2.2 Aritmética Intervalar de Máquina

Na matemática intervalar, em vez de se aproximar um valor real  $x$  para um número de máquina, ele é aproximado por um intervalo  $X$ , que possui, como limite inferior e superior, números de máquina, de forma que o intervalo contenha  $x$ . O tamanho deste intervalo pode ser usado como medida para avaliar a qualidade de aproximação. Os cálculos reais são substituídos por cálculos que usam a aritmética intervalar.

Pode-se justificar o uso de intervalos numéricos pelo fato da aritmética real não ser suficientemente confiável em muitos problemas relacionados à Física e a Química Experimental. Nestas áreas, nem sempre é possível efetuar medições de forma exata, mas sim dentro de uma certa faixa, que pode ser representada através de intervalos.

Para garantir a máxima exatidão em operações intervalares, é necessário definir as operações intervalares sobre o conjunto de intervalos de ponto-flutuante. Estas operações seriam definidas com o auxílio dos arredondamentos monotônicos direcionados como é especificado em [LEY93] e resumido a seguir.

Inverso aditivo:	$-X = [-x_2, -x_1]$
Pseudo inverso multiplicativo:	$1/X = [1/x_2, 1/x_1] \quad 0 \notin X$
Adição:	$X+Y = [x_1 \nabla^+ y_1, x_2 \Delta^+ y_2]$
Subtração	$X-Y = [x_1 \nabla^- y_2, x_2 \Delta^- y_1]$
Multiplicação:	$X*Y = [ \min\{ x_1 \nabla^* y_1, x_1 \nabla^* y_2, x_2 \nabla^* y_1, x_2 \nabla^* y_2 \},$ $\max\{ x_1 \Delta^* y_1, x_1 \Delta^* y_2, x_2 \Delta^* y_1, x_2 \Delta^* y_2 \} ]$
Divisão:	$X/Y = [ \min\{ x_1 \nabla^/ y_1, x_1 \nabla^/ y_2, x_2 \nabla^/ y_1, x_2 \nabla^/ y_2 \},$ $\max\{ x_1 \Delta^/ y_1, x_1 \Delta^/ y_2, x_2 \Delta^/ y_1, x_2 \Delta^/ y_2 \} ]$

Intersecção:	$X \cap Y =$	$\{ [\max\{x_1, y_1\}, \min\{x_2, y_2\}], \text{ se } x_1 < y_2 \text{ e } y_1 \leq x_2$ $\{ \emptyset \quad \text{ caso contrário.}$
União:	$X \cup Y =$	$\{ [\min\{x_1, y_1\}, \max\{x_2, y_2\}], \text{ se } X \cap Y \neq \emptyset$ $\{ \text{erro} \quad \text{ caso contrário.}$
União convexa:	$X \cup Y =$	$\{ [\min\{x_1, y_1\}, \max\{x_2, y_2\}]$

Menor valor absoluto:	$\langle X \rangle = \min \{  x  / x \in X \}$
Maior valor absoluto:	$ X  = \max \{  x_1 ,  x_2  \}$
Valor absoluto:	$\text{Abs}(X) = [ \langle X \rangle,  X  ]$

Ponto médio:	$m(X) = (x_1 + x_2)/2$
Raio do Intervalo:	$r(X) = (x_2 - x_1)/2$

Distância entre dois intervalos:	$q(X, Y) = \max \{  x_1 - y_1 ,  x_2 - y_2  \} \geq 0$
Diâmetro:	$d(X) = x_2 - x_1$
Diâmetro Relativo:	$d_{rel}(X) = d(X) / \langle X \rangle = (x_2 - x_1) / \langle X \rangle$

Relação de igualdade:	$X=Y \Leftrightarrow (x_1=y_1) \text{ e } (x_2=y_2)$
Desigualdade:	$X \neq Y \Leftrightarrow (x_1 \neq y_1) \text{ ou } (x_2 \neq y_2)$

**Demais relações:**

X menor que Y,	$X < Y \Rightarrow x_2 < y_1$
X maior que Y,	$X > Y \Rightarrow x_1 > y_2$
X está contido em Y,	$X \subseteq Y \Rightarrow y_1 \leq x_1 \text{ e } x_2 \leq y_2$
X está contido propriamente em Y,	$X \subset Y \Rightarrow y_1 \leq x_1 \text{ e } x_2 \leq y_2 \text{ e } (x_1 \neq y_1 \text{ ou } x_2 \neq y_2)$
X é interior de Y,	$X \overset{\circ}{\subseteq} Y \Rightarrow y_1 < x_1 \text{ e } x_2 < y_2$
X contém Y,	$X \supseteq Y \Rightarrow x_1 \leq y_1 \text{ e } y_2 \leq x_2$
X contém propriamente Y,	$X \supset Y \Rightarrow x_1 \leq y_1 \text{ e } y_2 \leq x_2 \text{ e } (x_1 \neq y_1 \text{ ou } x_2 \neq y_2)$
x pertence a X,	$x \in X \Rightarrow x_1 \leq x \leq x_2$
X intersecção Y é vazia	$X \cap Y = \emptyset \Rightarrow x_2 < y_1 \text{ ou } y_2 < x_1$



A principal vantagem da utilização da aritmética intervalar de máquina é a determinação de um conjunto que contém a solução do problema que se pretende resolver. Demonstra-se que a estrutura de  $(\mathbf{IF}, +, *)$ , onde  $+$  e  $*$  são definidas com arredondamentos é um **anelóide**<sup>1</sup> com elementos neutros  $[0,0]$  e  $[e,e]$ .

A adição e multiplicação são associativas em  $\mathbf{IR}$ , entretanto a associatividade destas operações não são válidas em  $\mathbf{IF}$ , como é ilustrado em [LEY93].

Através de estudos pode ser observado que no conjunto dos intervalos de número de ponto-flutuante se perdem várias propriedades das operações aritméticas, como a lei associativa para operações de adição e multiplicação, a perda do elemento simétrico para adição, a perda do elemento inverso da multiplicação, a perda da unicidade do elemento neutro para adição e a perda da distributividade de multiplicação em relação à adição.

Um exemplo simples mas significativo da influência da falta destas propriedades na resolução de problemas pode ser dado com o problema de resolução algébrica de uma equação do primeiro grau, do tipo:  $ax + b = 0$ . A solução real  $x = -b/a$  só pode ser obtida nos reais, porque os reais possuem a estrutura de anel, onde existem os elementos inversos da adição e multiplicação. O significado de "passar  $b$  para o outro lado" significa que se está adicionando nos dois lados da igualdade o simétrico de  $b$ , no caso  $-b$ . O mesmo ocorre em relação à multiplicação, dividem-se ambos os lados por  $a$ , mas se não são verificadas as propriedades da existência dos inversos no conjunto dos intervalos de ponto-flutuante, qualquer método algébrico baseado em tais propriedade produzirá resultados errados como a versão intervalar "ingênua" do método de Newton.

### 5.2.3 Características do Processamento Vetorial

Processamento vetorial ou vetorização tem diferentes significados para diferentes tipos de pessoas. Para pessoas que desenvolvem linguagens, por exemplo, vetorização é utilizada para caracterizar linguagens do tipo array, como a extensão do Fortran *vecran*. Para pessoas que desenvolvem compiladores, significa análise de dependências de comandos do código fonte (*do-loop*) e conversões de operações seqüenciais para operações vetoriais equivalentes. Para usuários, vetorização significa a introdução de instruções vetoriais de *hardware* em seus programas, para que a alta velocidade destas instruções possa ser utilizada de forma a melhorar o desempenho de seus programas.

Adotando o ponto de vista do usuário, a seguir são mostradas as três formas de se utilizar a vetorização e uma caracterização mais formal das instruções vetoriais.

O objetivo primário da vetorização é para achar operações seqüenciais que podem ser convertidas em operações vetoriais semanticamente equivalentes, fazendo uso de vantagens do *hardware* vetorial. Em geral, um comando pode ser vetorizado se ele não

---

<sup>1</sup>Anelóide estrutura algébrica definida em [CLA79]

requer entrada na iteração de uma variável do *loop*, a qual é computada anteriormente no *loop* (dependência verdadeira).

O resultado final da vetorização de um *loop* é produzir instruções de máquina que executem grupos de elementos de dados em registradores vetoriais. Na forma simples, um *do-loop* é transformado de uma iteração sobre elementos simples para uma iteração sobre grupos ou seções de elementos.

A primeira forma de se obter a vetorização é simplesmente recompilar os códigos escalares em Fortran, utilizando o compilador que vetoriza automaticamente os códigos. Compiladores Fortran deste tipo estão disponíveis nos supercomputadores como Cray Y MP, IBM 3090VF e Convex C210, existentes no Brasil.

Outra forma de vetorizar é reestruturar o código fonte para auxiliar o compilador a reconhecer mais oportunidades para gerar códigos vetoriais, obtendo assim mais benefícios, um maior grau de vetorização das rotinas.

Por fim, para vetorizar pode-se recodificar completamente a aplicação pela escolha ou pelo desenvolvimento de um novo algoritmo que produza maior benefício das características vetoriais da máquina.

Um vetor no Cray Y-MP2E é um conjunto ordenado de elementos, cada elemento é representado como uma palavra de 64 bits. A distinção de vetor e escalar é que escalar é um único elemento, enquanto que vetores são vários elementos. Exemplos de estruturas em Fortran que podem ser representados por vetores são *arrays* de uma dimensão, bem como linhas, colunas e diagonais de *arrays* multidimensionais. O processamento vetorial ocorre quando operações aritméticas e lógicas são aplicadas sobre vetores. A diferença do processamento escalar é que processa vários elementos no lugar de apenas um. Em geral processamento vetorial é mais rápido e mais eficiente do que processamento escalar.

Entre as características do processamento vetorial, está o operando vetorial, que contém um conjunto ordenado de N elementos, onde N é chamado do tamanho do vetor. Cada elemento em um vetor é uma quantidade escalar, a qual pode ser um número em ponto-flutuante, um inteiro, um valor lógico ou um caracter (*byte*). As instruções vetoriais podem ser classificadas em seis tipos de primitivas: **f0**:  $S \Rightarrow V$ ; **f1**:  $V \Rightarrow V$ ; **f2**:  $V \Rightarrow S$ ; **f3**:  $V \times V \Rightarrow V$ ; **f4**:  $V \times S \Rightarrow V$ ; **f5**:  $S \times S \Rightarrow V$ ; onde V e S denotam um operando vetorial e um operando escalar, respectivamente. As funções f1 e f2 são operações unárias, enquanto que f3 e f4 são binárias.

O processamento vetorial elimina a maioria dos testes e controle de operações, principalmente as associadas a *loops*. Ele minimiza o tempo dispendido na espera da carga dos operandos e de armazenagem do resultado, pela referência a grupos de locação de memória e pelo uso de registradores de múltiplos elementos.

No processamento vetorial, sucessivos elementos são supridos a cada período do relógio (um CP - *clock period*), e também, são concluídas operações, sendo os resultados enviados a elementos sucessivos do vetor resultante. As operações vetoriais continuam sendo executadas até que o número de operações efetuadas seja igual à quantidade especificada no registrador de tamanho denominado VL (*vector length*).

O processamento vetorial reduz o controle associado à manutenção de variáveis de controle de *loops* (incrementar e verificar com o contador) e muitas vezes são reduzidos problemas de conflito de acesso à memória.

Outra vantagem do processamento vetorial é que ele reduz o número de instruções em linguagem de máquina que precisam ser carregadas, decodificadas e executadas.

### 5.3 Biblioteca de rotinas intervalares *libavi.a*

A biblioteca de rotinas intervalares *libavi.a* foi projetada para viabilizar o uso da matemática intervalar em supercomputadores para a solução de problemas físicos, químicos e das engenharias que necessitem alta exatidão. Para tanto, além da aritmética vetorial intervalar (operações, funções e avaliação de expressões) foi necessário providenciar bibliotecas que tornassem disponíveis a estes usuários os métodos intervalares para solução de seus problemas. Inicialmente, foi especificada a biblioteca científica aplicativa *libselint.a*, composta por algumas rotinas intervalares de resolução de equações algébricas e sistemas de equações lineares.

A *libavi.a* foi desenvolvida em Fortran 90 da Cray. Ela totaliza cerca de 290 rotinas que manipulam intervalos reais, vetores e matrizes de intervalos reais, intervalos complexos e algumas rotinas intervalares estendidas da álgebra linear que manipulam matrizes e vetores, incluídas na biblioteca BLAS. Ela pode ser descrita como tendo quatro módulos. Os três primeiros módulos são os módulos que implementam os tipos de dados intervalo real, vetores e matrizes de intervalos e intervalos complexos. Estes módulos foram denominados de *básico*, *mvi* e *ci* respectivamente. Nos módulos *básico* e *ci* existem conjuntos de rotinas ou funções agrupados de acordo com a natureza de operações. Estes conjuntos são seis. O primeiro conjunto são as funções de transferência, as quais tratam de converter reais ou complexos para intervalos, vice-versa e, ainda funções que calculam particularidades geométricas dos intervalos, como diâmetro, raio, ponto médio e distância entre intervalos.

O segundo tipo são as funções relacionais, cujo tipo de dado resultante é geralmente lógico (booleano). Entre as funções relacionais estão a igualdade, a diferença, ser menor, maior, estar contido, conter, ser interior e a relação de pertinência, ou seja um real pertencer ao intervalo. Mais detalhes podem ser vistos no capítulo 2, onde há uma revisão sobre intervalos.

O terceiro conjunto de rotinas trata de operações entre conjuntos, entre estas operações estão a intersecção, a união, a união convexa. O quarto conjunto de rotinas

implementa a aritmética, ou seja as operações aritméticas de adição, subtração, multiplicação e divisão. São incluídas algumas operações aritméticas entre diferentes tipos de dados.

As funções elementares, como valor absoluto, raiz quadrada,  $x$  elevado ao quadrado, potenciação, exponencial, logaritmo e as trigonométricas constituem o quinto conjunto de rotinas. Por fim, estão as rotinas de entrada e saída, *read* e *write* para os novos tipos de dados.

O módulo de matrizes e vetores intervalares *mvi* foi organizado em três partes, a dos vetores, a das matrizes de intervalos reais e a parte que contém operações aritméticas de diferentes tipos de dados com intervalos, vetores e matrizes de intervalos reais. Elas contêm diferentes grupos de rotinas. Estes grupos são: Funções de transferência, Operações relacionais e entre conjuntos, Operações aritméticas, Transposição, Funções básicas e Rotinas de entrada e saída. O último módulo é o das aplicações, denominado *aplic*. Nesta parte, existem algumas rotinas que implementam operações aritméticas compostas existentes em outras bibliotecas. Para a escolha destas rotinas, foram analisadas algumas bibliotecas como a *NUMERALS* da Burroughs, as BLAS e o próprio Pascal XSC.

Utilizou-se o conceito da interface para garantir a sobrecarga das operações aritméticas, dos operadores relacionais e das funções elementares básicas. Desta forma o programador pode utilizar a notação matemática (e também disponível em Fortran) para as operações aritméticas, funções elementares e para os testes. O compilador, ao encontrar um destes operadores, verifica o tipo de dado dos argumentos e seleciona automaticamente a rotina correta que os manipula.

Organizou-se outro arquivo, denominado *libselint.a*, no qual foram agrupadas rotinas que tornam disponíveis métodos intervalares de resolução de equações algébricas e de resolução de sistemas de equações lineares. Foram selecionados apenas alguns métodos, para ilustrar o uso efetivo da matemática intervalar. Pretende-se que este arquivo venha a se tornar uma biblioteca científica intervalar.

A seguir são descritos os módulos da biblioteca de rotinas intervalares em detalhes e a hierarquia destes módulos.

### 5.3.1 Módulo de intervalos reais - *básico*

As rotinas que compõem o módulo *básico* podem ser divididas em seis grupos: rotinas de transferência; operações relacionais; operações entre conjuntos; operações aritméticas; funções elementares intervalares e rotinas de entrada e saída de dados intervalares. Todas as rotinas têm seu nome precedido pela letra *s*, esta padronização foi feita para indicar o tipo de dado com que as rotinas operam, no caso intervalo (ou segmento).



O tipo de dado intervalo está definido no arquivo *inter.inc* para facilitar o uso, pois todos os módulos que utilizam intervalos e todas as rotinas destes módulos que operam com intervalos precisam da definição do tipo intervalo inseridas no seu corpo. Na forma de arquivo, a definição é incluída através do uso do comando *include 'inter.inc'*. A definição de intervalos contida no arquivo *inter.inc* é dada pela figura 5.2.

```

type interval
  sequence
  real :: inf, sup
end type interval

```

Fig 5.2 Arquivo *inter.inc*

- **Funções de transferência** - Neste grupo estão as rotinas que inicializam intervalos e que fornecem os extremos dos intervalos. Estão ainda entre estas rotinas as que calculam particularidades geométricas. A tabela 5.1 fornece as rotinas implementadas deste grupo, os parâmetros de entrada e de saída (resultado) e a descrição.

Tabela 5.1 Funções de Transferência de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Carga de intervalo degenerado ( $A=[r1,r1]$ )	sintpt	r1: real	a: interval
Carga de intervalo ( $A=[r1,r2]$ )	sintval	r1,r2: real	a: interval
Busca do limite inferior - ínfimo	sinf	a: interval	r1: real
Busca do limite superior - supremo	ssup	a: interval	r2: real
Troca de dois intervalos reais $A \leftrightarrow B$	sswap	a,b: interval	a,b: interval
Copia intervalo real $B=A$	scopy	a: interval	b: interval
Inflação do intervalo real $B=[r1-\epsilon, r2+\epsilon]$	sblow	a: interval, $\epsilon$ : real	b: interval
Ponto Médio ( $m(A)$ )	smed	a: interval	r: real
Menor Valor Absoluto ( $\langle A \rangle$ )	smin	a: interval	r: real
Maior Valor Absoluto ( $ A $ )	smax	a: interval	r: real
Módulo do intervalo ( $ A $ )	smod	a: interval	r: real
Distância entre dois intervalos ( $q(A,B)$ )	sdis	a, b: interval	r: real
Diâmetro de um intervalo ( $d(A) \geq 0$ )	sdia	a: interval	r: real
Diâmetro relativo de um intervalo ( $drel(A)$ )	sdiar	a: interval	r: real
Raio de um intervalo ( $r(A)$ )	sraio	a: interval	r: real

- **Operações relacionais** - São rotinas que relacionam intervalos reais, fazendo comparações entre eles. As rotinas deste grupo são enumeradas pela tabela 5.2, cuja descrição contém a relação e sua notação matemática. Os operadores relacionais são acessados pelo nome da rotina ou pelo operador relacional definido em Fortran para reais, pois o compilador verifica o tipo de dado e carrega a rotina correta. O resultado é sempre um tipo lógico.

Tabela 5.2 Operadores relacionais entre intervalos reais

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Igualdade entre dois intervalos ( $A=B$ )	sequ	a,b: interval	l: logical
A menor que B? ( $A<B$ )	sless	a,b: interval	l: logical
A maior que B? ( $A>B$ )	sgre	a,b: interval	l: logical
A está contido em B? ( $A\subseteq B$ )	sleq	a,b: interval	l: logical
A contém B? ( $A\supseteq B$ )	sgeq	a,b: interval	l: logical
r pertence a A? ( $r \in A$ )	rins	r: real, a: interval	l: logical
A não é igual a B? ( $A \neq B$ )	sneq	a,b: interval	l: logical
A é interior de B? ( $A \overset{\circ}{\subseteq} B$ )	sxiny	a,b: interval	l: logical
A está contido propriamente em B ( $A \subset B$ )	sxcy	a,b: interval	l: logical
A contém propriamente B ( $A \supset B$ )	sxmy	a,b: interval	l: logical
A intersecção B é vazia? ( $A \cap B = \emptyset$ )	siv	a,b: interval	l: logical

- **Operações entre conjuntos** - As operações sobre conjuntos foram estendidas a intervalos reais. Estas operações são a união, a união convexa cujo resultado é o intervalo maximal e a intersecção. Na tabela 5.3 estas operações estão relacionadas com seus tipos de dados de parâmetros e resultante.

Tabela 5.3 Operações entre conjuntos reais

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
União de Intervalos ( $A \cup B$ )	sunl	a,b: interval	c: interval
União convexa de intervalos ( $A \cup_c B$ )	sunid	a,b: interval	c: interval
Intersecção de Intervalos ( $A \cap B$ )	sinter	a,b: interval	c: interval

- **Operações aritméticas** - Este grupo de operações possui os extremos inflacionados após a realização do cálculo, através das rotinas *infla\_down* e *infla\_up* que emulam os arredondamentos direcionados.

Tabela 5.4 Operações aritméticas entre intervalos reais

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Soma de Intervalos ( $c=a+b$ )	sadd	a,b: interval	c: interval
Soma unária ( $c=+a$ )	smt	a: interval	c: interval
Subtração de Intervalos ( $c=a-b$ )	ssub	a,b: interval	c: interval
Negação do intervalo ( $c=-a$ )	sneg	a: interval	c: interval
Multiplicação de intervalos ( $c=a*b$ )	smult	a,b: interval	c: interval
Divisão de Intervalos ( $c=a/b$ )	sdiv	a,b: interval	c: interval
Inversão de Intervalo ( $c=1/a$ )	sinv	a: interval	c: interval

- **Funções básicas** - Neste grupo estão as funções elementares algébricas, logarítmicas, exponenciais e trigonométricas. O domínio de definição das funções intervalares é dado pela tabela 5.5. Podem-se utilizar as funções pelo nome que consta na tabela 5.5 ou pelo nome real da função, pois dependendo do tipo de dado, o compilador carregará a função correta. Os valores das constantes que definem os domínios são dados nos anexos.

Tabela 5.5 Funções Elementares Intervalares

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	DOMÍNIO	RESULTADO
Valor Absoluto	sabs	a: interval	$[-\text{maxreal}, \text{maxreal}]$	c: interval
X ao quadrado	ssqr	a: interval	$[-\text{sqrmax}, +\text{sqrmax}]$	c: interval
Raiz quadrada	ssqrt	a: interval	$[0, \text{maxreal}]$	c: interval
Função exponencial	sexp	a: interval	$[-\text{maxreal}, \text{expmax}]$	c: interval
Função logarítmica neperiana	sln	a: interval	$[\text{minreal}, \text{maxreal}]$	c: interval
Função logarítmica decimal	slog	a: interval	$[\text{minreal}, \text{maxreal}]$	c: interval
Potenciação	spotn	a: interval, b: real	$a \in [-\text{maxreal}, \text{maxreal}]$ $b \in [0, \text{sln}(\text{maxreal})]$	c: interval
Arcotangente	sarctan	a: interval	$[-\text{maxreal}, \text{maxreal}]$	c: interval
Seno	ssin	a: interval	$[-\text{maxreal}, \text{maxreal}]$	c: interval
Cosseno	scos	a: interval	$[-\text{maxreal}, \text{maxreal}]$	c: interval
Tangente	stan	a: interval	$[-\text{maxreal}, \text{maxreal}]$ $e \pi/2 + k.\pi \notin a$	c: interval

- **Rotinas de entrada e saída** - Neste grupo estão as rotinas de leitura e impressão de intervalos reais, elas são denominadas de *sread* e *swrite*. Os dados são impressos em formato livre de reais, sendo que a cada linha é impresso um único intervalo. A leitura de intervalos é feita de arquivos em formato livre, como é detalhado em 5.3.5.

### 5.3.2 Módulo de vetores e matrizes de intervalos reais - *mvi*

O módulo que trata com matrizes e vetores de intervalos reais é o *mvi*. Este módulo foi subdividido em três partes: operações com vetores de intervalos, operações com matrizes de intervalos e a parte que implementa as operações aritméticas de diferentes tipos de dados com intervalos, vetores e matrizes de intervalos reais. As operações compostas envolvendo vetores e matrizes intervalares foram incluídas no módulo *aplic*.

Este módulo inclui e utiliza em todas as suas rotinas o módulo *básico*, onde foram definidas as operações e funções sobre intervalos. As operações sobre vetores e matrizes de intervalos são extensões naturais das operações intervalares, componente a componente. Os domínios e restrições das funções sobre vetores e matrizes são os mesmos das funções sobre intervalos, acrescidos da exigência que sejam válidos em cada componente do vetor ou matriz de intervalo.

Os tipos de dados vetor e matriz de intervalos são denominados por *vvector* e *smatrix* para fins de documentação e facilidade de referência nas tabelas de rotinas. Na verdade, não são definidos formalmente estes tipos de dados como no caso de intervalos reais. A especificação de vetores e matrizes de intervalos é feita pelo comando *type*, como foi ilustrada na figura 3.6, ou seja: *type (interval), dimension(:) :: vector\_interval* e *type (interval), dimension(:,:) :: matrix\_interval*. Nas definições são utilizados *arrays* dinâmicos, que possibilitam que os vetores e matrizes tenham um tamanho genérico. Na definição de vetor, o comando *dimension* possui uma dimensão indicada por dois pontos, já na definição de matriz existem duas vezes o sinal de dois pontos, indicando que a matriz é bidimensional.

A dimensão do vetor ou matriz é especificada no programa do usuário, o qual também deve conter uma definição de vetor e/ou matriz de intervalos compatível com o número de elementos definidos para cada dimensão. Caso as dimensões dos vetores ou matrizes de intervalos não sejam compatíveis com a operação desejada, esta não será executada e retornará uma mensagem de erro acusando dimensões inválidas.

As operações entre vetores e matrizes intervalares também estão agrupadas em seis conjuntos, de acordo com a natureza da operação; são elas: funções de transferência, operações relacionais e entre conjuntos, transposição, operações aritméticas, funções básicas e rotinas de entrada e saída de dados. As operações são extensões naturais das operações entre intervalos.



### 5.3.2.1 Operações com vetores de intervalos reais

As operações entre vetores de intervalos são uma extensão natural das operações entre intervalos, aplicada em cada componente do vetor. A operação só é validada se estiver definida para cada elemento do vetor de intervalos. Para fins de documentação utilizou-se *vector* para indicar vetor real e *svector* para vetores de intervalos. Todas as rotinas vetoriais intervalares iniciam pelas letras *sv*. As funções que calculam características geométricas de vetores de intervalos, as rotinas aritméticas entre vetores, os operadores relacionais e as funções básicas vetoriais podem ser acessadas pelo nome da rotina ou pelo nome padrão de funções reais do Fortran.

- **Funções de transferência** entre vetores reais e vetores de intervalos seguem o padrão das funções de transferência de reais com intervalos reais, sendo estas aplicadas em cada elemento do vetor. Estão também presentes neste grupo as funções que calculam características geométricas de vetores de intervalos, que são descritas pela tabela 5.6.

Tabela 5.6 Funções de Transferência de Vetores de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Carga de vetor intervalar degenerado ( $V=[v1.v1]$ )	svintpt	v1: vector	v: svector
Carga de vetor intervalar ( $V=[v1.v2]$ )	svintval	v1, v2: vector	v: svector
Busca do limite inferior - infimo	svinf	v: svector	v1: vector
Busca do limite superior - supremo	svsup	v: svector	v2: vector
Troca dois vetores intervalares entre si	svswap	v,w: svector	v,w: svector
Copia de um vetor intervalar	svcopy	v: svector	w: svector
Inflaciona um vetor intervalar	svblow	v: svector,r:real	w: svector
Ponto Médio de vetor intervalar ( $m(V)$ )	svmed	v: svector	v1: vector
Menor Valor Absoluto ( $\langle V \rangle$ )	svmin	v: svector	v1: vector
Maior Valor Absoluto ( $ V $ )	svmax	v: svector	v1: vector
Módulo do vetor de intervalo ( $ V $ )	svmod	v: svector	v1: vector
Distância entre dois vetores de intervalos ( $q(V,W)$ )	svdis	v, w: svector	v1: vector
Diâmetro de um vetor de intervalo ( $d(V) \geq 0$ )	svdia	v: svector	v1: vector
Diâmetro relativo de um vetor de intervalo ( $drel(V)$ )	svdiar	v: svector	v1: vector
Raio de um vetor de intervalo ( $r(V)$ )	svraio	v: svector	v1: vector

- **Operações relacionais e entre conjuntos** - neste grupo estão os operadores relacionais e as rotinas que manipulam conjuntos de vetores intervalares. Elas só são válidas se forem definidas em cada componente do vetor. A relação dos operadores e rotinas são dadas pelas tabelas 5.7 e 5.8.

Tabela 5.7 Operadores relacionais entre vetores de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Igualdade entre dois vetores de intervalos ( $V=W$ )	svequ	v, w: svector	l: logical
V menor que W? ( $V<W$ )	svless	v, w: svector	l: logical
V maior que W? ( $V>W$ )	svgre	v, w: svector	l: logical
V está contido em W? ( $V\subseteq W$ )	svleq	v, w: svector	l: logical
V contém W? ( $V\supseteq W$ )	svgeq	v, w: svector	l: logical
v1 pertence a V? ( $v1 \in V$ )	rvinsv	v1: vector, v: svector	l: logical
V não é igual a W? ( $V \neq W$ )	svneq	v, w: svector	l: logical
V é interior de W? ( $V \overset{\circ}{\subseteq} W$ )	svxiny	v, w: svector	l: logical
V está contido propriamente em W ( $V \subset W$ )	svxcy	v, w: svector	l: logical
V contém propriamente W ( $V \supset W$ )	svxmy	v, w: svector	l: logical
V intersecção W é vazia? ( $V \cap W = \emptyset$ )	sviv	v, w: svector	l: logical

Tabela 5.8 Operações entre conjuntos de vetores intervalares

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
União de vetores de Intervalos ( $V \cup W$ )	svuni	V,W: svector	U: svector
União convexa de vetores de intervalos ( $V \cup_c W$ )	svunid	V,W: svector	U: svector
Intersecção de vetores de Intervalos ( $V \cap W$ )	svinter	V,W: svector	U: svector

- **Operações aritméticas** entre vetores de intervalos reais: neste grupo estão definidas as operações cujos argumentos são vetores de intervalos. Estas rotinas também podem ser acessadas pelos símbolos matemáticos que identificam as operações aritméticas entre reais. As operações entre vetores de intervalos são extensões das operações intervalares, elas são aplicadas em cada componente do vetor. A tabela 5.9 as relaciona.

Tabela 5.9 Operações aritméticas entre vetores de intervalos reais

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Soma de vetores de Intervalos ( $U=V+W$ )	svadd	V,W: svector	U: svector
Soma unária ( $U=+V$ )	svmt	V: svector	U: svector
Subtração de vetores de Intervalos ( $U=V-W$ )	svsub	V,W: svector	U: svector
Negação do vetor de intervalo ( $U=-V$ )	svneg	V: svector	U: svector
Multiplicação de vetores de intervalos ( $U=V*W$ )	svmult	V,W: svector	U: svector

- **Funções básicas elementares** também são estendidas para vetores de intervalos reais. As funções vetoriais têm o mesmo domínio das funções intervalares, sendo aplicadas em cada componente do vetor intervalar. Todos os elementos do vetor intervalar, argumento da função intervalar, têm que estar dentro do domínio para que a função seja validada.

Tabela 5.10 Funções Básicas de Vetores de Intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Valor Absoluto	svabs	V: svector	U: svector
X ao quadrado	svsqr	V: svector	U: svector
Raiz quadrada	svsqrt	V: svector	U: svector
Função exponencial	svexp	V: svector	U: svector
Função logaritmica neperiana	svln	V: svector	U: svector
Função logaritmica decimal	svlog	V: svector	U: svector
Potenciação	svpotn	V:svector n:integer	U: svector
Arcotangente	svarctan	V: svector	U: svector
Seno	svsin	V: svector	U: svector
Cosseno	svcos	V: svector	U: svector
Tangente	svtan	V: svector	U: svector

- **Funções pré-definidas** - entre as funções pré-definidas para vetores de intervalos reais estão as funções: *svnull* e *svum* que inicializam o vetor nulo e unitário, respectivamente.
- **Rotinas de entrada e saída** - neste grupo estão as rotinas de leitura e impressão de vetores de intervalos reais, elas são denominadas de *svread* e *svwrite*. Os elementos dos vetores devem ser armazenados em arquivos, um elemento por linha como é descrito em 5.3.5. A impressão de vetores de intervalos é baseada na impressão de intervalos, sendo um intervalo por linha.

### 5.3.2.2 Operações com matrizes de intervalos de reais

As operações com matrizes de intervalos são uma extensão natural das operações entre intervalos, aplicadas em cada componente da matriz. A operação só é validada se estiver definida para cada elemento da matriz de intervalos. Para fins de documentação, utilizou-se a denominação *matrix* para indicar matriz real e *smatrix* para matrizes de intervalos. Todas as rotinas de matrizes intervalares iniciam pelas letras *sm*. As funções que calculam as características geométricas de matrizes de intervalos, as rotinas aritméticas com matrizes, os operadores relacionais e as funções básicas elementares de matrizes de intervalos podem ser acessadas pelo nome da rotina ou pelo nome padrão da função real do Fortran.

- **Funções de transferência** entre matrizes reais e matrizes de intervalos seguem o padrão das funções de transferência de reais com intervalos reais, sendo estas aplicadas a cada elemento da matriz. Estão também presentes neste grupo as funções que calculam características geométricas de matrizes de intervalos, estas são descritas pela tabela 5.11.

Tabela 5.11 Funções de Transferência de Matrizes de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Carga de matriz intervalar degenerada ( $M=[m1.m1]$ )	smintpt	m1: matrix	M: smatrix
Carga de matriz intervalar ( $M=[m1.m2]$ )	smintval	m1,m2: matrix	M: smatrix
Busca do limite inferior - infimo	sminf	M: smatrix	m1: matrix
Busca do limite superior - supremo	smsup	M: smatrix	m1: matrix
Troca duas matrizes intervalares entre si	smswap	M,N: smatrix	M,N: smatrix
Copia uma matriz intervalar em outra	smcopy	M: smatrix	N: smatrix
Inflaciona uma matriz intervalar	smblow	M: smatrix, r: real	N: smatrix
Ponto Médio de matriz de intervalos (m(M))	smmed	M: smatrix	m1: matrix
Menor Valor Absoluto (<M>)	smin	M: smatrix	m1: matrix
Maior Valor Absoluto ( M )	smax	M: smatrix	m1: matrix
Módulo da matriz de intervalos ( M )	smmod	M: smatrix	m1: matrix
Distância entre duas matrizes intervalos (q(M,N))	smdis	M,N: smatrix	m1: matrix
Diâmetro de uma matriz de intervalo (d(M)≥0)	smdia	M: smatrix	m1: matrix
Diâmetro relativo de uma matriz de intervalo (drel(M))	smdiar	M: smatrix	m1: matrix
Raio de uma matriz de intervalo (r(M))	smraio	M: smatrix	m1: matrix

- **Operações relacionais e entre conjuntos** - neste grupo estão os operadores relacionais e as rotinas que manipulam conjuntos de matrizes intervalares. Elas somente serão válidas se forem definidas em cada componente da matriz. A relação dos operadores e rotinas são dadas pelas tabelas 5.12 e 5.13.

Tabela 5.12 Operadores relacionais entre matrizes de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Igualdade entre duas matrizes de intervalos ( $M=N$ )	smequ	M,N: smatrix	l: logical
M menor que N ( $M<N$ )	smless	M,N: smatrix	l: logical
M maior que N? ( $M>N$ )	smgre	M,N: smatrix	l: logical
M está contido em N? ( $M\subseteq N$ )	smleq	M,N: smatrix	l: logical
M contém N? ( $M\supseteq N$ )	smgeq	M,N: smatrix	l: logical
m1 pertence a M? ( $m1 \in M$ )	rminsm	m1: matrix, M: smatrix	l: logical
M não é igual a N? ( $M \neq N$ )	smneq	M,N: smatrix	l: logical
M é interior de N? ( $M \overset{\circ}{\subseteq} N$ )	smxiny	M,N: smatrix	l: logical
M está contido propriamente em N ( $M \subset N$ )	smxcy	M,N: smatrix	l: logical
M contém propriamente N ( $M \supset N$ )	smxmy	M,N: smatrix	l: logical
M intersecção N é vazia? ( $M \cap N = \emptyset$ )	smiv	M,N: smatrix	l: logical



Tabela 5.13 Operações entre conjuntos de matrizes intervalares

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
União de matrizes de intervalos ( $M \cup N$ )	smuni	M,N: smatrix	H: smatrix
União convexa de matrizes de intervalos ( $M \cup_c N$ )	smunid	M,N: smatrix	H: smatrix
Intersecção de matrizes de intervalos ( $M \cap N$ )	sminter	M,N: smatrix	H: smatrix

- **Operações aritméticas** entre matrizes de intervalos reais, neste grupo estão definidas as operações cujos argumentos são matrizes de intervalos reais. Estas rotinas também podem ser acessadas pelos símbolos matemáticos que identificam as operações aritméticas entre reais. As operações entre matrizes de intervalos são extensões das operações intervalares, sendo aplicadas em cada componente da matriz. A tabela 5.14 as relaciona.

Tabela 5.14 Operações aritméticas entre matrizes de intervalos reais

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Soma de matrizes de intervalos ( $H=M+N$ )	smadd	M,N: smatrix	H: smatrix
Soma unária ( $H=+M$ )	smmt	M: smatrix	H: smatrix
Subtração de matrizes intervalos ( $H=M-N$ )	smsub	M,N: smatrix	H: smatrix
Negação da matriz intervalar ( $H=-M$ )	smneg	M: smatrix	H: smatrix
Multiplicação de matrizes de intervalos ( $H=M*N$ )	smmult	M,N: smatrix	H: smatrix

- **Funções básicas elementares** também são estendidas para matrizes de intervalos reais. A função de argumento matricial tem o mesmo domínio das funções intervalares, sendo porém aplicada em cada componente da matriz intervalar. Todos os elementos da matriz intervalar, argumento da função intervalar, têm que estar dentro do domínio para que a função seja validada.

Tabela 5.15 Funções Básicas de Matrizes de Intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Valor Absoluto	smabs	M:smatrix	H:smatrix
X ao quadrado	smsqr	M:smatrix	H:smatrix
Raiz quadrada	smsqrt	M:smatrix	H:smatrix
Função exponencial	smexp	M:smatrix	H:smatrix
Função logarítmica neperiana	smln	M:smatrix	H:smatrix
Função logarítmica decimal	smlog	M:smatrix	H:smatrix
Potenciação	smpotn	M:smatrix,b:integer	H:smatrix
Arcotangente	smarctan	M:smatrix	H:smatrix
Seno	smsin	M:smatrix	H:smatrix
Cosseno	smcos	M:smatrix	H:smatrix
Tangente	smtan	M:smatrix	H:smatrix

- **Funções pré-definidas** - entre as funções pré-definidas para matrizes de intervalos, têm-se: *smtransp*, que calcula a matriz transposta, *smnull*, que inicializa a matriz nula e a função *smid*, que inicializa a matriz identidade.
- **Rotinas de entrada e saída** - neste grupo estão as rotinas de leitura e impressão de matrizes de intervalos reais, elas são denominadas de *smread* e *smwrite*. As matrizes são lidas e impressas por colunas. No início da impressão de cada coluna é dado o número da coluna. A impressão por linhas pode ser obtida utilizando-se a rotina de impressão de vetores, ajustando os índices para as linhas. Detalhes são dados em 5.3.5.

### 5.3.2.3 Operações de diferentes dados com intervalos, vetores e matrizes de intervalos

Este conjunto de rotinas abrange as operações aritméticas entre diferentes tipos de dados, como: reais com intervalos, vetores e matrizes de intervalos; intervalos com vetores e matrizes de intervalos, bem como entre vetores e matrizes de intervalos. A tabela 5.16 inclui as operações aritméticas com tipos de dados intervalares diferentes, disponíveis na biblioteca *libavi.a*.

Tabela 5.16 Tabela dos operadores aritméticos pré-definidos

right left \	real	interval	vector	svector	matrix	smatrix	scomplex
real	+, -, *, /	+, -, *, /	+, -, *	+, -, *	+, -, *	+, -, *	
interval	+, -, *, /	+, -, *, /		+, -, *		+, -, *	
vector	+, -, *, /		+, -, *	+, *	*	*	
svector	+, -, *, /	+, -, *, /	+, -, *	+, -, *	*	*	
matrix	+, -, *, /		*	*	+, -, *	+, *	
smatrix	+, -, *, /	+, -, *, /	*	*	+, *	+, -, *	
scomplex							+, -, *, /

Estas operações podem ser acessadas pelo nome da subrotina, indicado nas tabelas 5.17 a 5.23, ou pelo símbolo matemático da operação aritmética. O próprio compilador seleciona a rotina para os tipos de dados passados como parâmetros. O tipo de dado resultante é sempre o maior tipo de dado envolvido. A seguir são dados os sete conjuntos de rotinas que compõem esta terceira parte do módulo *mvi*.

- **Operações de reais com intervalos** - neste conjunto estão as quatro operações aritméticas básicas de reais com intervalos. O tipo resultante é sempre intervalo. Está também incluída a operação de união convexa de real com intervalo, onde o resultado pode ser o próprio intervalo, no caso de real estar contido, ou então o intervalo resultante possuirá, como um dos limites, o real. A idéia destas operações é converter o real num intervalo degenerado e, então, operar os intervalos.

Tabela 5. 17 Operações aritméticas de reais com intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Real mais intervalo $c = r + a$	radds	r: real, a: interval	c: interval
Intervalo mais real $c = a + r$	saddr	r: real, a: interval	c: interval
Real menos intervalo $c = r - a$	rsubs	r: real, a: interval	c: interval
Intervalo menos real $c = a - r$	ssubr	r: real, a: interval	c: interval
Real vezes intervalo $c = r * a$	rmults	r: real, a: interval	c: interval
Intervalo vezes real $c = a * r$	smultr	r: real, a: interval	c: interval
Real dividido por intervalo $c = r / a$	rdivs	r: real, a: interval	c: interval
Intervalo dividido por real $c = a / r$	sdivr	r: real, a: interval	c: interval
União convexa de real com intervalo	runids	r: real, a: interval	c: interval
União convexa de intervalo com real	sunidr	r: real, a: interval	c: interval

- **Operações de reais com vetores de intervalos** - neste conjunto de operações de reais com vetores de intervalos, também segue-se a idéia de converter o real em um intervalo degenerado e, então, operá-lo com o vetor de intervalos. O intervalo degenerado é operado com cada componente do vetor. A operação de soma também é conhecida como inflação do vetor intervalar, quando o valor real somado é um valor bem pequeno. Somente a divisão do vetor intervalar por real foi providenciada.

Tabela 5.18 Operações aritméticas de reais com vetores de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Real mais vetor de intervalo $w = r + v$	raddsv	r: real, v: svector	w: svector
Vetor de intervalo mais real $w = v + r$	svaddr	r: real, v: svector	w: svector
Real menos vetor de intervalo $w = r - v$	rsubsv	r: real, v: svector	w: svector
Vetor de intervalo menos real $w = v - r$	svsubr	r: real, v: svector	w: svector
Real vezes vetor de intervalo $w = r * v$	rmultsv	r: real, v: svector	w: svector
Vetor de intervalo vezes real $w = v * r$	svmultr	r: real, v: svector	w: svector
Vetor de intervalo dividido por real $w = v / r$	sdivr	r: real, v: svector	w: svector

- **Operações de reais com matrizes de intervalos** - neste conjunto de operações de reais com matrizes de intervalos, também segue-se a idéia de converter o real em um intervalo degenerado e, então, operá-lo com a matriz de intervalos. O intervalo degenerado é operado com cada elemento da matriz. A operação de soma também é conhecida como inflação da matriz intervalar, quando o valor real somado é um valor bem pequeno. Somente a divisão de matriz intervalar por real foi providenciada.

Tabela 5.19 Operações aritméticas de reais com matrizes de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Real mais matriz de intervalo $H = r + M$	raddsm	r: real, M: smatrix	H: smatrix
Matriz de intervalo mais real $H = M + r$	smaddr	r: real, M: smatrix	H: smatrix
Real menos matriz de intervalo $H = r - M$	rsubsm	r: real, M: smatrix	H: smatrix
Matriz de intervalo menos real $H = M - r$	smsubr	r: real, M: smatrix	H: smatrix
Real vezes intervalo $H = r * M$	rmultsm	r: real, M: smatrix	H: smatrix
Matriz de intervalo vezes real $H = M * r$	smmultr	r: real, M: smatrix	H: smatrix
Matriz de intervalo dividido por real $H = M / r$	smdivr	r: real, M: smatrix	H: smatrix

- **Operações de intervalos com vetores de intervalos** - este conjunto de operações é a generalização, pois agora se têm intervalos genéricos operados com vetores de intervalos, sendo que a operação é feita componente a componente do vetor de intervalo. Tem-se somente a divisão de um vetor de intervalos por um intervalo, que é feita como a multiplicação do vetor intervalar pelo pseudo inverso multiplicativo componente a componente.

Tabela 5.20 Operações aritméticas de intervalos com vetores de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Intervalo mais vetor de intervalo $w = a + v$	saddsv	v: svector, a: interval	w: svector
Vetor de intervalo mais intervalo $w = v + a$	svadds	v: svector, a: interval	w: svector
Intervalo menos vetor de intervalo $w = a - v$	ssubsv	v: svector, a: interval	w: svector
Vetor de intervalo menos intervalo $w = v - a$	svsubs	v: svector, a: interval	w: svector
Intervalo vezes vetor de intervalo $w = a * v$	smultsv	v: svector, a: interval	w: svector
Vetor de intervalo vezes intervalo $w = v * a$	svmults	v: svector, a: interval	w: svector
Vetor de intervalo dividido por intervalo $w = v / a$	svdivs	v: svector, a: interval	w: svector

- **Operações de intervalos com matrizes de intervalos** - este conjunto de operações é a generalização, pois agora se têm intervalos genéricos operados com vetores de intervalos, sendo que a operação é feita componente a componente do vetor de intervalo. Tem-se somente a divisão de um vetor de intervalos por um intervalo, que é feita como a multiplicação do vetor intervalar pelo pseudo inverso multiplicativo componente a componente.



Tabela 5.21 Operações aritméticas de intervalos com matrizes de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Intervalo mais matriz de intervalo $H = a + M$	saddsm	a:interval, M: smatrix	H: smatrix
Matriz de intervalo mais intervalo $H = M + a$	smadds	a:interval, M: smatrix	H: smatrix
Intervalo menos matriz de intervalo $H = a - M$	ssubsm	a:interval, M: smatrix	H: smatrix
Matriz de intervalo menos intervalo $H = M - a$	smsubs	a:interval, M: smatrix	H: smatrix
Intervalo vezes matriz de intervalo $H = a * M$	smultsm	a:interval, M: smatrix	H: smatrix
Matriz de intervalo vezes intervalo $H = M * a$	smmults	a:interval, M: smatrix	H: smatrix
Matriz de intervalo dividido por intervalo $H = M/a$	smdivs	a:interval, M: smatrix	H: smatrix

- **Operações entre vetores e matrizes de intervalos** - neste conjunto só foi incluída a operação de multiplicação, por ser a única de real aplicabilidade. Estas operações são limitadas a condições de compatibilidade entre as dimensões do vetor e da matriz de intervalo. O produto é sempre feito linha por coluna, logo esta condição sempre é verificada antes da execução da operação.

Tabela 5.22 Operações aritméticas entre vetores e matrizes de intervalos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Matriz de intervalos vezes vetor de intervalos $W_{n \ 1} = M_{n \ k} * V_{k \ 1}$	smmultsv	V: svector, M: smatrix	W: svector
Vetor de intervalos vezes matriz de intervalos $W_{1 \ m} = V_{1 \ n} * M_{n \ m}$	svmultsm	V: svector, M: smatrix	W: svector

- **Operações aritméticas entre vetores e matrizes reais com vetores e matrizes de intervalos** - neste conjunto foram incluídas as operações aritméticas de vetores e matrizes reais com vetores ou matrizes de intervalos. O resultado é sempre intervalar, pois os dados reais são tratados como intervalos degenerados. Estas rotinas foram implementadas utilizando as rotinas definidas anteriormente de reais com intervalos, elas são aplicadas componente a componente. A multiplicação de vetores reais com intervalares é um produto escalar cujo resultado é um intervalo.

Tabela 5.23 Operações entre vetores e matrizes reais e intervalares

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Soma de vetores $sw = rv + sv$	<b>rvaddsv</b>	sv: svector, rv:vector	sw: svector
Soma de vetores $sw = sv + rv$	<b>svaddrv</b>	sv: svector, rv:vector	sw: svector
Subtração de vetores $sw = sv - rv$	<b>svsubrv</b>	sv: svector, rv:vector	sw: svector
Multiplicação de vetores $a = rv \cdot sv$	<b>rvmultsv</b>	sv: svector, rv:vector	a: interval
Multiplicação de vetores $a = sv \cdot rv$	<b>svmultrv</b>	sv: svector, rv:vector	a: interval
Soma de matrizes $sn = rm + sm$	<b>rnaddsm</b>	rm: matrix, sm: smatrix	sn: smatrix
Soma de matrizes $sn = sm + rm$	<b>smaddrm</b>	rm: matrix, sm: smatrix	sn: smatrix
Subtração de matrizes $sn = sm - rm$	<b>smsubrm</b>	rm: matrix, sm: smatrix	sn: smatrix
Multiplicação de matrizes $sn = sm \cdot rm$	<b>smmultrm</b>	rm: matrix, sm: smatrix	sn: smatrix
Multiplicação de matrizes $sn = rm \cdot sm$	<b>rmmultsm</b>	rm: matrix, sm: smatrix	sn: smatrix
Multiplicação vetor-matriz $sw = sm \cdot rv$	<b>smmultrv</b>	rv: vector, sm: smatrix	sw: svector
Multiplicação vetor-matriz $sw = rm \cdot sv$	<b>rmmultsv</b>	sv: svector, rm: matrix	sw: svector

### 5.3.3 Módulo de Intervalos complexos - *ci*

Um número complexo é composto por  $z = z_{re} + i z_{im}$ , onde  $z_{re}$  e  $z_{im}$  correspondem às partes real e imaginária do número complexo, sendo ambos reais. Num intervalo complexo tanto a parte real quanto a parte imaginária são intervalos reais. A definição de intervalos complexos também está contida no arquivo *inter.inc* e é mostrada pela figura 5.3. O módulo dos intervalos complexos é o *ci*, que contém seis grupos de rotinas da mesma forma que o módulo *básico*. Como a definição do tipo de dado intervalo complexo está contida no arquivo *inter.inc*, o módulo deve ter a inclusão deste arquivo, assim como todas as rotinas deste módulo. A inclusão é feita pelo comando *include 'inter.inc'*.

```

type cinterval
  sequence
  type (interval) re, im
end type cinterval

```

Fig 5.3 Definição de intervalos complexos arquivo *inter.inc*

Para fins de documentação, foi denominado o tipo intervalo complexo como *scomplex*. Para o uso na programação é mantida a nomenclatura existente na bibliografia, ou seja, *cinterval*. Este módulo inclui e utiliza em todas as suas rotinas o módulo *básico*, onde foram definidas as operações e funções sobre intervalos. Nem todas as operações sobre intervalos complexos são extensões naturais das operações intervalares, portanto recomenda-se a leitura do capítulo 2, onde a aritmética complexa intervalar é descrita.

Este módulo não contém operações entre vetores e matrizes de intervalos complexos, pois isto ultrapassa os objetivos propostos neste trabalho. Entretanto são dadas as definições de vetores e matrizes de intervalos complexos (figura 5.4). As operações para estes tipos de dados são extensões naturais dos intervalos complexos.

*type (cinterval), dimension(:) :: compl\_int\_vector*

*type (cinterval), dimension(:,:) :: compl\_int\_matrix*

Fig 5.4 Definição de vetores e matrizes de intervalos complexos

Nas definições da figura 5.4, analogamente às definições de vetores e matrizes de intervalos reais, são utilizados *arrays* dinâmicos, que possibilitam que os vetores e matrizes tenham um tamanho genérico. Na definição de vetor o comando *dimension* possui uma dimensão indicada por dois pontos, já na definição de matriz existem duas vezes o sinal de dois pontos, indicando que a matriz é bidimensional.

- **Funções de transferência** - Neste grupo estão as rotinas que inicializam intervalos complexos, fornecem os extremos e a parte real ou imaginária dos intervalos complexos. A tabela 5.24 fornece as rotinas implementadas deste grupo, os parâmetros de entrada e de saída (resultado) e a descrição. Um intervalo complexo  $z$  pode ser escrito na forma:  $z = [x.inf, x.sup] + [y.inf, y.sup]$ . Então existem duas maneiras de se carregarem intervalos complexos. Na primeira intervalos reais ou reais são convertidos em intervalos complexos, por isso a rotina se denomina *sccompl*. Na segunda maneira, introduzem-se valores complexos, a partir dos quais são montados os intervalos da parte real e da parte imaginária. Como se introduzem valores complexos a rotina se denomina *scintval*. Caso um dos parâmetros seja um real, implicará que a parte imaginária correspondente seja nula, ou seja, um dos limites do intervalo da parte imaginária será nulo. Observa-se que os valores de entrada devem ser consistentes para constituir um intervalo.

Da mesma forma existem, entre as funções de transferência, rotinas que para dado intervalo complexo fornecem os limites inferior e superior do intervalo complexo; no caso, estes limites são valores complexos (funções *scinf*, *scsup*) e para este mesmo intervalo existem rotinas que fornecem os intervalos que compõem a parte real e a parte imaginária (funções *scre* e *scim*).

Tabela 5.24 Funções de Transferência de intervalos complexos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Carga de intervalo complexo por intervalos ( $Z = (a1, a2) = a1 + a2 i$ )	sccompl	a1,a2: interval r: real, a2: interval a1: interval, r: real a1: interval <small>(a2=0)</small>	Z: scomplex
Carga de intervalo complexo por complexos ( $Z = (z1, z2) = z1 + z2i$ onde $z1 = [c1.re, c2.re]$ $z2 = [c1.im, c2.im]$ ) e $c1 = c1.re + ic1.im$ , $c2 = c1.re + ic2.im$	scintval	c1,c2: complex r: real; c2: complex c1: complex, r: real c1: complex <small>(degenerado)</small>	Z: scomplex
Cálculo do complexo intervalar conjugado	scconj	Z: scomplex	Zc: scomplex
Busca do limite inferior da parte real - ínfimo real	screinf	Z: scomplex	z1re: real
Busca do limite superior da parte real - supremo real	scresup	Z: scomplex	z2re: real
Busca do limite inferior da parte imaginária - ínfimo imaginário	sciminf	Z: scomplex	z1im: real
Busca do limite superior da parte imaginária - supremo imaginário	scimsup	Z: scomplex	z2im: real
Busca do intervalo da parte real	scre	Z: scomplex	a1: interval
Busca do intervalo da parte imaginária	scim	Z: scomplex	a2: interval
Busca do limite inferior - ínfimo	scinf	Z: scomplex	c1: complex
Busca do limite superior - supremo	scsup	Z: scomplex	c2: complex
Ponto Médio (m(Z))	scmed	Z: scomplex	z1: complex
Diâmetro de um intervalo ( $d(Z) \geq 0$ )	scdia	Z: scomplex	z1: complex
Troca de dois intervalos complexos $Z1 \leftrightarrow Z2$	scswap	Z1,Z2: scomplex	Z1,Z2: scomplex
Inflaciona o intervalo complexo	scblow	Z: scomplex, r:real	Z: scomplex
Cópia do intervalo complexo $Z2 = Z1$	sccopy	Z1: scomplex	Z2: scomplex
Ângulo da representação de coordenadas polares	scang	Z: scomplex	a: interval
Raio da representação de coordenadas polares	scraio	Z: scomplex	b: interval

• **Operações relacionais** - São rotinas que relacionam intervalos complexos, fazendo comparações entre eles. As rotinas deste grupo são enumeradas pela tabela 5.25, cuja descrição contém a relação das rotinas e sua notação matemática. Os operadores relacionais são acessados pelo nome da rotina ou pelo operador relacional definido em Fortran para reais, pois o compilador verifica o tipo de dado e carrega a rotina correta. O resultado é sempre uma variável lógica.



Tabela 5.25 Operadores relacionais entre intervalos complexos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Igualdade entre dois intervalos complexos ( $Z_a=Z_b$ )	scequ	Za,Zb: scomplex	l: logical
Za menor que Zb? ( $Z_a<Z_b$ )	scless	Za,Zb: scomplex	l: logical
Za maior que Zb? ( $Z_a>Z_b$ )	scgre	Za,Zb: scomplex	l: logical
Za está contido em Zb? ( $Z_a\subseteq Z_b$ )	scleq	Za,Zb: scomplex	l: logical
Za contém Zb? ( $Z_a\supseteq Z_b$ )	scgeq	Za,Zb: scomplex	l: logical
z1 pertence a Za? ( $z1 \in Z_a$ )	rcinsc	z1:complex,Za: scomplex	l: logical
Za não é igual a Zb? ( $Z_a \neq Z_b$ )	scneq	Za,Zb: scomplex	l: logical
Za é interior de Zb? ( $Z_a \subset Z_b$ )	scxiny	Za,Zb: scomplex	l: logical
Za está contido propriamente em Zb ( $Z_a \subset Z_b$ )	scxey	Za,Zb: scomplex	l: logical
Za contém propriamente Zb ( $Z_a \supset Z_b$ )	scxmy	Za,Zb: scomplex	l: logical
Za intersecção Zb é vazia? ( $Z_a \cap Z_b = \emptyset$ )	sciv	Za,Zb: scomplex	l: logical

- **Operações entre conjuntos** - As operações sobre conjuntos foram estendidas a intervalos complexos. Estas operações são a união, a união convexa, cujo resultado é o intervalo maximal e a intersecção. Na tabela 5.26 estas operações estão relacionadas com seus tipos de dados de parâmetros e do resultado.

Tabela 5.26 Operações entre conjuntos complexos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
União de intervalos complexos ( $Z_a \cup Z_b$ )	scuni	Za,Zb: scomplex	Zc: scomplex
União convexa de intervalos complexos ( $Z_a \cup_c Z_b$ )	scunid	Za,Zb: scomplex	Zc: scomplex
Intersecção de intervalos complexos ( $Z_a \cap Z_b$ )	scinter	Za,Zb: scomplex	Zc: scomplex

- **Operações aritméticas** - Este grupo de operações possui os extremos inflacionados após a realização do cálculo, feito através das rotinas *infla\_down* e *infla\_up*, que emulam os arredondamentos direcionados. Elas são definidas pela aritmética intervalar complexa descrita no capítulo 2.

Tabela 5.27 Operações aritméticas entre intervalos complexos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Soma de intervalos complexos ( $Z_c=Z_a+Z_b$ )	scadd	Za,Zb: scomplex	Zc: scomplex
Soma unária ( $Z_c=+Z_a$ )	scmt	Za: scomplex	Zc: scomplex
Subtração de intervalos complexos ( $Z_c=Z_a-Z_b$ )	scsub	Za,Zb: scomplex	Zc: scomplex
Negação do intervalo complexo ( $Z_c=-Z_a$ )	scneg	Za: scomplex	Zc: scomplex
Multiplicação de intervalos complexos ( $Z_c=Z_a*Z_b$ )	scmult	Za,Zb: scomplex	Zc: scomplex
Divisão de intervalos complexos ( $Z_c=Z_a/Z_b$ )	sctdiv	Za,Zb: scomplex	Zc: scomplex
Inversão de intervalo complexo ( $Z_c=1/Z_a$ )	scinv	Za: scomplex	Zc: scomplex

- **Funções Elementares de Intervalos complexos** - Neste grupo estão as funções elementares algébricas, logarítmicas, exponenciais e trigonométricas de argumentos intervalares complexos. O domínio de definição destas funções é dado juntamente na tabela 5.28. Pode-se utilizar as funções pelo nome matemático ou o mnemônico, pois dependendo do tipo de dado, o compilador carregará a função correta.

Tabela 5.28 Funções Elementares de Intervalos Complexos

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Valor Absoluto	scabs	Za: scomplex	a: interval
X ao quadrado	scsqr	Za: scomplex	Zc: scomplex
Raiz quadrada	scsqrt	Za: scomplex	Zc: scomplex
Função exponencial	scexp	Za: scomplex	Zc: scomplex
Função logarítmica neperiana	scln	Za: scomplex	Zc: scomplex
Função logarítmica decimal	sclog	Za: scomplex	Zc: scomplex
Potenciação	scpotn	Za: scomplex,b:interger	Zc: scomplex
Arcotangente	scarctan	Za: scomplex	Zc: scomplex
Seno	scsin	Za: scomplex	Zc: scomplex
Cosseno	sccos	Za: scomplex	Zc: scomplex
Tangente	sctan	Za: scomplex	Zc: scomplex

- **Rotinas de entrada e saída** - neste grupo estão as rotinas de leitura e impressão de intervalos complexos, elas são denominadas de *scread* e *sewrite*. Como cada intervalo complexo é composto por quatro *reais*, na leitura/impressão são lidos/impressos quatro valores por linha. Eles correspondem aos intervalos da parte real e da parte imaginária, como é descrito em 5.3.5.

### 5.3.4 Módulo de aplicações - *aplic*

O módulo de aplicações é composto de operações que correspondem aos três níveis da biblioteca BLAS e de rotinas que facilitam o desenvolvimento de métodos intervalares para solução de sistemas lineares. As operações tipo BLAS são para matrizes genéricas, não há versões específicas para cada tipo de matriz. As rotinas são enumeradas pela tabela 5.29.

- **Rotinas entre vetores da álgebra linear** - estas rotinas realizam operações básicas entre vetores do tipo combinações lineares SAXPY de vetores. São calculadas: a norma de um vetor, a soma dos valores absolutos e dos elementos de vetores. Foi implementado o produto escalar em dupla precisão. Estas rotinas estão contidas no conjunto de rotinas do nível 1 da BLAS.
- **Rotina entre matriz e vetores da álgebra linear** - esta rotina realiza a operação do tipo  $y = rAx + sy$ , onde A é a matriz, x e y são vetores e r e s são reais. Esta rotina é uma das rotinas do nível 2 da BLAS. Esta rotina também é calculada em dupla precisão.

- **Rotinas entre matrizes da álgebra linear** - estas rotinas realizam operações entre matrizes do tipo  $C = rAB + sC$ , em precisão simples e dupla. Elas correspondem a operações do nível 3 da BLAS.

Tabela 5.29 Rotinas entre matrizes e vetores da álgebra linear

DESCRIÇÃO DA ROTINA	NOME	PARÂMETROS	RESULTADO
Soma de um múltiplo escalar do vetor intervalar a outro vetor intervalar $w = r.x + y$ (operação elementar)	svsaxpy	r: real; x,y: svector	w: svector
Produto escalar usando a rotina dot da BLAS	svdot	x,y: svector	a: interval
Produto escalar em dupla precisão	svddot	x,y: svector	a: interval
Operação $w = sy + r.A.x$ (cálculo do resíduo)	sgemv	r,s:real;A:matrix x,y: svector	w: svector
Operação $w = sy + r.Ax$ em dupla precisão	sdgemv	r,s:real;A:matrix x,y: svector	w: svector
Operação $D = sC + rAB$	sgemm	r,s:real; A,B,C: smatrix	D: smatrix
Operação $D = sC + rAB$ em dupla precisão	sdgemm	r,s:real; A,B,C: smatrix	D: smatrix
Cálculo da matriz inversa por Gauss-Jordan( $H=M^{-1}$ )	rminv	M: smatrix	H: matrix
Cálculo da matriz inversa intervalar( $H=M^{-1}$ )	sminv	M: smatrix	H: smatrix
Cálculo da norma euclidiana de um vetor de intervalos	svnorm2	x: svector	r: real
Norma máxima de linha de Matriz real	rnorm1	M: matrix	r: real
Norma máxima de linha de Matriz de intervalo	rmnorm1	M: smatrix	r: real
Norma máxima de linha de Matriz de intervalo	smnorm1	M: smatrix	s: interval
Norma máxima de coluna de Matriz real	rnormc	M: matrix	r: real
Norma máxima de coluna de Matriz de intervalo	rmnormc	M: smatrix	r: real
Norma máxima de coluna de Matriz de intervalo	smnormc	M: smatrix	s: interval
Norma Euclidiana de Matriz real	rnorm2	M: matrix	r: real
Norma Euclidiana de Matriz de intervalo	rmnorm2	M: smatrix	r: real
Norma Euclidiana de Matriz de intervalo	smnorm2	M: smatrix	a: interval
Soma dos valores absolutos dos elementos de um vetor real	rvasum	x: vector	r: real
Soma dos valores dos elementos de um vetor real	rvsun	x: vector	r: real
Soma dos valores absolutos dos elementos de um vetor de intervalos	svasum	x: svector	a: interval
Soma dos valores dos elementos de um vetor de intervalos	svsun	x: svector	a: interval
Cálculo do determinante da matriz real (via MEG)	rdet	M: matrix	r: real
Cálculo do determinante de matriz intervalar	smdet	M: smatrix	r: real
Cálculo do condicionamento da matriz (norma euclidiana)	rconda	M: matrix	r: real
Retrosustituição M é triangular inferior	retro	M: matrix, y: vector	x: vector
Calcula matriz intervalar de Hilbert de ordem n	smhilbn	n: integer	H: smatrix
Calcula o determinante da Matriz de Hilbert de ordem n	smdeth	n: integer	r: real

### 5.3.5 Rotinas de entrada e saída de tipos intervalares

Este item trata das rotinas de leitura e impressão de intervalos, de vetores de intervalos, de matrizes de intervalos e de intervalos complexos. Para a definição destas rotinas levou-se em conta: a característica do processamento no Cray ser muito rápido, por não ser um processamento interativo, não se perder tempo aguardando a edição de valores; as características de facilidade de uso e as peculiaridades dos comandos *read* e *write* em Fortran 90.

Para a utilização de arquivos de dados na leitura de tipos intervalares é necessário, no início do programa, definir e abrir o arquivo de dados *nome.dat*. O comando em Fortran que realiza tal operação é o *open*. Sua sintaxe é dada por (5.1), onde é definido '5' para arquivo de entrada de dados. O arquivo *nome.dat* é o arquivo de dados do problema a ser resolvido.

$$\text{open ( unit = 5, file = 'nome.dat')} \quad (5.1)$$

A leitura de valores reais em Fortran 90 é realizada pelo comando *read*. Uma referência para a leitura de reais é feita como mostra (5.2). O formato de leitura é especificado pelo segundo campo entre parêntesis. Este formato pode ser especificado segundo a sintaxe do comando *format* ou ser em formato livre, para tanto utiliza-se do caracter \*, como no caso do comando (5.2).

$$\text{read( 5, * ) r1, ..., rn} \quad (5.2)$$

A extensão do comando *read* para intervalos é feita como se vê em (5.3). Como é uma subrotina, deve ser referenciada pelo comando *call*. Adotou-se que na rotina *sread* os dados serão lidos de arquivos. O formato dos dados no arquivo é livre, para simplificar o acesso para os usuários; mas cada linha do arquivo deve conter apenas um intervalo, que corresponde a dois valores reais, onde o primeiro será atribuído ao extremo inferior e o segundo ao extremo superior do intervalo. A validação da execução se dará se o inferior for menor ou igual ao superior, caso contrário será emitida mensagem de erro e a execução suspensa. Em (5.3) *s1* é a variável que receberá o valor intervalar lido.

$$\text{call sread (5, s1 )} \quad (5.3)$$

Para leitura de vetores de intervalos, a extensão é similar, como mostra (5.4). Na referência da rotina *svread*, a variável *sv1* é do tipo *svector*, cuja dimensão foi definida na declaração. Por exemplo em (5.5) a variável possui dez componentes ou elementos. Portanto são lidos tantos intervalos quantas forem especificadas as dimensões do vetor de intervalos. Cada elemento do vetor é um intervalo e, portanto, será lido de uma linha do arquivo de dados. Mais uma vez se convencionou que a cada linha serão lidos, em formato livre, dois valores reais, correspondendo o primeiro ao extremo inferior e o segundo ao



extremo superior. A rotina só será validada, se todos os elementos forem intervalos, ou seja, se a leitura dos valores dos intervalos de cada componente do vetor for validada. Caso para algum elemento a rotina não seja validada, será emitida uma mensagem de erro e a execução suspensa.

```
call svread ( 5, sv1 ) (5.4)
```

```
type ( interval ), dimension ( 10 ) :: sv1 (5.5)
```

Da mesma forma, a rotina *smread* realiza a leitura dos valores intervalares de cada componente da matriz de intervalos. Os valores lidos corresponderão a um intervalo por linha do arquivo, sendo que os valores serão introduzidos por colunas, ou seja, primeiro serão lidos os intervalos que compõem a primeira coluna da matriz, depois os da segunda coluna e assim por diante até a última coluna. A rotina só será validada se forem válidos todos os intervalos-elementos da matriz. Se para algum elemento a rotina falhar será emitida mensagem de erro e suspensa a execução. Em (5.6) é dada uma referência da rotina. A variável *sm1* é do tipo *smatrix* e a dimensão é especificada na declaração da variável, como mostra (5.7).

```
call smread ( 5, sm1 ) (5.6)
```

```
type ( interval), dimension ( n, m ) :: sm1 (5.7)
```

A rotina que implementa a leitura de intervalos complexos é a *scread* e segue o mesmo princípio, mas para cada intervalo complexo devem ser lidos quatro valores reais, correspondendo o primeiro ao extremo inferior real, o segundo ao extremo superior real, o terceiro ao extremo inferior imaginário e o último, ao extremo superior imaginário. Uma referência desta rotina é dada em (5.8), onde *sc1* é do tipo *scomplex*, composta por dois intervalos ou quatro reais.

```
call scread ( 5, sc1 ) (5.8)
```

As rotinas de impressão de tipos intervalares são extensões do comando *write* para valores reais, o qual é referenciado como em (5.9). Esta referência contém o uso mais simples da rotina, sendo '6' a indicação de arquivo de saída padrão e '\*' a indicação de uso do formato livre. As variáveis *r1* a *rn* são do tipo real.

```
write ( 6, * ) r1, ..., rn (5.9)
```

A rotina que imprime intervalos reais é a *swrite*. Como é do tipo subrotina, uma referência deve ser precedida pelo comando *call*, como mostra (5.10). Ela se utiliza da rotina de impressão de reais para imprimir os dois extremos, inferior e superior do intervalo. Os valores são separados por vírgula e ficam entre colchetes. Sempre é impresso um intervalo por linha.

call swrite ( 6, s1) (5.10)

A impressão de um vetor de intervalos é realizada pela rotina *swrite* e são impressos os intervalos-elementos em formato livre de reais, um intervalo por linha. Ao final do vetor é impressa uma linha em branco para indicar o final do vetor. Uma referência da rotina é dada em (5.11).

call svwrite( 6, sv1) (5.11)

A impressão de matrizes de intervalos se dá por colunas. Antes de cada coluna é impressa a mensagem "coluna k", onde k representa a coluna que está sendo impressa e varia de um até número de colunas. A cada bloco de intervalos, correspondentes a colunas, são impressos tantos intervalos quanto o número de linhas da matriz. Ao final de cada coluna é impressa uma linha em branco e ao final da matriz são impressas duas linhas em branco. A rotina que imprime matrizes é denominada *smwrite*. Pode-se imprimir linhas da matriz, para tanto basta utilizar a rotina que imprime vetores adequando os índices, variando por linha. O formato de cada intervalo impresso é o mesmo definido para a rotina *swrite*, pois a rotina *smwrite* se utiliza desta rotina. Uma referência é dada em (5.12).

call smwrite( 6, sm1) (5.12)

A impressão de intervalos complexos se dá pela rotina *scwrite*. Ela imprime dois intervalos entre parêntesis, separados por vírgula. Cada intervalo possui dois reais separados por vírgula e entre colchetes. Portanto são impressos quatro valores reais para cada intervalo complexo, onde os dois primeiros correspondem aos extremos do intervalo da parte real e os dois últimos ao intervalo da parte complexa. O número imaginário *i* não é impresso, uma vez que o intervalo complexo é dado na forma de par ordenado de intervalos. Uma referência é dada em (5.13).

call scwrite( 6, sc1) (5.13)

Observa-se que nesta primeira versão da biblioteca *libavi.a*, só está disponível a rotina de impressão em formato livre. Para a próxima versão, está projetado o desenvolvimento de rotinas que possibilitam ao usuário especificar o formato de saída dos dados intervalares.

Da mesma forma, a leitura de dados é feita somente por arquivos em formato livre. Em caso de erros de consistência de intervalos na leitura, a execução é suspensa, pois se os dados não são intervalos não podem ser operados. Não há processamento interativo na leitura de intervalos. No caso de vetores e matrizes, caso ocorra a inconsistência de intervalos em uma componente, no final da leitura de todos os elementos será avisado o erro. Maiores detalhes destas rotinas podem ser encontrados no manual de utilização da *libavi.a*, referenciado por [DIV95c].

### 5.3.6 Hierarquia dos módulos

Como ilustra a figura 5.5, a biblioteca de rotinas intervalares é composta de quatro módulos. O módulo *básico* inclui o arquivo que contém a definição de intervalos reais e complexos. Neste módulo são implementadas todas as operações entre intervalos reais. Estas operações servem de base a todos os demais módulos.

O módulo *mvi* inclui o módulo *básico* e é incluído no módulo de aplicações *aplic*. Este módulo implementa todas as operações entre vetores de intervalos, matrizes de intervalos e rotinas de diferentes tipos de dados com vetores e matrizes de intervalos. O módulo *aplic* inclui o módulo *mvi* e o *básico*. Todas as rotinas dos dois módulos estão disponíveis.

O módulo dos intervalos complexos *ci* contém o módulo *básico*. Não foram implementadas rotinas que manipulam vetores e matrizes de intervalos complexos, mas podem ser facilmente implementados a partir do módulo *mvi* e *ci*. Basta estender as operações complexas para cada elemento do vetor ou matriz de intervalo complexo, de forma análoga à feita com intervalos reais.

Os métodos intervalares não foram incluídos na *libavi.a*, por serem específicos a cada problema que se quer resolver. Definiu-se a biblioteca *libselint.a* como sendo uma biblioteca científica que agrupa as rotinas que implementam os métodos intervalares de solução de sistemas lineares.

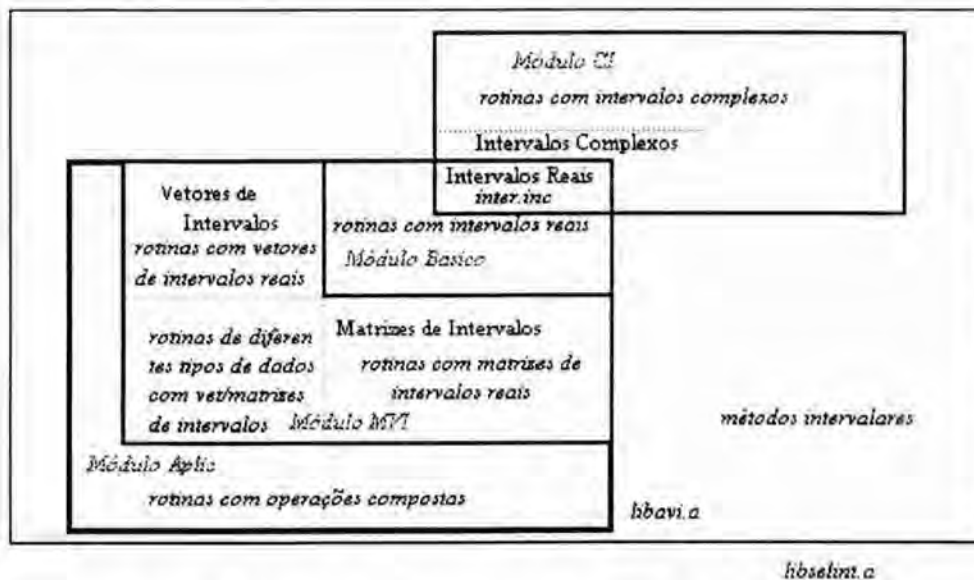


Fig 5.5 Hierarquia dos módulos da biblioteca *libavi.a*

#### 5.4 A Biblioteca de rotinas científicas intervalares *libselint.a*

Neste item serão feitas algumas considerações sobre a proposta da biblioteca de rotinas científicas. Algumas de suas rotinas foram desenvolvidas com o objetivo de validar o uso da matemática intervalar no supercomputador vetorial. São feitas algumas considerações sobre o conjunto de rotinas que deverão compor esta biblioteca baseadas no estudo preliminar descrito pelo capítulo quatro. Estas rotinas abordam métodos intervalares de solução de sistemas lineares. Considera-se uma equação algébrica como um sistema linear de ordem um, ou seja, com uma só equação.

Através do estudo dos métodos intervalares e de seus algoritmos relatado no capítulo quatro, visualizou-se a necessidade de definir bibliotecas aplicativos que tratassem de capítulos da matemática numérica. A primeira destas bibliotecas científicas aplicativos é a que trata da solução de sistemas lineares, denominada *libselint.a* (*sel* de sistemas de equações lineares e *int* de intervalos). A criação de bibliotecas científicas independentes, justifica-se no fato de que estas devem ser incluídas nos programas aplicativos dos usuários.

Através deste estudo foram identificadas três tipos de metodologias, três abordagens dos métodos intervalares. Estas abordagens devem gerar, cada uma, um módulo da biblioteca. Estes módulos devem ser:

- Módulo *dirint*, incluindo métodos baseados em operações algébricas intervalares e propriedades intervalares. Também são conhecidos como métodos diretos intervalares;
- Módulo *refint*, incluindo métodos baseados em inclusões ou refinamentos intervalares da solução e do erro. Também podem ser chamados de métodos híbridos, uma vez que se pode utilizar a solução inicial calculada por métodos pontuais ou a matriz inversa pontual;
- Módulo *itrint*, incluindo métodos iterativos intervalares, conhecidos também como métodos de relaxação. Eles também se baseiam em inclusões monotônicas.

Além destes três módulos foi definido um quarto módulo, *equalg*, que deve conter rotinas do caso particular de sistema linear de ordem um, ou seja, resolução algébrica de equações por métodos intervalares, como as versões intervalares de Newton. Como existe uma grande variedade de versões intervalares para o método de Newton, como pode ser visto em [DIV91a], foi escolhida apenas uma versão para se validar o uso da matemática intervalar.

Para o uso da biblioteca *libselint.a*, também deve ser incluído no programa aplicativo do usuário o arquivo *inter.inc*, que contém as definições de intervalos reais e complexos, e a biblioteca *libavi.a*, uma vez que as rotinas necessárias de manipulação de



intervalos, vetores e matrizes de intervalos foram ali implementadas. Pode ser necessária ainda a inclusão da biblioteca *libsci.a*, pois algumas das rotinas do módulo aplic se utilizam de rotinas da biblioteca BLAS, incluída nesta biblioteca.

O completo desenvolvimento da biblioteca *libselint.a*, que consiste na implementação, documentação, verificação e validação das rotinas ultrapassa o objetivo desta pesquisa. Este trabalho está sendo proposto como atividade de pesquisa e desenvolvimento do Grupo de Matemática Computacional.

A seguir serão caracterizados os módulos, através de algumas das rotinas que deverão compô-los, algumas das quais, foram implementadas para serem utilizadas para validar o uso efetivo da matemática intervalar em supercomputadores vetoriais.

#### **5.4.1 Módulo *dirint***

O módulo *dirint* deverá ser constituído de rotinas que implementem o método de Eliminação de Gauss intervalar e o cálculo da matriz inversa. Alguns destes métodos foram descritos no capítulo quatro. Estes métodos são baseados em propriedades e operações algébricas intervalares.

#### **5.4.2 Módulo *refint***

O módulo *refint* deverá ser constituído de rotinas que implementem os métodos baseados em refinamento. Alguns destes métodos necessitam de uma solução aproximada inicial produzida por métodos pontuais ou da inversa pontual da matriz dos coeficientes. Adotou-se para o cálculo da inversa pontual o método de Gauss-Jordan e, para o cálculo de estimativa inicial o método de Eliminação de Gauss sem pivotamento. Esta escolha foi arbitrária, poderia ser qualquer outro método, por isso, eles foram implementados em subrotinas.

#### **5.4.3 Módulo *itrint***

O módulo *itrint*, deverá ser constituído de métodos iterativos intervalares conhecidos como métodos de relaxação, como o método de Gauss-Seidel Intervalar. A convergência destes métodos é definida pelo teorema de Ponto-Fixo de Brouwer (capítulo 2). Os métodos iterativos intervalares também são mais utilizados no caso de sistemas lineares, onde a matriz dos coeficientes é esparsa.

#### **5.4.4 Módulo *equalg***

Este módulo deverá conter rotinas que implementem as versões intervalares do Método de Newton para resolução de equações algébricas. Estas versões foram descritas em detalhes em [DIV91a].

## 5.5 Características e Potencialidades

A *libavi.a* é uma biblioteca de rotinas básicas de intervalos para o ambiente do supercomputador Cray. As padronizações e convenções de especificação e documentação seguem o padrão da *Cray Research Inc.*

O nome das funções genéricas e as chamadas destas funções são o resultado de uma combinação do tipo de dado da rotina com o de sua função. Esta convenção já era utilizada na biblioteca de rotinas matemáticas *libm.a*, onde funções reais são anotadas pelo nome usual matemático. Portanto, funções reais não têm prefixo, mas as demais possuem uma letra de prefixo seguidas de seu mnemônico. As funções inteiras são precedidas por I, funções em dupla precisão por D e funções complexas por C.

Os argumentos são dados nos seus tipos: reais (R), inteiros (I), complexos (Z), lógicos (L), booleanos (B) e em dupla precisão (D). Os resultados são fornecidos da mesma forma: R, I, Z, L, B e D.

A biblioteca *libavi.a* introduz novos tipos de dados, como intervalos de reais e complexos, vetores e matrizes de intervalos reais. Para tanto, é necessário que seja adotada uma letra-prefixo que identifique o tipo de dado. Optou-se em utilizar a convenção de tipo intervalo (*interval*) do Pascal-XSC, mas como a letra I está associada ao tipo de dado inteiro, será usado o prefixo S (S de segmento, usado para intervalos na linguagem Fortran-SC) para intervalos; SC para intervalos complexos; SV para vetores de intervalos reais e SM para matrizes de intervalos reais.

As rotinas implementadas na *libavi.a* são compostas por três tipos de rotinas, elas podem ser subrotinas, funções ou operadores. As rotinas necessitam ser chamadas pelo comando *call* do Fortran que exige que seja escrito em uma linha do programa a chamada da rotina, o nome da rotina e seus devidos argumentos. As funções geralmente necessitam dos argumentos, mas o resultado é atribuído a uma variável do tipo pré-definido e elas podem ser incluídas em expressões matemáticas e serão avaliadas em primeiro lugar. O terceiro tipo de rotinas são os operadores, que são uma espécie de função que é chamada por um operador aritmético ou relacional. Este tipo de rotina foi sobrecarregado, ou seja, todas as funções de soma podem ser acessadas pelo sinal de +, as multiplicações pelo sinal \* e assim por diante. O tipo de dados que compõem a expressão são usados pelo compilador para determinar qual a rotina a ser utilizada. O tipo de dado resultante, geralmente é o tipo mais elaborado envolvido (um intervalo, por exemplo).

A biblioteca de rotinas intervalares *libavi.a* contém rotinas que podem ser acessadas por programas em linguagem Fortran. Ela é o conjunto de quatro módulos com vários conjuntos de rotinas. Os módulos são: *básico*, que contém o tipo de dado intervalo real com suas operações e funções; *ci*, que introduz intervalo complexo com suas operações e funções; *mvi*, que introduz vetores e matrizes de intervalos reais com as operações e funções que os manipulam; e o módulo *aplic* que contém operações

compostas entre reais, intervalos, vetores e matrizes reais. Entre estas rotinas estão as que correspondem aos três níveis da biblioteca BLAS.

Para se testar a biblioteca de rotinas intervalar, foram desenvolvidos alguns programas que implementam a resolução de sistemas de equações lineares por métodos intervalares. Estes programas foram agrupados, gerando a proposta de biblioteca denominada de *libselint.a* (biblioteca de solução de sistemas de equações lineares por intervalos).

Como o propósito do desenvolvimento desta biblioteca foi o de providenciar uma ferramenta útil para a programação científica (em ambiente vetorial), as rotinas implementadas necessitavam de certas características como: exatidão, eficiência, confiabilidade, validação, facilidade de uso, boa documentação, modificabilidade, completude e transportabilidade.

A exatidão destas rotinas foi obtida pelo uso das rotinas disponíveis no Fortran 90 para manipulação de números reais (números em ponto-flutuante), especialmente as funções básicas elementares. Para obter melhor exatidão, utilizaram-se algumas rotinas em dupla precisão, especialmente para tentar se ter um produto escalar mais próximo do ótimo. Utilizaram-se, ainda, algumas das rotinas da biblioteca BLAS para a implementação das operações entre vetores e matrizes de intervalos. A extensão para rotinas intervalares se deu através da aplicação de princípios de conversão real-intervalo e das propriedades de semimorfismo. Manteve-se a exatidão disponível do supercomputador Cray para processamento científico, mas se acrescentou, com intervalos, a garantia dos resultados.

Em relação à eficiência, evoluiu-se mais neste conceito, pois inicialmente eficiência significava um uso otimizado de memória, por ser muito cara e escassa. Com a evolução tecnológica, eficiência se vinculou à idéia de velocidade de processamento. Perdeu-se um pouco da idéia de qualidade para se ter velocidade. Com a matemática intervalar (acrescida de aritmética de alta exatidão) é resgatado o conceito de eficiência como qualidade e garantia do resultado calculado com rapidez. A biblioteca *libavi.a* produz resultados um pouco mais lentos do que a aritmética de ponto-flutuante ordinária, uma vez que os cálculos são efetuados para os extremos dos intervalos, mas ganha-se na garantia dos resultados.

As qualidades de confiabilidade e validação dizem respeito ao fato de a biblioteca ser consistente com a documentação, ou seja, a execução do programa realiza exatamente o que seu propósito descrito na documentação define, resolvendo a classe de problemas a que se propõe.

### 5.5.1 Padrão Cray de documentação

Uma boa documentação deve ser clara, concisa, fácil de entender, explanando como funciona e como usar. Isto é útil ao usuário para decidir sobre o uso ou não da rotina ou programa e, também, para diagnosticar as dificuldades provenientes do uso. Deve ser capaz de satisfazer a usuários que se utilizam esporadicamente da documentação para resolver pequenas dificuldades.

A documentação da biblioteca de rotinas intevalares é constituída do manual de utilização da *libavi.a*. Ele é constituído por duas partes, a primeira contém informações sobre o uso da biblioteca e a outra parte contém a documentação das rotinas. A primeira parte visa atender um usuário que deseja maiores informações da biblioteca, enquanto que a segunda parte visa atender ao uso esporádico.

Na documentação das rotinas da biblioteca *libavi.a* será usado o mesmo formato usado na documentação das bibliotecas de rotinas da *Cray Research Inc.* O objetivo desta padronização, de acordo com as normas da Cray, é anexar futuramente estas rotinas à biblioteca de rotinas matemáticas já à disposição. Logo, a documentação das bibliotecas *libavi.a* e *libselint.a* seguiram o padrão encontrado nas bibliotecas *libm.a* e *libsci.a* da Cray. Este padrão estabelece que a biblioteca seja dividida em módulos, que são compostos por rotinas de um mesmo grupo.

A documentação de cada módulo inicia pela descrição do módulo, dando seu objetivo. A seguir são apresentadas em tabelas todas as rotinas que compõem o módulo. As tabelas contêm as informações do tipo: propósito, nome da rotina e o mnemônico chave para localizar a documentação específica da rotina. Foi incluído ainda o número da página da documentação detalhada de cada rotina, assim o uso esporádico é facilitado.

Para cada rotina é apresentada uma documentação específica, denominada de documentação detalhada da rotina. O formato de documentação detalhada das rotinas é o descrito na figura 5.6. Muitos dos campos são opcionais, devendo ser usados na documentação de algumas rotinas que contenham particularidades. As únicas seções obrigatórias são NOME, SINOPSE e DESCRIÇÃO.

As documentações específicas, são organizadas na ordem alfabética. Para os nomes das rotinas foram adotados mnemônicos que lembram a função real ou matemática, antecedidas por prefixos que identificam o tipo de dado envolvido. Foram adotados, para os tipos de dados já existentes em Fortran 90, os mesmos prefixos, ou seja: *r* para dados reais, *d* para dupla precisão, *i* para inteiro, *c* para complexo e *l* para lógico. Para os novos tipos de dados, adotaram-se: *s* para intervalo, *sv* para vetor de intervalo, *sm* para matriz de intervalo e *sc* para intervalo complexo.



<b>NOME</b>	Contém o nome da rotina
<b>SINÔNIMO</b>	Mnemônico ou símbolo identificador que permite o acesso à rotina.
<b>SINOPSE</b>	Descreve a sintaxe das entradas. [] indica que o componente é opcional. Parêntesis indicam que o comando pode ser repetido
<b>DESCRIÇÃO</b>	Descreve a rotina com detalhes
<b>IMPLEMENTAÇÃO</b>	Contém a listagem e detalhes para o uso da rotina em máquinas ou sistemas operacionais específicos
<b>STANDARDS</b>	Notas relevantes a respeito da rotina que está sendo descrita, como o tipo de rotina, que pode ser subrotina, função ou operador.
<b>NOTAS</b>	Pontos ou itens de particular importância
<b>CUIDADOS</b>	Descreve ações que podem destruir dados ou resultados da rotina
<b>WARNINGS</b>	Descreve ações que podem danificar arquivos, equipamento ou sistema operacional
<b>EXEMPLOS</b>	Contém exemplos de uso
<b>ARQUIVOS</b>	Lista os arquivos que cada parte da rotina usa
<b>MENSAGENS</b>	Descreve informações, diagnósticos e mensagens e erro que podem aparecer.
<b>VER TAMBÉM</b>	Lista outras rotinas que tenham ligação com a rotina que está sendo descrita.

Fig 5.6 Formato da documentação detalhada das rotinas

O item *sinônimo* contém o identificador da rotina sobrecarregado, ou seja, o sinal que pode acessar a função ou o operador. Todos os tipos de soma podem ser representados por +, as multiplicações por \*, subtrações por - e divisões por /. As tabelas de sobrecarga de operadores serão fornecidas mais adiante.

Na *descrição* é definida a operação realizada, não foram colocados detalhes excessivos, para não confundir o usuário. É informado o que a rotina realiza. Detalhes de como é realizada a operação podem ser encontrados na bibliografia ou nos capítulos iniciais desta pesquisa.

No item *implementação* é dada a listagem da rotina tal como foi implementada. Como muitas vezes uma rotina é implementada através de outra, ou se utiliza de outra, aconselha-se no item *ver também* as demais rotinas relacionadas a ela.

Como a leitura de intervalos (reais e complexos, de vetores e matrizes) são realizadas por rotinas que lêem de arquivos os dados, no item *arquivos* podem-se encontrar referências a estes arquivos de dados, explicando como criá-los e acessá-los. Pode haver também arquivos que devem ser incluídos, pois a rotina se utiliza de outras funções ou rotinas de outras bibliotecas, como por exemplo da BLAS..

As mensagens de erro, advertências ou valores especiais retornados em caso de algum erro são especificadas no item *mensagens*. Não foram incluídas as mensagens de advertência básica de intervalos incorretos ou de não ser um intervalo. Esta informação por ser comum a um grande número de rotinas foi colocada nas rotinas de verificação *ck\_int* e *check\_int*.

*Exemplos* são fornecidos para ilustrar o uso da rotina. Neles é considerado apenas o uso específico da rotina. O domínio de uso da rotina ou situações especiais como zero não podem pertencer ao intervalo e são identificados nos itens *notas* e *cuidados*.

Maiores informações sobre a documentação pode ser obtida no Manual de Utilização da Biblioteca de Rotinas Intervalares *libavi.a* ([DIV95c]).

### 5.5.2 Facilidade de uso

As bibliotecas *libavi.a* e *libselint.a* são entidades do Fortran 90 denominada módulo. Para se ter acesso a estes módulos é necessário que os arquivos que os contêm, sejam incluídos no programa.

Há três formas de se incluir em arquivos contendo bibliotecas de rotinas. Estas formas estão relacionadas ao tipo de arquivo. Em geral, pode-se incluir uma biblioteca no programa pelo uso de uma das opções na linha de compilação ou, pela inclusão do arquivo dentro do programa.

A inclusão ou o acesso do arquivo através das opções na linha de compilação, ou seja, na linha de comando do f90, pode ser feita por duas das opções: *-l* ou *-p*. A opção *-l <arq.a>*, mais conhecida e utilizada torna acessíveis as rotinas contidas no arquivo do tipo biblioteca (*.a*) que segue a opção. O arquivo tem que ser um *arq.a*. A opção *-p <arq.o>* inclui ou torna acessível um arquivo objeto ao programa Fortran. Portanto, a linha de comando do compilador Fortran 90 é da forma: *f90 -p libavi.a -l libsci.a program.f90*.

A inclusão no programa é feita pelo comando *include 'arq.f90'*, ou seja, inclui um arquivo em linguagem fonte (Fortran 90) ao programa. Portanto, esta forma exige que o usuário possua o código fonte.

Qualquer uma destas formas requer ainda, que no programa seja especificado o módulo que será utilizado no programa. Esta especificação é feita através do comando *use <módulo>*, como já foi tratado na seção 3.2.2.4. Observa-se somente que, deve-se utilizar o comando *use* para cada módulo utilizado, ou seja, se for usado em um programas os módulos *básico* e *mvi*, deve ser incluídos dois comandos *use*, *use basico* e, em outra linha, *use mvi*.

A definição dos tipos de dados intervalos reais e complexos foi separada em um arquivo especial, denominado *inter.inc*. A definição foi isolada, por ser necessário sua inclusão não só em cada módulo, como também, no corpo de cada rotina que se utiliza deste tipo de dado. Assim, basta incluir o arquivo através do comando *include 'inter.inc'*.

Uma característica, decorrente do tipo de processamento do supercomputador Cray, é a leitura de dados somente de arquivos. No Cray não se deve ter processamento interativo, uma vez que a máquina é voltada para alto desempenho, com grande velocidade de processamento, especialmente em cálculos em ponto-flutuante. Adotou-se que intervalos podem ser lidos de arquivos ou atribuídos no programa, para tanto foram implementadas as funções de transferência. A impressão de intervalos também deve ser feita em arquivos. Portanto, para resumir, as rotinas *read* e *write* trabalham com arquivos. O conteúdo dos arquivos são as informações que devem ser lidas. Isto facilita a introdução de vetores e matrizes de intervalos de alta ordem. Vetores e matrizes de intervalos são lidos por colunas, a cada linha do arquivo é lido ou impresso um intervalo (extremo inferior e superior). As rotinas de leitura são especificados em detalhes na seção 5.3.5.

Tabela 5.30 Sobrecarga dos operadores = e +

Símbolo	Rotinas	Argumentos	Módulo
=	scopy	s1, s2	Básico
	sccopy	sc1, sc2	CI

Símbolo	Rotinas	Argumentos	Módulo
+	sadd	s1, s2	Básico
	smt	s1	Básico
	svadd	sv1, sv2	MVI
	svmt	sv1	MVI
	smadd	sm1, sm2	MVI
	smmt	sm1	MVI
	scadd	sc1, sc2	CI
	scmt	sc1	CI
	radds	r1, s1	MVI
	saddr	s1, r1	MVI
	raddsv	r1, sv1	MVI
	svaddr	sv1, r1	MVI
	raddsm	r1, sm1	MVI
	smaddr	sm1, r1	MVI
	saddsv	s1, sv1	MVI
	svadds	sv1, s1	MVI
	saddsm	s1, sm1	MVI
	smadds	sm1, s	MVI
	rvaddsv	rv1, sv1	MVI
	svaddrv	sv1, rv1	MVI
rmaddsm	rm1, sm1	MVI	
smaddrm	sm1, rm1	MVI	

Sobrecarga de funções e operadores é uma das facilidades da biblioteca de rotinas intervalares. A sobrecarga possibilita que o usuário utilize o nome genérico de uma função, independente do tipo de argumentos desta função. A tarefa de identificar a rotina correta que opera com tais argumentos é transferida ao compilador. Isto facilita a programação. Foram sobrecarregados os símbolos das operações aritméticas de soma,

subtração, multiplicação, divisão e os símbolos relacionais existentes no Fortran 90. As demais funções matemáticas foram sobrecarregadas ao nome usual da notação matemática, como mostram as tabelas 5.30 a 5.35.

Tabela 5.31 Sobrecarga dos operadores aritméticos -, \* e /

Símbolo	Rotinas	Argumentos	Módulo
-	ssub	s1, s2	Básico
	svsub	sv1, sv2	MVI
	smsub	sm1, sm2	MVI
	smneg	sm1, sm2	MVI
	scsub	sc1, sc2	CI
	scneg	sc1	CI
	rsubs	r1, s1	MVI
	ssubr	s1, r1	MVI
	rsubsv	r1, sv1	MVI
	svsubr	sv1, r1	MVI
	rsubsm	r1, sm1	MVI
	smsubr	sm1, r1	MVI
	rsubsv	s1, sv1	MVI
	svsubs	sv1, s1	MVI
	ssubsm	s1, sm1	MVI
	smsubs	sm1, s1	MVI
	svsubrv	sv1, rv1	MVI
	smsubrm	sm1, rm1	MVI

Símbolo	Rotinas	Argumentos	Módulo
*	smult	s1, s2	Básico
	svmult	sv1, sv2	MVI
	smmult	sm1, sm2	MVI
	scmult	sc1, sc2	CI
	rmults	r1, s1	MVI
	smultr	s1, r1	MVI
	rmultsv	r1, sv1	MVI
	svmultr	sv1, r1	MVI
	rmultsm	r1, sm1	MVI
	smmultr	sm1, r1	MVI
	smultsv	s1, sv1	MVI
	svmults	sv1, s1	MVI
	smultsm	s1, sm1	MVI
	smmults	sm1, s1	MVI
	smmultsv	sm1, sv1	MVI
	svmultsm	sv1, sm1	MVI
	rvmultsv	rv1, sv1	MVI
	svmultrv	sv1, rv1	MVI
	smmultrm	sm1, rm1	MVI
	rmultsm	rm1, sm1	MVI
smmultrv	sm1, rv1	MVI	
rmultsv	rm1, sv1	MVI	

Símbolo	Rotinas	Argumentos	Módulo
/	sdiv	s1, s2	Básico
	scdiv	sc1, sc2	CI
	rdivs	r1, s1	MVI
	sdivr	s1, r1	MVI
	svdivr	sv1, r1	MVI
	smdivr	sm1, r1	MVI
	svdivs	sv1, s1	MVI
smdivs	sm1, s1	MVI	



Tabela 5.32 Sobrecarga dos operadores relacionais

Símbolo	Rotinas	Argumentos	Módulo
<b>==</b>	sequ svequ smequ scequ	s1, s2 sv1, sv2 sm1, sm2 sc1, sc2	Básico MVI MVI CI
<b>!=</b>	sneq sveq smeq sneq	s1, s2 sv1, sv2 sm1, sm2 sc1, sc2	Básico MVI MVI CI
<b>&gt;</b>	sgre svgre smgre scre	s1, s2 sv1, sv2 sm1, sm2 sc1, sc2	Básico MVI MVI CI
<b>&lt;</b>	sless svless smless scless	s1, s2 sv1, sv2 sm1, sm2 sc1, sc2	Básico MVI MVI CI
<b>&gt;=</b>	sgeq svgeq smgeq screq	s1, s2 sv1, sv2 sm1, sm2 sc1, sc2	Básico MVI MVI CI
<b>&lt;=</b>	sleq svleq smleq screq	s1, s2 sv1, sv2 sm1, sm2 sc1, sc2	Básico MVI MVI CI

Tabela 5.33 Sobrecarga das funções trigonométricas

Símbolo	Rotinas	Argumentos	Módulo
<b>sin</b>	ssin svsin smisin scsin	s1 sv1 sm1 sc1	Básico MVI MVI CI
<b>cos</b>	scos svcos smcos ccos	s1 sv1 sm1 sc1	Básico MVI MVI CI
<b>tan</b>	stan svtan smtan ctan	s1 sv1 sm1 sc1	Básico MVI MVI CI
<b>arctan</b>	saretan svaretan smaretan scaretan	s1 sv1 sm1 sc1	Básico MVI MVI CI

Tabela 5.34 Sobrecarga das funções exponenciais e logarítmicas

Símbolo	Rotinas	Argumentos	Módulo
<b>exp</b>	sexp svexp smexp scexp	s1 sv1 sm1 sc1	Básico MVI MVI CI
<b>ln</b>	sln svln smln scln	s1 sv1 sm1 sc1	Básico MVI MVI CI
<b>log</b>	slog svlog smlog sclog	s1 sv1 sm1 sc1	Básico MVI MVI CI
<b>sqr</b>	ssqr svsqr smisqr scsqr	s1 sv1 sm1 sc1	Básico MVI MVI CI
<b>sqrt</b>	ssqrt svsqrt smisqrt scsqrt	s1 sv1 sm1 sc1	Básico MVI MVI CI

Tabela 5.35 Sobrecarga das demais funções

Símbolo	Rotinas	Argumentos	Módulo
<b>null</b>	smnullm	sm1	MVI
	smnulld	i1, i2	MVI
	svnullv	sv1	MVI
	svnulld	i1	MVI
<b>smid</b>	smidm	sm1	MVI
	smidd	i1, i2	MVI
<b>svum</b>	svumv	sv1	MVI
	svumd	i1	MVI

Os uso de arrays dinâmicos na definição de vetores e matrizes de intervalos se constitui em outra característica da linguagem Fortran, que foi incorporada às características da biblioteca de rotinas intervalares. Ela contribui para a simplificação e facilidade de uso da biblioteca. *Arrays* dinâmicos permitem não só um uso otimizado da memória, através do uso de vetores e matrizes de tamanho aleatório, garantindo alocação e liberação do espaço da memória, mas também facilitam a programação uma vez que o usuário apenas utiliza a sua definição segundo a compatibilidade do sistema. O sistema não impõem restrições desnecessárias e nem exige conhecimento prévio de limites. Quando o usuário declara um vetor ou matriz de intervalos no seu programa aplicativo é que o tamanho ou o números de elementos é definido. Toda a referência a este vetor ou matriz de intervalos, leva consigo o tamanho e o tipo dos elementos, inclusive no momento de leitura e impressão de vetores e matrizes intervalares. Não há restrições de tamanho de vetores e matrizes no módulo *mvi* da biblioteca.

### 5.5.3 Rotinas otimizadas e vetorizadas

A otimização e vetorização das rotinas da *libavi.a* se deram como que por herança das características da máquina, do compilador e das rotinas que são utilizadas das bibliotecas, como as rotinas da BLAS. O uso destas rotinas não só proporcionou esta vantagem, mas também proporcionou a mesma qualidade em termos de exatidão dos resultados, adicionado ao uso de intervalos, o que proporciona resultados contidos dentro do intervalo solução.

Cuidou-se, também, na definição dos novos tipos de dados a questão do armazenamento seqüencial dos componentes dos intervalos, dos elementos dos vetores de intervalos e dos elementos das matrizes, de forma a evitar conflitos no acesso aos bancos de memória.

### 5.5.4 Biblioteca Modular

Modularidade é outra característica da biblioteca de rotinas intervalares. A modularidade é aplicada no sentido de módulos independentes e no sentido do compilador Fortran. Em cada módulo são definidas as operações que manipulam com certo tipo de elementos ou dados. Isto facilita o entendimento, a manutenção e produz maior estruturação na biblioteca. Estas rotinas são sobrecarregadas através do conceito de operador genérico através das interfaces, possibilitando ao usuário a utilização de

operações pelo símbolo matemático, sem se preocupar com o tipo de dados dos argumentos, uma vez que esta tarefa é transferida ao compilador. Isto facilita a programação e o usuário, que não necessita conhecer, ou melhor, memorizar todas as diferentes versões de uma determinada rotina ou função, pode acessá-la pelo símbolo matemático ou por sua notação usual.

Os módulos facilitam a organização da biblioteca, o uso e a própria manutenção, pois rotinas afins são agrupadas e, quando for identificado um problema, este estará automaticamente localizado dentro dos domínios do módulo que manipula determinado dado.

A modularidade também é importante para garantir a transportabilidade da biblioteca, pois uma vez que existam partes dependentes de máquina que necessitam ser adaptadas ou modificadas ao novo ambiente, elas devem ser isoladas, organizadas e documentadas de maneira a ser possível esta modificação.

### 5.5.5 Abrangência da biblioteca

A abrangência da biblioteca de rotinas básicas intervalares é, inicialmente, o espaço dos intervalos reais, dos vetores intervalos de reais, das matrizes de intervalos reais e dos intervalos complexos. A extensão a vetores e matrizes de intervalos complexos pode ser facilmente obtida. A definição destes tipos já está especificada no módulo *ci*. As operações são estendíveis componente a componente, de forma análoga ao que foi feito para vetores e matrizes de intervalos reais.

Considerando os módulos de vetores e matrizes e o módulo de aplicações, pode-se dizer que a biblioteca de rotinas *libavi.a* contém um conjunto amplo de rotinas intervalares para álgebra linear. Os capítulos de resolução de equações e solução de sistemas de equações lineares deverão ser abrangidos pela biblioteca de rotinas intervalares aplicativa *libselint.a*. Outras bibliotecas aplicativas podem ser especificadas e desenvolvidas a parte desta biblioteca básica. Podem-se resolver vários tipos de problemas, como é demonstrado em [HAM93].

Outra característica relacionada aqui é a completude desta biblioteca, sendo um indicador da extensão da biblioteca. Métodos numéricos envolvem parâmetros, os quais podem assumir valores determinados de acordo com as propriedades básicas do problema particular. A análise do processo é geralmente baseada no comportamento do problema modelo. O comportamento da maioria dos problemas não diverge, na prática, do esperado pelo modelo. Mas existem problemas em que a solução se deteriora, afastando-se do modelo. A extensão deste afastamento é uma medida de completude, da potencialidade do programa. Ela também se refere à habilidade de contornar dificuldades. Esta propriedade no caso de intervalos se reflete no diâmetro do intervalo solução.

## 5.6 Conclusões

Neste capítulo foi detalhada a biblioteca de rotinas intervalares *libavi.a*, composta de 291 rotinas, divididas em quatro módulos. A biblioteca de rotinas intervalares torna disponível a aritmética intervalar básica no Cray, bem como cálculos com vetores e matrizes de intervalos reais. Também estão disponíveis operações e funções com intervalos complexos. As rotinas que manipulam vetores e matrizes de intervalos complexos podem ser facilmente implementadas, através da aplicação das operações entre intervalos complexos em cada componente do vetor ou da matriz.

Está aberto um amplo campo de aplicações da matemática intervalar, pois possibilita a usuários desenvolverem programas para resolver seus próprios problemas nas mais diversas áreas, tendo resultados entre limites garantidos, dentro do intervalo solução.

As rotinas *infla\_down* e *infla\_up* proporcionam o controle de arredondamento nas rotinas numéricas da biblioteca, proporcionando que os resultados das operações estejam entre limites confiáveis. Outra garantia para verificar a consistência do resultado das operações intervalares, são as rotinas *check\_int* e *ck\_int* que verificam se os parâmetros são intervalos.

No próximo capítulo será feita uma comparação da biblioteca *libavi.a* com outras bibliotecas científicas, o objetivo será comparar o domínio de problemas que ela é capaz de resolver com o de outras bibliotecas existentes. No capítulo sete, será feita a verificação da biblioteca, na tentativa de não só avaliar a qualidade numérica mas o tipo de problema que se pode resolver através dela.

No capítulo final são comentados os resultados obtidos com esta ferramenta, as limitações do trabalho e sugestões de melhorias que podem ser feitas a biblioteca de rotinas intervalares.

No anexo 10.4 é apresentado o extrato da listagem dos programas que compõem a biblioteca. Eles totalizam cerca de 4900 linhas de programação. As rotinas foram testadas e documentadas. A documentação encontra-se em [DIV95b] e os testes em [DIV95c].



## 6 COMPARAÇÃO DA *libavi.a* COM OUTRAS BIBLIOTECAS

Neste capítulo são analisadas algumas bibliotecas de rotinas e extensões de linguagens com o objetivo de compará-las, em sua amplitude, com a biblioteca de rotinas intervalares *libavi.a*, descrita no capítulo anterior.

Para cada biblioteca ou linguagem são descritos os tipos de dados ou rotinas e, por fim, são caracterizadas as vantagens e/ou desvantagens de cada uma destas bibliotecas em relação à *libavi.a*.

Este estudo auxiliou na determinação das rotinas que vieram a compor a biblioteca de rotinas intervalares.

### 6.1 A *Numerals* da Burroughs

*NUMERALS* significa *Numerical Analysis Program Library* e é uma coleção de procedimentos em Algol e rotinas em Fortran, disponível em sistemas de computadores da Burroughs, que abrange diversos tópicos na área da matemática numérica. Entre estes, salienta-se o da álgebra linear.

Em 1982 este pacote foi estudado e documentado pelo Grupo de Matemática Computacional da UFRGS. Neste estudo, trinta e nove rotinas deste sistema foram descritas. Elas abordavam o capítulo de álgebra linear e solução de sistemas de equações.

A biblioteca deveria ser incorporada a programas de usuários através de comandos de inclusão de rotinas, do tipo *include mathlib*, onde *mathlib* é o módulo em questão. O nome da rotina é um mnemônico composto que descreve a utilidade da rotina. A convenção utilizada é dada pela figura 6.1 e as rotinas são descritas pelas tabelas 6.1 e 6.2. As matrizes diagonais são armazenadas em vetores.

VEC	vetor	TIMES	vezes	DET	determinante
MAT	matriz	PLUS	mais	LIN	linear
DIAG	diagonal	MINUS	menos	EQN	equações
REC	retangular	WITH	com	IMPR	refinamento
BAND	banda	OVER	sobre	CX	complexa
R	escalar	OF	de	TRANSPOSE	transposta
SYM	simétrica	'S'	plural	INVERSE	inversa

Fig 6.1 Convenção utilizada na composição dos mnemônicos

### 6.1.1 Módulo *mathlib*

Entre o conjunto das rotinas analisadas foram verificadas rotinas que operam entre matrizes, entre matrizes e vetores, entre vetores e escalares e outras operações. Em relação a matrizes estas podem ser: quadradas ( $n \times n$ ), retangulares ( $n \times m$ ), diagonais e bandas. Há vários métodos para a resolução de sistemas de equações lineares, como o método de eliminação de Gauss com pivotamento parcial - com ou sem refinamento; método de Cholesky para matrizes simétricas definidas e positivas - com ou sem refinamento; métodos de solução de sistemas sobre-determinados e métodos para resolução de sistemas complexos.

Tabela 6.1 Rotinas entre matrizes da *Numerals*

NOME	REPRES. MATCA	DESCRIÇÃO
MATPLUSMAT	$C[1:N,1:N] = A[1:N,1:N] + B[1:N,1:N]$	Soma de matrizes quadradas
MATMINUSMAT	$C[1:N,1:N] = A[1:N,1:N] - B[1:N,1:N]$	Subtração de matrizes quadradas
MATTIMSMAT	$C[1:N,1:N] = A[1:N,1:N] * B[1:N,1:N]$	Multiplicação de matrizes quadradas
TRANPOSE	$C[1:N,1:N] = A[1:N,1:N]^t$	Cálculo da matriz transposta
ATIMESATRANPOSE	$C[1:N,1:N] = A[1:N,1:N] * A[1:N,1:N]^t$	Multiplicação de A pela sua transposta
ATIMESBTRANPOSE	$C[1:N,1:N] = A[1:N,1:N] * B[1:N,1:N]^t$	Multiplicação de A pela transposta de B
MATTIMSEVEC	$Y[1:N] = A[1:N,1:N] * X[1:N]$	Multiplicação de matriz quadrada por vetor
VECTIMSMAT	$Y[1:N] = X[1:N] * A[1:N,1:N]$	Multiplicação de vetor por matriz quadrada
VECPUSRVEC	$Y[1:N] = X[1:N] + r * Z[1:N]$	Soma de vetor com o produto de real por vetor
MATPLUSDIAG	$C[1:N,1:N] = A[1:N,1:N] + D[1:N]$	Soma de matriz com matriz diagonal
MATMINUSDIAG	$C[1:N,1:N] = A[1:N,1:N] - D[1:N]$	Subtração de matriz com matriz diagonal
MATTIMSDIAG	$C[1:N,1:N] = A[1:N,1:N] * D[1:N]$	Multiplicação de matriz por matriz diagonal
DIAGTIMSMAT	$C[1:N,1:N] = D[1:N] * A[1:N,1:N]$	Multiplicação de matriz diagonal por matriz
RPLUSMAT	$C[1:N,1:N] = r + A[1:N,1:N]$	Multiplicação de real por matriz
RMINUSMAT	$C[1:N,1:N] = r - A[1:N,1:N]$	Subtração de real com matriz
RTIMSMAT	$C[1:N,1:N] = r * A[1:N,1:N]$	Multiplicação de real por matriz
MATCOPY	$C[1:N,1:N] = A[1:N,1:N]$	Cópia da matriz
RECMATPLUSMAT	$C[1:N,1:M] = A[1:N,1:M] + B[1:N,1:M]$	Soma de matrizes retangulares
RECMATMINUSMAT	$C[1:N,1:M] = A[1:N,1:M] - B[1:N,1:M]$	Subtração de matrizes retangulares
RECMATTIMSMAT	$C[1:N,1:M] = A[1:N,1:K] + B[1:K,1:M]$	Multiplicação de matrizes genéricas

### 6.1.2 Comparação da *libavi.a* com *Numerals*

A comparação feita neste item só diz respeito ao conjunto de operações disponíveis pelas bibliotecas, não está sendo considerada a exatidão produzida nos resultados e nem o tempo de execução para resolver os problemas. O objetivo desta comparação foi inicialmente identificar novas rotinas para serem incorporadas ao módulo *aplic* da *libavi.a*.

A primeira comparação é que a biblioteca *Numerals* não tem uma interface amigável e não proporciona facilidades de uso, uma vez que não tem a si incorporada a sobrecarga de identificadores e símbolos. Não se pode utilizar a notação matemática. Devem ser utilizados os mnemônicos das rotinas com seus respectivos argumentos. A biblioteca *libavi.a* facilita seu uso, facultando ao usuário a utilização de símbolos e operadores.

O conjunto de rotinas que operam com matrizes na *Numerals* está contido no conjunto de rotinas da *libavi.a*; observa-se que as operações de matriz com matriz diagonal, disponíveis na *Numerals*, na verdade são aplicações das rotinas que operam matrizes com vetores, pois a matriz diagonal é armazenada na forma de um vetor.

No conjunto de métodos pontuais para resolução de sistemas, a *Numerals* possui rotinas que operam com sistemas sobre-determinados e a *libavi.a* só trata com sistemas  $n \times n$ .

Por fim, observa-se que os resultados da *Numerals* estão sujeitos aos erros de arredondamento e a instabilidade numérica, o que, dependendo do problema, pode levar a situações inesperadas, portanto não se tem garantia do resultado. Já o resultado produzido pela *libavi.a* contém o valor exato, ou seja, está entre limites confiáveis.

Tabela 6.2 Rotinas de resolução de sistemas da *Numerals*

NOME	DESCRIÇÃO
LINEQN	Resolve sistema por eliminação com pivotamento parcial
DETLINEQN	Resolve sistema por eliminação com pivotamento parcial e calcula o determinante
LINEQNIMPR	Efetua um refinamento na solução
DETLINEQNIMPR	Efetua refinamentos e calcula o determinante
LINEQNS	Resolve sistemas por eliminação com pivotamento parcial
DETLINEQNS	Resolve sistemas por eliminação com pivotamento parcial e calcula o determinante
INVERSE	Calcula a matriz inversa e o determinante
INVWTHIMPRV	Calcula a matriz inversa refinada e o determinante
DETOFA	Calcula o determinante
INVERSEOVERA	Calcula a inversa de A, armazenando-a na própria matriz A
LEASTSQUARES	Resolve sistemas sobre-determinados (Matriz A é retangular)
LEASTSQUARESIMPRV	Resolve sistemas sobre-determinados com refinamentos (Matriz A é retangular)
SYMLINEQN	Resolve sistema simétrico por Cholesky
SYMLINEQNIMPRV	Resolve sistema simétrico por Cholesky com refinamentos
SYMLINEQNS	Resolve sistemas simétricos por Cholesky
SYMBANDLINEQN	Resolve sistemas por método iterativo. A é esparsa do tipo Banda.
SYMINVWTHIMPRV	Calcula a inversa de A com refinamento. A é simétrica, definida positiva
SYMINVERSEOVERA	Calcula a inversa de A armazenando sobre A. A é simétrica, definida positiva
CXLINEQN	Resolve sistema complexo

## 6.2 A Biblioteca científica da Cray *libsci.a*

Existem várias bibliotecas disponíveis no Cray, entre estas estão a *libc.a* (biblioteca de rotinas da linguagem C), *libu.a* (biblioteca de programas utilitários), *libm.a* (biblioteca de rotinas matemáticas), *libio.a* (biblioteca de rotinas de E/S), *libp.a* (biblioteca de rotinas da linguagem pascal) e a *libsci.a* (biblioteca de rotinas científicas).

As bibliotecas são referenciadas pela opção *-l* ou pela opção *-p* na linha de comando da compilação. As funções intrínsecas Fortran são incluídas no tempo de compilação, produzindo um módulo objeto com referências a rotinas externas. No momento da carga, as referências externas são resolvidas quando o carregador procurar o sistema para as rotinas referenciadas no programa.

As funções disponíveis geralmente operam para tipos de dados reais, dupla precisão e complexos, sendo o nome real o mnemônico da função. As versões em dupla precisão e complexa são acrescidas da letra *d* ou *c* na frente do mnemônico real.

A biblioteca de rotinas matemáticas contém funções aritméticas gerais, funções logarítmicas e exponenciais, funções trigonométricas, rotinas de conversão de tipos e rotinas de aritmética *booleana* (lógica).

As rotinas científicas foram agrupadas em dez conjuntos de rotinas. Elas tratam de álgebra linear, níveis 1, 2 e 3, conhecidas como BLAS1, BLAS2 e BLAS3; rotinas para resolver sistemas lineares densos (LAPACK e LINPACK); rotinas para resolver sistemas lineares especiais como tridiagonais; resolvidores de sistemas esparsos; autovalores; processamento de sinais (FFT); rotinas de classificação e pesquisa e, também, rotinas especiais com vetores.

A *libm.a* contém rotinas que podem ser acessadas por programas nas linguagens Fortran, C e Assembler. Ela é agrupada em cinco conjuntos de rotinas segundo o tipo de função que executa. Estes grupos são definições matemáticas gerais; funções exponenciais e logarítmicas; funções trigonométricas; funções de conversão de tipos e funções *booleanas*.

A maioria destas rotinas são plenamente vetorizadas. A geração do código de muitas delas é externa. Maiores detalhes sobre a biblioteca podem ser obtidos através de consulta ao manual on-line disponível através do comando *man* ou no manual da biblioteca Nro SR 2081 da Cray, ou ainda, no relatório de pesquisa Nro 210 ([DIV93]).



## 6.2.1 Módulos científicos

As rotinas científicas são agrupadas em dez conjuntos de rotinas. Muitas das rotinas científicas admitem processamento paralelo do tipo multitarefa. Isto significa que quando um programa chama uma das rotinas da libsci que admite multitarefa, será executado no modo paralelo com a vantagem de usar múltiplos processadores, mesmo que o programa não tenha solicitado o uso de multitarefas. Se grande parte do tempo de execução do programa for gasto com rotinas da libsci, implicará redução do tempo de execução. A seguir são enumerados os grupos de rotinas da biblioteca científica.

- a) **BLAS1 - Basic Linear Algebra Subprograms Level 1.** O nível 1 da BLAS realiza as operações vetor a vetor, com dados reais e complexos em precisão simples. Os tipos de operações que estão disponíveis são produtos escalares e normas de vetores, geração e aplicações planas ou rotações de planos, soma, cópia, troca e cálculo de combinações lineares de vetores. As combinações lineares são da forma  $Y \leftarrow rX+Y$ , onde  $X$  e  $Y$  são vetores e  $r$  um valor real.

Tabela 6.3 Rotinas reais da BLAS nível 1

BLAS		nível 1	rotinas reais
NOME	DESCRIÇÃO		FÓRMULA
SROTG	Construção de um dado plano de rotação		
SROTMG	Construção de modificação de um plano de rotação		
SROT	Aplicação de um plano ortogonal de rotação		
SROTM	Aplicação de uma modificação de um plano de rotação		
SSWAP	Troca de dois arrays reais		$X \leftrightarrow Y$
SSCAL	Cálculo de um vetor múltiplo escalar		$X = rX$
SCOPY	Cópia de um vetor para outro		$Y = X$
SAXPY	Soma de um múltiplo escalar do vetor a outro vetor		$Y = rX + Y$
SDOT	Cálculo de um produto escalar entre dois vetores		$\text{dot} = X^T \cdot Y$
SNRM2	Cálculo da norma euclidiana		$\ X\ _2$
SASUM	Soma dos valores absolutos dos elementos de um vetor		$\text{sum} = \sum  x_i $
SSUM	Soma dos valores dos elementos de um vetor		$\text{sum} = \sum x_i$
SPDOT	Cálculo de um produto escalar esparsos de dois vetores		$\text{dot} = X^T \cdot Y$
SPAXPY	Soma de um múltiplo escalar do vetor a outro vetor esparsos		$Y = rX + Y$

- b) **BLAS2 - Basic Linear Algebra Subprograms Level 2.** O nível 2 da BLAS consiste de rotinas *assembler* para dados reais e complexos. Ele garante operações de matrizes com vetores. Elas foram escritas de forma a garantir uma execução ótima nos sistemas Cray. As operações são da forma  $Y \leftarrow rAX+sY$ , onde  $X$  e  $Y$  são

vetores reais ou complexos,  $r$  e  $s$  valores reais ou complexos e  $A$  uma matriz real ou complexa. Ela pode ser densa, banda, Hermitiana, simétrica ou triangular.

Tabela 6.4 Rotinas reais da BLAS nível 2

BLAS		nível 2	rotinas reais
NOME	DESCRIÇÃO		
SGEMV	Multiplicação de um vetor por uma matriz geral $Y = rAX + sY$		
SGBMV	Multiplicação de um vetor por uma matriz banda $Y = rAX + sY$		
SSYMV	Multiplicação de um vetor por uma matriz simétrica $Y = rAX + sY$		
SSBMV	Multiplicação de um vetor por uma matriz banda simétrica $Y = rAX + sY$		
SSPMV	Multiplicação de um vetor por uma matriz simétrica compactada $Y = rAX + sY$		
STRMV	Multiplicação de um vetor por uma matriz triangular		
STBMV	Multiplicação de um vetor por uma matriz banda triangular		
STPMV	Multiplicação de um vetor por uma matriz triangular compactada		
STRSV	Resolução de um sistema de equações triangular		
STBSV	Resolução de um sistema de equações triangular esparso (tipo banda)		
STPSV	Resolução de um sistema de equações triangular compactado		
SGER	Calcula um rank 1 e atualiza a matriz $A = rXY^T + A$		
SSYR	Calcula um rank 1 e atualiza a matriz simétrica $A = rXX^T + A$		
SSPR	Calcula um rank 1 e atualiza a matriz compacta $A = rXX^T + A$		
SSYR2	Calcula um rank 2 e atualiza a matriz simétrica $A = rXY^T + rYX^T + A$		
SSPR2	Calcula um rank 2 e atualiza a matriz simétrica compactada $A = rXY^T + rYX^T + A$		
SSPR12	Calcula dois rank 1 simultâneos e atualiza a matriz simétrica compactada $A = rXX^T + sYY^T + A$		

- c) BLAS3 - *Basic Linear Algebra Subprograms Level 3*. O nível 3 da BLAS consiste de rotinas *assembler* para pacotes de dados reais ou complexos. Ela garante operações entre matrizes. O formato das operações é da forma  $C=rAB+sC$ , onde  $A$ ,  $B$  e  $C$  são matrizes e  $r$  e  $s$  valores reais ou complexos. As matrizes podem ser densas, Hermitianas, simétricas ou triangulares.

Tabela 6.5 Rotinas reais da BLAS nível 3

BLAS		nível 3	rotinas reais
NOME	DESCRIÇÃO		
SGEMM	Multiplicação de duas matrizes gerais $C = rAB + sC$		
SSYMM	Multiplicação de uma matriz por outra matriz simétrica		
SSYRK	Cálculo de um rank $k$ e atualização da matriz simétrica $C = rAA^T + sC$		
SSYR2K	Cálculo de um rank 2 e atualização da matriz simétrica $C = rAB^T + rBA^T + sC$		
STRMM	Multiplicação de uma matriz geral por uma matriz triangular		
STRSM	Resolução de um sistema triangular com múltiplos lados direitos		
SGEMMS	Multiplicação de matrizes gerais usando a variação do algoritmo de Strassen		

d) Rotinas de solução de sistemas densos de equações lineares são compostas pelas rotinas LAPACK e LINPACK. LAPACK é uma biblioteca de domínio público que resolve eficientemente problemas de álgebra linear (densos) em computadores de alto desempenho. Os objetivos do desempenho foram atingidos pela implementação de uma grande quantidade de algoritmos em termos dos níveis 2 e 3 da BLAS e pela incorporação dos avanços em algoritmos para cálculos de álgebra linear. As rotinas resolvem sistemas de equações usando fatoração LU e Cholesky para matrizes gerais, bandas, simétricas e definida positiva e matrizes bandas simétricas definida positiva. LINPACK é um pacote de rotinas Fortran para resolver sistemas de equações lineares e calcular decomposição QR, Cholesky, LU e valores singulares. Estas rotinas foram otimizadas pela substituição de rotinas BLAS por rotinas em código Fortran vetorizado. As operações podem ser de fatorar e estimar a condição, resolver, calcular determinante, decompor, atualizar, recuperar e trocar.

Tabela 6.6 Rotinas de resolução de sistemas lineares densos - LAPACK

LAPACK - sistemas lineares densos - rotinas reais	
NOME	DESCRIÇÃO
SGBCON	Estima o recíproco do número do condicionamento da matriz banda, usando a fatoração LU calculada pela SGBTRF
SGBTRF	Calcula a fatoração LU da matriz banda $m \times n$ , usando pivotamento parcial
SGBTRS	Resolve sistemas lineares $AX=B$ , onde A é uma matriz banda, cuja fatoração LU foi calculada por SGBTRF.
SGECON	Estima o recíproco do número do condicionamento da matriz genérica,, usando a fatoração LU calculada pela SGETRF
SGETRF	Calcula a fatoração LU da matriz genérica $m \times n$ , usando pivotamento parcial
SGETRI	Calcula a inversa da matriz, usando fatoração LU calculada por SGETRF
SGETRS	Resolve sistemas lineares $AX=B$ , onde A é uma matriz genérica, cuja fatoração LU foi calculada por SGETRF.
SPBCON	Estima o recíproco do número do condicionamento da matriz banda, simétrica definida positiva usando a fatoração de Cholesky calculada pela SPBTRF
SPBTRF	Calcula a fatoração de Cholesky de uma matriz banda simétrica definida positiva
SPBTRS	Resolve sistemas lineares $AX=B$ , onde A é uma matriz banda, simétrica definida positiva cuja fatoração de Cholesky foi calculada pela SPBTRF
SPOCON	Estima o recíproco do número do condicionamento da matriz simétrica definida positiva usando a fatoração de Cholesky calculada pela SPOTRF
SPOTRF	Calcula a fatoração de Cholesky de uma matriz simétrica definida positiva
SPOTRI	Calcula a inversa da matriz simétrica definida positiva, a fatoração de Cholesky calculada pela SPOTRF
SPOTRS	Resolve sistemas lineares $AX=B$ , onde A é uma matriz simétrica definida positiva cuja fatoração de Cholesky foi calculada pela SPOTRF
SPPCON	Estima o recíproco do número do condicionamento da matriz compacta simétrica definida positiva usando a fatoração de Cholesky calculada pela SPPTRF
SPPTRF	Calcula a fatoração de Cholesky de uma matriz compacta simétrica definida positiva
SPPTRI	Calcula a inversa da matriz compacta simétrica definida positiva cuja fatoração de Cholesky foi calculada pela SPPTRF
SPPTRS	Resolve sistemas lineares $AX=B$ , onde A é uma matriz compacta simétrica definida positiva cuja fatoração de Cholesky foi calculada pela SPPTRF
STPTRI	Calcula a inversa de uma matriz triangular (superior e inferior) no formato de armazenamento compactado
STRTRI	Calcula a inversa de uma matriz triangular (superior e inferior)

- e) Rotinas de solução de sistemas de equações lineares especiais. Estas rotinas resolvem sistemas tridiagonais, recorrências lineares de primeira e segunda ordem. Elas são agrupadas sob o pacote de nome SPEC\_SYS.
- f) Rotinas de solução de sistemas lineares esparsos. O pacote destas rotinas é denominado de SPARSE. Elas resolvem sistemas reais esparsos pelo uso do método pré-condicionado tipo gradiente conjugado e sistemas reais simétricos esparsos e definidos usando a fatoração de matrizes definidas positivas.
- g) Rotinas de cálculo de autovalores de sistemas lineares densos. O pacote destas rotinas é conhecido por EISPACK. É um pacote de rotinas Fortran para resolver problemas de autovalores, para computar e usar decomposição de valores singulares. As rotinas no Cray, mantém o mesmo nome e algoritmo das originais. Elas foram otimizadas. Esta otimização inclui o uso de rotinas nos níveis 1, 2 e 3 da BLAS e de vetorização, sempre que possível, proporcionando a redução do tempo de execução.
- h) Rotinas de processamento de sinal. As rotinas de processamento de sinal incluem rotinas de transformadas rápidas de Fourier (FFT) e filtros. Elas aplicam transformadas de Fourier, sendo que cada rotina realiza análise ou síntese de Fourier. As rotinas de filtro são usadas para análise e projeto de filtros.
- i) Rotinas de ordenação e pesquisa. O pacote SEARCH é composto por rotinas de ordenação e pesquisa. As rotinas de pesquisa são agrupadas em rotinas que procuram elementos máximo e mínimo e rotinas de pesquisa vetorial. As rotinas de procura do máximo e mínimo procuram o maior e o menor elemento em um vetor, retornando o índice de cada um deles. As rotinas vetoriais de pesquisa procuram o número de ocorrências de um objeto no vetor e procuram pelo objeto no vetor.
- j) Rotinas científicas de operações especiais com vetores. Elas constituem o pacote SUPERSEDED. Elas resolvem sistemas equações pela inversa da matriz quadrada, multiplicam matrizes por vetores, matriz por matriz, matriz pelo vetor coluna somando o resultado a outro vetor coluna, matriz por vetor linha somando o resultado a outro vetor linha e espalha e agrupa vetores.

Outra biblioteca disponível é a IMSL, *International Mathematical and Statistical Libraries*. Ela é um conjunto de rotinas matemáticas escritas em Fortran e testadas pela *IMSL Incorporation*. Ela abrange vários campos da matemática e estatística, destacando-se em especial um conjunto de rotinas para resolver sistemas de equações.



### 6.2.2 Comparação da *libavi.a* com a *libsci.a*

A *libsci.a* é uma biblioteca científica que reúne vários pacotes e bibliotecas numéricas consagradas pelo passar dos anos, como as BLAS, LAPACK, IMSL etc. Seus códigos foram otimizados e vetorizados ao serem incorporados à *libsci.a*. Entretanto, por melhor que sejam os resultados numéricos produzidos em ponto-flutuante, eles ainda estão sujeitos a erros de arredondamentos e a problemas de instabilidade, não se podendo garantir plenamente os resultados.

A *libsci.a*, mais especificamente nas bibliotecas BLAS e LAPACK, trata de operações da álgebra linear e a resolução de sistemas lineares. Sendo um pacote específico para esta área, elas contêm uma série de rotinas para resolver diferentes tipos de sistemas reais e complexos. Os métodos empregados e a forma de armazenar os dados são voltados a características específicas da matriz de coeficientes.

Na *libavi.a*, só estão disponíveis rotinas que tratam de sistemas reais gerais, considerando as matrizes como densas. Algumas das rotinas existentes na *libsci.a* foram estendidas a intervalos reais na biblioteca *libavi.a*, contudo pode-se dizer que a exatidão é mantida. Ainda mais, como se trabalha com intervalos e a solução está incluída no intervalo, têm-se uma forma de garantir o resultado. Isto se dá graças aos arredondamentos direcionados para baixo e para cima aplicados nos extremos inferior e superior do intervalo, respectivamente. A inclusão de novas rotinas a *libavi.a* não deve ser problema por sua natureza modular.

## 6.3 Pascal XSC - módulos avançados

A descrição da linguagem PASCAL-XSC foi dada no capítulo 3, neste ítem só serão abordados os módulos padrões, pois o objetivo é a comparação com a biblioteca de rotinas intervalares *libavi.a*.

O Pascal-XSC possui módulos aritméticos avançados os quais contêm operadores e um conjunto de funções pré-definidas. Para os tipos *complex*, *interval* e *cinterval*, este conjunto contém todas as funções que são providenciadas para o tipo real. Os módulos disponíveis no Pascal-XSC são o C\_ARI (aritmética complexa), I\_ARI (aritmética intervalar), CI\_ARI (aritmética intervalar complexa), MV\_ARI (aritmética de matriz/vetor real), MVC\_ARI (aritmética de matriz/vetor complexa), MVI\_ARI (aritmética de matriz/vetor intervalar) e MVCI\_ARI (aritmética de matriz/vetor de intervalos complexos).

O tipo do resultado das operações aritméticas escalares é definido de acordo com a hierarquia de tipos dada na figura 6.2. O resultado é sempre o do maior tipo, dentre os tipos envolvidos na operação. Para as operações de matriz/vetor, a estrutura do resultado é dada pela tabela 6.7.

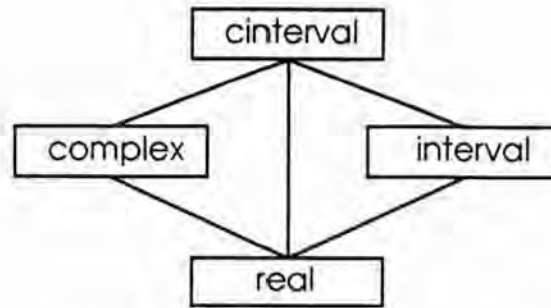


Fig.6.2 Hierarquia dos Tipos Aritméticos

Tabela 6.7 - Estrutura do resultado das operações com matriz/vetor

$v + v = v$	$v / s = v$	$s * m = m$
$m + m = m$	$v * v = s$	$m * s = m$
$v - v = v$	$s * v = v$	$m * m = m$
$m - m = m$	$v * s = v$	
$m / s = m$	$m * v = v$	

s - escalar v - vetor m - matriz

### 6.3.1 Módulo I\_ARI (Intervalos reais)

Este módulo suporta todos os operadores, funções e rotinas que manipulam intervalos de ponto-flutuante. O tipo intervalo é definido por:

```
type interval = record inf,sup: real end;
```

- Os operadores aritméticos e relacionais para os tipos de dados inteiro, real e intervalo são dados pela tabela 6.8.

Tabela 6.8 - Operadores do Módulo I\_ARI

Op.esquerdo/Op.direito	integer/real	interval
monodic		+, -
integer/real	++	o.in,=,<>,++
interval	o,=,<>,++	o.in,v,><,++,**

$o \in \{+, -, *, /\}$   $v \in \{=, <, >, <=, >= \}$

- Funções pré-definidas: (i,i1,i2 = interval ; r = real)

Tabela 6.9 - Funções Pré-definidas do Módulo I\_ARI

Função	Tipo de resultado	Descrição
sqr(i)	interval	intervalo ao quadrado
sqrt(i)	interval	raiz quadrada
exp(i)	interval	função exponencial
exp2(i)	interval	função potência, base 2
exp10(i)	interval	função potência, base 10
ln(i)	interval	logaritmo natural
log2(i)	interval	logaritmo, base 2
log10(i)	interval	logaritmo, base 10
sin(i)	interval	seno
cos(i)	interval	cosseno
tan(i)	interval	tangente
cot(i)	interval	cotangente
arcsin(i)	interval	arco seno
arccos(i)	interval	arco cosseno
arctan(i)	interval	arco tangente
arctan2(i1,i2)	interval	arco tangente
arccot(i)	interval	arco cotangente
sinh(i)	interval	seno hiperbólico
cosh(i)	interval	cosseno hiperbólico
tanh(i)	interval	tangente hiperbólica
coth(i)	interval	cotangente hiperbólica
arsinh(i)	interval	seno hiperbólico inverso
arcosh(i)	interval	cosseno hiperbólico inverso
artanh(i)	interval	tangente hiperbólica inversa
arcoth(i)	interval	cotangente hiperbólica inversa
abs(i)	interval	valor absoluto
mid(i)	real	ponto médio de i
diam(i)	real	diâmetro de i
blow(i,r)	interval	inflação epsilon

- Funções de transferência: (r, r1, r2 = real; i = interval)

Tabela 6.10 - Funções de Transferência do Módulo I\_ARI

Função	Resultado	Função	Resultado
intval(r1,r2)	interval	inf(i)	real
intval(r)	interval	sup(i)	real

- *Procedures* de entrada/saída:

```
procedure read (var f: text; var a: interval);
procedure write (var f: text; a: interval);
```

### 6.3.2 Módulo MV\_ARI (matrizes e vetores reais)

Este módulo suporta todas as operações, funções e *procedures* necessárias para computações com vetores e matrizes.

- Tipos: os tipos dinâmicos para a representação de vetores e matrizes reais são definidos por:
 

```
type rvector = dynamic array [*] of real;
type rmatrix = dynamic array [*] of rvector;
```
- Operadores:

Tabela 6.11 - Operadores do Módulo MV\_ARI

op.esquerdo/op.direito	integer/real	rvector	rmatrix
monadic		+, -	+, -
integer/real		*, *<, *>	*, *<, *>
rvector	*, *<, *>, /, /<, />	0, v	
rmatrix	*, *<, *>, /, /<, />	*, *<, *>	0, v

$o \in \{+, +<, +>, -, -<, ->, *, *<, *>\}$        $v \in \{=, <>, <, <=, >, >= \}$

- Funções pré-definidas: ( $v = \text{rvector}$ ;  $n, n1, n2 = \text{integer}$ ;  $M, M1, M2 = \text{rmatrix}$ )

Tabela 6.12 - Funções Pré-definidas do Módulo MV\_ARI

Função	Tipo de Resultado	Descrição
null(v)	rvector	vetor nulo com a faixa de índices de v
vnull(n)	rvector	vetor nulo com a faixa de índices [1..n]
null(M)	rmatrix	matriz nula com a faixa de índices de M
null(M1,M2)	rmatrix	matriz nula com a faixa de índices do produto de $M1 * M2$
null(n)	rmatrix	matriz nula com faixa de índices [1..n, 1..n]
null(n1,n2)	rmatrix	matriz nula com faixa de índices [1..n1, 1..n2]
id(M)	rmatrix	matriz identidade com a faixa de índices de M
id(M1,M2)	rmatrix	matriz identidade com a faixa de índices do produto de $M1 * M2$
id(n)	rmatrix	matriz identidade com a faixa de índices [1..n, 1..n]
id(n1,n2)	rmatrix	matriz identidade com a faixa de índices [1..n1, 1..n2]
transp(M)	rmatrix	matriz transposta $M_t$ de M com $M_t[i,j] = M[j,i]$

- *Procedure* de entrada/saída:

```
procedure read (var f: text; var a: rvector);
procedure read (var f: text; var A: rmatrix);
procedure write (var f: text; a: rvector);
procedure write (var f: text; A: rmatrix);
```



### 6.3.3 Módulo MVI\_ARI (matrizes e vetores de intervalos reais)

Este módulo suporta todas as operações, funções e procedures necessárias para computações com vetores e matrizes intervalares. O módulo MVI\_ARI providencia os operadores, funções e rotinas genéricas denominadas *intval*, *inf*, *sup*, *diam*, *mid*, *blow*, *transp*, *null*, *id read* e *write*. A função *intval* é usada para gerar vetores e matrizes e intervalos, onde *inf* e *sup* são funções que selecionam o ínfimo e supremo do intervalo em questão. O diâmetro e o ponto médio do vetor ou matriz intervalar pode ser calculado por *diam* e *mid*. O comando *blow* gera uma inflação no intervalo e *transp* gera a transposta da matriz. Vetores e matrizes nulas são geradas pela função *null*, enquanto que *id* produz uma matriz identidade do tipo apropriado. Finalmente, existem rotinas genéricas de entrada e saída, *read* e *write*, que se podem usar com todos os tipos de vetores e matrizes definidos nos módulos mencionados aqui.

- Tipos: type ivector = dynamic array [\*] of interval;  
imatrix = dynamic array [\*] of ivector;
- Operadores:

Tabela 6.13 - Operadores do Módulo MVI\_ARI

op.esquerdo/ op.direito	integer real	interval	rvector	ivector	rmatrix	imatrix
monadic				+, -		+, -
integer/real				*		*
interval			*	*	*	*
rvector		*, /	+*	O,=,<>,in,+*		
ivector	*, /	*, /	O,=,<>,+*	O,in,v,><,+*,**		
rmatrix		*, /		*	+*	O,=,<>,in,+*
imatrix	*, /	*, /	*	*	O,=,<>,+*	O,in,v,><,+*,**

$O \in \{+, -, *\}$      $V \in \{=, <, <=, >, >=\}$

- *Procedures* de entrada/saída:

```

procedure read (var f: text; var a: ivector);
procedure read (var f: text; var A: imatrix);
procedure write (var f: text; a: ivector);
procedure write (var f: text; A: imatrix);

```

- **Funções de transferência para vetores intervalares e matrizes intervalares**

Tabela 6.14 - Funções de Transferência do Módulo MVI\_ARI

Função	Tipo de resultado	Descrição
intval(rv1,rv2)	ivector	vetor intervalar iv com $iv[i] = \text{intval}(rv1[i],rv2[i])$
intval(rv)	ivector	vetor intervalar iv com $iv[i] = \text{intval}(rv[i])$
inf(iv)	rvector	vetor rv do limite inferior com $rv[i] = \text{inf}(iv[i])$
sup(iv)	rvector	vetor rv do limite superior com $rv[i] = \text{sup}(iv[i])$
intval(rM1,rM2)	imatrix	vetor intervalar iM com $iM[i,j] = \text{intval}(rM1[i,j],rM2[i,j])$
intval(rM)	imatrix	vetor intervalar iM com $iM[i,j] = \text{intval}(rM[i,j])$
inf(iM)	rmatrix	vetor rM do limite inferior com $rM[i,j] = \text{inf}(iM[i,j])$
sup(iM)	rmatrix	vetor rM do limite superior com $rM[i,j] = \text{sup}(iM[i,j])$

- **Funções pré-definidas:** (r = real; iv = ivector; iM,iM1,iM2 = imatrix)

Tabela 6.15 - Funções Pré-definidas do Módulo MVI\_ARI

Função	Tipo de resultado	Descrição
null(iv)	rvector	vetor nulo com a faixa de índices de iv
null(iM)	rmatrix	matriz nula com a faixa de índices de iM
null(iM1,iM2)	rmatrix	matriz nula com a faixa de índices do produto de $iM1 * iM2$
id(iM)	rmatrix	matriz identidade com a faixa de índices de iM
id(iM1,iM2)	rmatrix	matriz identidade com a faixa de índices do produto de $iM1 * iM2$
mid(iv)	rvector	ponto médio do vetor rv com $rv[i] = \text{mid}(iv[i])$
diam(iv)	rvector	diâmetro do vetor rv com $rv[i] = \text{diam}(iv[i])$
mid(iM)	rmatrix	ponto médio da matriz rM com $rM[i,j] = \text{mid}(iM[i,j])$
diam(iM)	rmatrix	diâmetro da matriz rM com $rM[i,j] = \text{diam}(iM[i,j])$
transp(iM)	rmatrix	matriz transposta iMt de iM com $iMt[i,j] = iM[j,i]$
blow(iv,r)	rvector	inflação do vetor epsilon ive com $ive[i] = \text{blow}(iv[i],r)$
blow(iM,r)	rmatrix	inflação do vetor epsilon iMe com $iMe[i,j] = \text{blow}(iM[i,j],r)$

### 6.3.4 Módulo CI\_ARI (Intervalos complexos)

Este módulo proporciona todas as operações, funções e rotinas necessárias para cálculos com intervalos complexos. A definição do tipo intervalo complexo, *cinterval* é um record com duas partes: *re* e *im* (respectivamente a parte real e a parte imaginária do intervalo complexo), ambos intervalos reais. Então, um intervalo complexo *z*, em Pascal XSC, é da forma:  $z = [x.inf, x.sup] + [y.inf, y.sup]$

Existem duas maneiras de se carregarem intervalos complexos. Na primeira, intervalos reais ou reais são convertidos em intervalos complexos, por essa razão a rotina se denomina *compl*. Na segunda maneira, introduzem-se valores complexos, destes são montados os intervalos da parte real e da parte imaginária. Como se introduz valores complexos, a rotina se denomina *intval*. Da mesma forma, existem entre as funções de transferência rotinas que, para dado intervalo complexo, fornecem os limites inferior e superior do intervalo complexo. No caso, estes limites são valores complexos (funções *inf*, *sup*) e, para este mesmo intervalo, existem rotinas que fornecem os intervalos que compõem a parte real e a parte imaginária (funções *re* e *im*).

As operações aritméticas e os operadores relacionais estão disponíveis para intervalos complexos, aplicando as operações entre intervalos reais na parte real e na parte imaginária. A operação necessita ser validada em ambas as partes.

Todas as funções do Pascal XSC disponíveis para argumentos reais estão disponíveis para argumentos intervalos complexos *ci*. Ainda estão disponíveis as funções que calculam: o conjugado do intervalo complexo *ci*, o componente angular da representação exponencial, o ponto médio, o diâmetro e a inflação do intervalo complexo. Elas estão descritas na tabela 6.17. Os domínios e os intervalos das funções pré-definidas são dependentes de máquina, variando de acordo com a implementação, sendo descritos no manual.

Tabela 6.16 Funções de transferência do módulo CI\_ARI

Função	Tipo do resultado	Significado
<i>compl(i1,i2)</i> <i>compl(r, i)</i> <i>compl(i, r)</i> <i>compl(i)</i>	<i>cinterval</i>	Intervalo complexo com parte real <i>i1</i> e parte complexa <i>i2</i> Intervalo complexo com parte real <i>r</i> e parte complexa <i>i</i> Intervalo complexo com parte real <i>i</i> e parte complexa <i>r</i> Intervalo complexo com parte real <i>i</i> e parte complexa nula ( 0 )
<i>intval(c1,c2)</i> <i>intval(r,c)</i> <i>intval(c,r)</i> <i>intval(c)</i>	<i>cinterval</i>	Intervalo complexo com parte real [ <i>c1.re</i> , <i>c2.re</i> ] e parte complexa [ <i>c1.im</i> , <i>c2.im</i> ] Com $c1 \leq c2$ Intervalo complexo com parte real [ <i>r</i> , <i>c.re</i> ] e parte complexa [0, <i>c.im</i> ] Com $r \leq c$ Intervalo complexo com parte real [ <i>c.re</i> , <i>r</i> ] e parte complexa [ <i>c.im</i> , 0] Com $c \leq r$ Intervalo complexo com parte real [ <i>c.re</i> , <i>c.re</i> ] e parte complexa [ <i>c.im</i> , <i>c.im</i> ] (degenerado)
<i>re(ci)</i>	<i>interval</i>	parte real de <i>ci</i> $i1 = [ci.re.inf, ci.re.sup]$
<i>im(ci)</i>	<i>interval</i>	parte imaginária de <i>ci</i> $i2 = [ci.im.inf, ci.im.sup]$
<i>inf(ci)</i>	<i>complex</i>	limite inferior complexo <i>z</i> de <i>ci</i> , onde $z = (ci.re.inf, ci.im.inf)$
<i>sup(ci)</i>	<i>complex</i>	limite superior complexo <i>z</i> de <i>ci</i> , onde $z = (ci.re.sup, ci.im.sup)$

Tabela 6.17 Funções pré-definidas do módulo CI\_ARI

Função	Tipo de resultado	Descrição
sqr(ci)	cinterval	intervalo ao quadrado
sqrt(ci)	cinterval	raiz quadrada
exp(ci)	cinterval	função exponencial
exp2(ci)	cinterval	função potência, base 2
exp10(ci)	cinterval	função potência, base 10
ln(ci)	cinterval	logaritmo natural
log2(ci)	cinterval	logaritmo, base 2
log10(ci)	cinterval	logaritmo, base 10
sin(ci)	cinterval	seno
cos(ci)	cinterval	coosseno
tan(ci)	cinterval	tangente
cot(ci)	cinterval	cotangente
arcsin(ci)	cinterval	arco seno
arccos(ci)	cinterval	arco coosseno
arctan(ci)	cinterval	arco tangente
arccot(ci)	cinterval	arco cotangente
sinh(ci)	cinterval	seno hiperbólico
cosh(ci)	cinterval	coosseno hiperbólico
tanh(ci)	cinterval	tangente hiperbólica
coth(ci)	cinterval	cotangente hiperbólica
arsinh(ci)	cinterval	seno hiperbólico inverso
arcosh(ci)	cinterval	coosseno hiperbólico inverso
artanh(ci)	cinterval	tangente hiperbólica inversa
arcoth(ci)	cinterval	cotangente hiperbólica inversa
conj(ci)	cinterval	conjugado de $ci = a + bi$
abs(ci)	interval	valor absoluto de ci
arg(ci)	interval	componente angular da representação exponencial
mid(ci)	complex	ponto médio de ci
diam(ci)	real	diâmetro de ci
blow(ci,r)	cinterval	inflação epsilon blow=compl(blow(ci,rc,r), blow(ci,im,r))

As rotinas de entrada e saída de dados adotam a notação entre parêntesis para os componentes complexos e entre colchetes para intervalos. O módulo CI\_ARI torna disponíveis as rotinas; **procedure** read(**var** f:text;**var** a:cinterval); e a **procedure** write(**var** f:text; a:cinterval); com um parâmetro opcional de arquivo, um número arbitrário de parâmetros de entrada/saída sem formato específico. Intervalos complexos podem ser introduzidos na forma (6.1). A impressão dos intervalos complexos é da forma de intervalo complexo geral.

$$\left( \begin{array}{ll}
 ([x,y],[v,w]) & \text{intervalo complexo geral} \\
 (x,[v,w]) & \text{quando } x=y \\
 ([x,y],v) & \text{quando } v=w \\
 [x,y] & \text{quando } v=w=0 \\
 (x,v) & \text{quando } x=y \text{ e } v=w \\
 x & \text{quando } x=y \text{ e } v=w=0
 \end{array} \right. \quad (6.1)$$



### 6.3.5 Comparação da *libavia* com o Pascal-XSC

O Pascal-XSC é, aritmeticamente falando, completo, pois segue o padrão binário de aritmética de ponto-flutuante IEEE 754, possui arredondamentos direcionados, controle de arredondamento nas operações aritméticas, produto escalar exato e avaliação exata de expressões. Proporciona também algumas vantagens como: portabilidade do código, produz o mesmo resultado em qualquer plataforma, tem disponível a matemática intervalar e uma aritmética de alta exatidão, possibilitando o desenvolvimento de algoritmos com verificação automática de resultados.

O Pascal-XSC é portátil em várias plataformas, estando disponível para computadores pessoais (PC's) e estações de trabalho do tipo Sun e HP. Essa portabilidade é garantida pelo uso de um compilador que traduz para a linguagem ANSI-C. Como o principal objetivo do sistema é a portabilidade, o mesmo teve que providenciar uma fácil portabilidade do compilador e do sistema de execução; teve que permitir a necessária reconfiguração do compilador para cada novo computador; permitiu a portabilidade do código gerado através da compilação cruzada; e providenciou a consistência dos resultados para todas as plataformas.

A produção de resultados idênticos em todas as plataformas é resultante do Pascal-XSC proporcionar uma completa simulação da aritmética de ponto-flutuante definida pelo padrão binário IEEE-754 ([HOL94a]).

O Pascal-XSC é uma linguagem de programação de propósito geral que proporciona condições especiais à implementação de algoritmos numéricos sofisticados, que verificam matematicamente os resultados. Pelo uso dos módulos matemáticos, os algoritmos numéricos que providenciam alta exatidão e verificação automática de resultados podem ser facilmente programados. Além disso, o projeto de programas para as Engenharias e para a Computação Científica é simplificado, graças à estrutura modular dos programas, à possibilidade de definição de operadores, à sobrecarga de funções, rotinas e operadores, às funções e operadores com tipos arbitrários de dados e aos *arrays* dinâmicos.

Os programas escritos nesta linguagem são de fácil leitura e existe ainda uma grande quantidade de problemas numéricos que podem ser resolvidos pelas bibliotecas de rotinas com verificação automática de resultados.

Esta característica é desejável a *libavia*, mas, devido a limitações e restrições resultantes da arquitetura do Cray não foi possível este padrão de qualidade no momento. Entretanto, a *libavia* possui a vantagem de ser uma biblioteca em um supercomputador, com maior capacidade de memória e maior velocidade de processamento, um maior poder computacional, o que possibilita a produção de resultados confiáveis para problemas de grande porte e computações de larga escala.

## 6.4 Conclusões

Neste capítulo foi comparada a biblioteca de rotinas intervalares *libavi.a* com pacotes e bibliotecas numéricas. Inicialmente o objetivo da comparação foi identificar qualidades, operações e funções disponíveis nestas bibliotecas para que também fossem incorporadas à biblioteca *libavi.a*. Não houve preocupação neste capítulo de avaliar qualitativamente os resultados, pois esta tarefa será feita nos capítulos seguintes. Observa-se, entretanto, que o uso da aritmética de ponto-flutuante, por melhor que seja, sempre estará sujeita a arredondamentos e problemas de instabilidade numérica, o que pode tornar difícil ou mesmo impossível, a garantia dos resultados.

Com o uso da aritmética intervalar, têm-se que o resultado está contido no intervalo. Nas operações, os arredondamentos direcionados para baixo e para cima e aplicados nos extremos do intervalo, garantem a inclusão da solução nos cálculos, além do uso da alta exatidão, que proporciona que os limites sejam os mais exatos possíveis, gerando resultados confiáveis e verificados automaticamente.

Considerou-se apenas a linguagem Pascal-XSC para a comparação das extensões de linguagens de computação científica, por ser sua filosofia análoga às demais linguagens disponíveis. Em termos gerais, o que vale para o PASCAL-XSC vale para C-XSC e Fortran-XSC.

Como os resultados produzidos pelo Pascal-XSC são idênticos em qualquer plataforma, eles serão tomados como padrão de qualidade para comparações com resultados produzidos pela biblioteca *libavi.a*.

A característica modular da *libavi.a* facilita futuras incorporações de novos métodos, como por exemplo a resolução de sistemas lineares especiais ou complexos.

## 7 USO EFETIVO DA MATEMÁTICA INTERVALAR

Neste capítulo é fornecido um relato dos testes desenvolvidos no sentido de verificar a biblioteca de rotinas intervalares e de validar o uso efetivo da matemática intervalar em supercomputadores vetoriais.

Os testes de verificação englobam programas que verificam as operações aritméticas entre intervalos, vetores e matrizes de intervalos e intervalos complexos. Estes testes foram de consistência.

Para os demais testes foram desenvolvidos programas em Fortran 90, utilizando-se a *libavi.a* e programas em Pascal-XSC. Os resultados de ambos os programas foram comparados. Considerando que os resultados em Pascal-XSC são os mesmos em qualquer plataforma, estes foram tomados como corretos.

Neste capítulo são apresentados os programas junto com comentários dos resultados. A listagem completa dos testes (programas e execução) em Fortran 90 e em Pascal XSC podem ser encontradas em [DIV95c].

### 7.1 Verificação da Biblioteca de Rotinas Intervalares

Neste ítem serão descritos os testes realizados com a biblioteca de rotinas intervalares *libavi.a*, no sentido de verificar a confiabilidade bem como a validação das rotinas implementadas e a qualidade dos seus resultados.

A confiabilidade e a validação foram verificadas através de representantes típicos de cada possível caso das operações aritméticas e entre conjuntos, os demais testes foram por amostragem geral. Também foram testados alguns casos de instabilidade verificados em [DIV95a].

A qualidade dos resultados foi avaliada em função dos resultados obtidos em Pascal-XSC. Ou seja, os mesmos programas foram desenvolvidos e executados no Cray, com a *libavi.a*, e no PC486 em Pascal-XSC. Os resultados foram comparados e analisados. As conclusões destas comparações e análises são dadas a seguir.

Observa-se que os resultados produzidos em Pascal-XSC foram tomados como exatos, por ele produzir resultados com alta exatidão independente da plataforma onde os cálculos são realizados.

### 7.1.1 Aritmética Básica

A aritmética intervalar básica foi testada através de representantes típicos de cada um dos possíveis casos. Decidiu-se por testar todas as possibilidades, pois essas rotinas que implementam as operações aritméticas elementares são utilizadas em todos os demais níveis e módulos da biblioteca. Para o desenvolvimento desses testes, foram analisadas as operações e definidos exemplos para cada uma das possíveis situações ou casos. Esses exemplos foram combinados entre si, gerando o conjunto de testes para os quais foram verificadas as operações de adição, subtração e multiplicação. No caso da divisão, os mesmos intervalos foram tomados como numeradores. Os intervalos utilizados como denominadores são todos os que não contêm o zero, pois, caso contrário, o resultado implica uma situação de erro, denominada de divisão por zero.

Também foram testadas as operações entre conjuntos como união e intersecção. Para a escolha dos exemplos, foram identificados doze casos. Estes incluem intervalos disjuntos e contidos. Escolheram-se, ainda, intervalos onde a continuidade ou descontinuidade, depende do tipo de arredondamento, por serem utilizados números de máquina bem próximos. Utilizou-se a convenção de que o conjunto vazio é definido pelo valor  $[1, -1]$ , que não é um intervalo (pela definição). No caso da união, onde ela só é definida para intervalos não disjuntos, utilizou-se o valor  $[1, 0]$ . Na verdade, este valor é utilizado sempre como um indicador de função não definida.

Foram testadas ainda as funções trigonométricas (*sin*, *cos* e *tan*) nos quatro quadrantes e com a redução de quadrantes e as funções *sqr* e *sqrt*, que calculam a potência ao quadrado e a raiz quadrada.

As mesmas rotinas foram implementadas em Fortran 90 (com o uso da *libavi.a*) e em Pascal XSC, com o objetivo de testar a qualidade dos resultados obtidos e o tratamento de erros através de uma comparação. Portanto, foram desenvolvidos programas que testaram a adição, subtração, multiplicação, divisão, raiz quadrada, potência ao quadrado, intersecção, união e união convexa de intervalos reais.

#### 7.1.1.1 Adição e subtração de intervalos

As primeiras rotinas analisadas foram a **adição e subtração** de intervalos. Para este teste foram necessários dez intervalos. Estes são somados e subtraídos entre si. Os resultados são armazenados em duas matrizes de intervalos denominadas:  $AD[i,j]$ , que armazena as somas dos intervalos  $int[i]$  com o intervalo  $int[j]$ , e  $SU[i,j]$ , que armazena as diferenças entre os intervalos  $int[i]$  e  $int[j]$ . As cem somas e as cem subtrações são impressas no final da execução do programa. Os intervalos são atribuídos no corpo do programa e armazenados em um vetor de intervalos.



```

program somasub
use basico
use mvi
type (interval), dimension(10,10) :: AD, SUB
type (interval), dimension(10) :: INT
integer :: I,J
  print *, 'Teste da aritmetica basica - Adicao e subtracao de intervalos'
  INT(1) = sintpt(0.0)
  INT(2) = sintpt(7.0)
  INT(3) = sintpt(-13.0)
  INT(4) = sintval(-5.0,-2.0)
  INT(5) = sintval(-2.0,-1.0)
  INT(6) = sintval(1.0,2.0)
  INT(7) = sintval(6.0,7.0)
  INT(8) = sintval(-3.0,0.0)
  INT(9) = sintval(0.0,3.0)
  INT(10) = sintval(-1.0,1.0)
  call svwrite(6,int)
  do I = 1, 10
    do J = 1, 10
      AD(I,J) = INT(I)+INT(J)
      SU(I,J) = INT(I)-INT(J)
    end do
  end do
  print *, 'Matriz SOMA'
  call smwrite(6,ad)
  print *, 'Matriz DIFERENCA'
  call smwrite(6,sub)
end program

```

Fig 7.1 Programa para a adição e subtração de intervalos em Fortran 90

```

program somasub(input, output);
use i_ari, mvi_ari;
var AD,SU: imatrix [1..10, 1..10];
    int: ivector[1..10];
    i,j: integer;
begin
writeln('Teste da aritmética básica - Adição e subtração de intervalos');
writeln;
int[1]:=0; int[2]:=7;int[3]:=-13;int[4]:=intval(-2,-1);int[5]:=intval(1,2);
int[6]:=intval(6,7);int[7]:=intval(-3,0);int[8]:=intval(0,3);int[9]:=intval(-1,1);int[10]:=intval(-5,-2);
for i:=1 to 10 do
  begin
    writeln('intervalo',i);
    writeln(int[i]);
  end;
for i:=1 to 10
for j:=1 to 10 do
  begin
    AD[i,j]:=int[i] + int[j];
    SU[i,j]:=int[i] - int[j];
  end;
writeln('matriz soma');
for i:=1 to 10
for j:=1 to 10 do
  begin
    writeln(i, j, AD[i,j]);
  end;
writeln('matriz diferenca');
for i:=1 to 10
for j:=1 to 10 do
  begin
    writeln(i, j, SU[i,j]);
  end;
end.

```

Fig 7.2 Programa para a adição e subtração de intervalos em Pascal XSC

O programa descrito na figura 7.1 realiza a adição e subtração dos intervalos seleccionados, em Fortran 90, utilizando o módulo *básico* da biblioteca *libavi.a*. A figura 7.2 apresenta o programa análogo, em Pascal XSC. Os resultados de ambos os programas foram armazenados em matrizes de intervalos. Para a impressão dos resultados foram utilizadas as rotinas de impressão de tipos intervalares. Tratam-se das rotinas *swrite*, *svwrite* e *smwrite* que imprimem, respectivamente, intervalos, vetores e matrizes de intervalos.

Os resultados foram comparados e, a não ser pelo arredondamento efetuado na impressão, produziram os mesmos valores. Com isto, pode-se afirmar que as rotinas que implementam a adição e subtração de intervalos no módulo básico não foram somente verificadas, mas também validadas, pois produzem a mesma qualidade do resultado, isto é, entre 14 a 15 dígitos significativos exatos.

### 7.1.1.2 Multiplicação de intervalos

Para a **multiplicação** intervalar foram identificados nove casos. Foram escolhidos três intervalos de forma que, operados entre si, testem todos os nove casos da multiplicação intervalar, como é descrito pela figura 7.3. Foram também seleccionados quatro intervalos degenerados: um positivo, um negativo, o nulo e o unitário, o qual atua como elemento neutro da multiplicação intervalar.

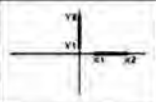


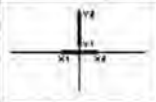
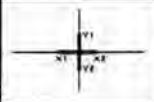
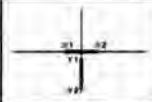
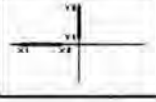
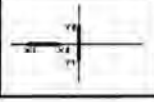

$X \cdot Y$	$0 \leq Y1$	$Y1 < 0 < Y2$	$Y2 \rightarrow 0$
$0 \leq X1$			
$X1 < 0 < X2$			
$X2 \leq 0$			

Fig 7.3 Nove casos da definição de produto de intervalos

No total foram onze intervalos combinados entre si. Os intervalos foram inicializados nos programas pelas rotinas de atribuição e *intval*, em Pascal XSC, e pelas rotinas *sinprt* e *sinprval* do módulo *básico* da *libavi.a*. Observou-se que em Pascal XSC há diferença na forma como os valores são atribuídos. No caso degenerado, o número é armazenado com todos os dígitos disponíveis. No caso da rotina *intval* somente são armazenados os dígitos introduzidos. Isto repercute na aritmética de máquina, pois os arredondamentos direccionados para baixo e para cima devem ser aplicados no dígito menos significativo da mantissa.

```

program mult
use basico, mvi
type (interval), dimension(11,11) :: MUL
type (interval), dimension(11) :: INT
integer :: I,J
  print *, 'Teste da aritmetica basica - Multiplicao de intervalos'
  INT(1)=sintpt(0.0)
  INT(2)=sintpt(7.0)
  INT(3)=sintpt(-13.0)
  INT(4)=sintval(-5.0,-2.0)
  INT(5)=sintval(-2.0,-1.0)
  INT(6)=sintval(1.0,2.0)
  INT(7)=sintval(6.0,7.0)
  INT(8)=sintval(-3.0,0.0)
  INT(9)=sintval(0.0,3.0)
  INT(10)=sintval(-1.0,1.0)
  INT(11)=sintpt(1.0)
  do I = 1, 11
    print *, 'intervalo', I
    call swrite(6,INT(I))
  end do
  do I = 1, 11
    do J = 1, 11
      MUL(I,J) = INT(I)*INT(J)
    end do
  end do
  print *, 'Matriz PRODUTO'
  call smwrite(6,MUL)
end program

```

Fig. 7.4 Programa do produto de intervalos em Fortran 90

```

program Produto (input,output) ;
use
  i_ari,nvi_ari;
var
  PROD:matrix[1..11,1..11];
  int:vector[1..11];
  i,j:integer;
begin
  writeln('Teste de Aritmetica Basica - Produto');
  writeln;
  int[1]:=0;
  int[2]:=7;
  int[3]:=-13;
  int[4]:=intval(-5,-2);
  int[5]:=intval(-2,-1);
  int[6]:=intval(1,2);
  int[7]:=intval(6,7);
  int[8]:=intval(-3,0);
  int[9]:=intval(0,3);
  int[10]:=intval(-1,1);
  int[11]:=1;
  for i:=1 to 11 do
    writeln('Intervalo ',i,' = ',int[i]);
  end for;
  for i:=1 to 11 do
    for j:=1 to 11 do
      PROD[i,j]:=int[i]*int[j];
    end for;
  end for;
  writeln('Matriz do Produto');
  for i:=1 to 11 do
    for j:=1 to 11 do
      begin
        writeln(i,' ',j,' ',PROD[i,j]);
      end;
    end for;
  end for;
end.

```

Fig. 7.5 Programa do produto de intervalos em Pascal-XSC

O programa em Fortran 90 é descrito pela figura 7.4, enquanto que a figura 7.5 descreve o programa em Pascal XSC. Observou-se que em geral os resultados produzidos pelos dois programas são similares, exceto pelo arredondamento realizado na impressão.

### 7.1.1.3 Divisão de intervalos

Para a **divisão** foram utilizados os mesmos intervalos da multiplicação (figura 7.6), com a alteração de que o divisor não contenha o zero, como mostra a figura 7.7. Foi inserido o teste do intervalo com extremo muito próximo ao zero (esta proximidade é diferente nas linguagens Pascal e Fortran 90).

Vetor de intervalos A - numerador

```
A[1]= [5, 5]
A[2]= [-6, -6]
A[3]= [1, 1]
A[4]= [-1, -1]
A[5]= [ 3.0E+000, 4.0E+000 ]
A[6]= [-7.0E+000, -3.0E+000 ]
A[7]= [ 7.8E+001, 9.5E+001 ]
A[8]= [-8.3E+001, -4.5E+001 ]
A[9]= [-3.0E+000, -4.9E-324 ]
A[10]= [ 0, 0]
A[11]= [-1.0E+000, 1.0E+000 ]
A[12]= [ 0.0E+000, 1.0E+000 ]
A[13]= [-1.0E+000, 0.0E+000 ]
```

Fig.7.6 Intervalos utilizados na divisão

Vetor de intervalos B - denominador

```
B[1]= [5, 5]
B[2]= [-6, -6]
B[3]= [1, 1]
B[4]= [-1, -1]
B[5]= [ 3.0E+000, 4.0E+000 ]
B[6]= [-7.0E+000, -3.0E+000 ]
B[7]= [ 7.8E+001, 9.5E+001 ]
B[8]= [-8.3E+001, -4.5E+001 ]
B[9]= [-3.0E+000, -1.0E-306 ]
```

Fig.7.7 Intervalos utilizados na divisão como divisores (zero não está incluído)

Na figura 7.8 tem-se o programa em Pascal XSC e na figura 7.9 o correspondente em Fortran 90. Observa-se que há diferença nos valores próximos ao zero. Foram testados ainda os casos onde a divisão possuía como denominador um intervalo contendo zero. O erro foi identificado e sinalizado de acordo com as informações da documentação. No resultado obtido em Fortran, identificaram-se várias vezes números com expoentes extremamente grandes, na ordem de  $10^x$ , onde  $x \geq 2463$ , resultantes da divisão de valores pequenos.



```

program Divisao (input,output);
use
  i_ari,mvi_ari;
var
  DIVISAO:matrix[1..9,1..13];
  A:ivector[1..13];
  B:ivector[1..9];
  i,j:integer;
begin
  writeln('Teste de Aritmetica Basica - Divisao em Pascal XSC');
  A[1]:=5; A[2]:=-6; A[3]:=1; A[4]:=-1;
  A[5]:=intval(3,4); A[6]:=intval(-7,-3); A[7]:=intval(78,95);
  A[8]:=intval(-83,-45); A[9]:=intval(-3,-4.9E-324); A[10]:=0;
  A[11]:=intval(-1,1); A[12]:=intval(0,1); A[13]:=intval(-1,0);
  B[1]:=5; B[2]:=-6; B[3]:=1; B[4]:=-1; B[5]:=intval(3,4); B[6]:=intval(-7,-3);
  B[7]:=intval(78,95); B[8]:=intval(-83,-45); B[9]:=intval(-3,-1.0E-306);
  for i:=1 to 13 do
    writeln('Intervalo A[,i,]= ',A[i]);
  for r:=1 to 9 do
    writeln('Intervalo B[,r,]= ',B[r]);
  for j:=1 to 13 do
    for i:=1 to 9 do
      DIVISAO[i,j]:=A[i]/B[i];
  writeln('Matriz da Divisao');
  for j:=1 to 13 do
    begin
      writeln('Coluna ',j);
      for i:=1 to 9 do
        writeln(i,' ',DIVISAO[i,j]);
      end;
  end.

```

Fig 7.8 Programa da divisão em Pascal XSC

```

program TESTE_DIV
use basico
use mvi
type (interval), dimension(9,13) :: DIV
type (interval), dimension(13) :: INT
integer :: I,J
  print *,'Teste da aritmetica basica - Divisao'
  INT(1)=sintpt(5,0)
  INT(2)=sintpt(-6,0)
  INT(3)=sintpt(1,0)
  INT(4)=sintpt(-1,0)
  INT(5)=sintval(3,0,4,0)
  INT(6)=sintval(-7,0,-3,0)
  INT(7)=sintval(78,0,95,0)
  INT(8)=sintval(-83,0,-45,0)
  INT(9)=sintval(-3,0,-1,51562346234E-1500)
  INT(10)=sintpt(0,0)
  INT(11)=sintval(-1,0,1,0)
  INT(12)=sintval(0,0,1,0)
  INT(13)=sintval(-1,0,0,0)
  call svwrite(6,INT)
  do I=1,13
    do J=1,9
      DIV(I,J)=INT(I)/INT(J)
    end do
  end do
  print *,'Matriz Quociente'
  call smwrite(6,DIV)
end program

```

Fig 7.9 Programa da divisão em Fortran 90

### 7.1.1.4 Funções entre conjuntos - intersecção e união

Para a **Intersecção e União** foram selecionados doze casos, de forma a esgotar as possibilidades da rotina e de verificar o comportamento dessas operações quando os extremos dos intervalos são muito próximos, distanciando-se por apenas um  $\epsilon$  muito pequeno, na ordem do menor número de máquina.

```
program Uniao_Interseccao (input,output);
```

```
use
```

```
  _ari,mvl_ari;
```

```
var
```

```
  A:interval;
```

```
  B,UNIAO,INTER:vector[1..12];
```

```
  i:integer;
```

```
begin
```

```
  writeln('Teste de Aritmetica Basica - Uniao e Interseccao');
```

```
  A:=intval(5,10);
```

```
  B[1]:=intval(13,20);
```

```
  B[2]:=intval(8,15);
```

```
  B[3]:=intval(6,9);
```

```
  B[4]:=intval(4,11);
```

```
  B[5]:=intval(10,20);
```

```
  B[6]:=intval(10+1.0E-100,20);
```

```
  B[7]:=intval(10-1.0E-100,20);
```

```
  B[8]:=intval(-5,3);
```

```
  B[9]:=intval(2,7);
```

```
  B[10]:=intval(0,5-1.0E-100);
```

```
  B[11]:=intval(0,5+1.0E-100);
```

```
  B[12]:=intval(0,5);
```

```
  writeln('Intervalo A= ',A);
```

```
  for i:=1 to 12 do
```

```
    writeln('Intervalo B[',i,']= ',B[i]);
```

```
  writeln;
```

```
  for i:=1 to 12 do
```

```
    begin
```

```
      UNIAO[i]:=A+B[i];
```

```
      if (A.sup<B[i].inf) or (B[i].sup<A.inf) then
```

```
        INTER[i]:=0
```

```
      else
```

```
        INTER[i]:=A*B[i];
```

```
    end;
```

```
  writeln;
```

```
  writeln('Vetor da Uniao');
```

```
  writeln;
```

```
  for i:=1 to 12 do
```

```
    writeln(i, ',UNIAO[i];
```

```
  writeln('Vetor da Interseccao');
```

```
  for i:=1 to 12 do
```

```
    writeln(i, ',INTER[i];
```

```
end.
```

Fig 7.10 Programa de união e intersecção de intervalos em Pascal XSC

```

program uni_inter
use basico
use mvl
type (interval) :: A
type (interval), dimension(12) :: INT
integer :: I,J
  print *, 'Teste de operações como conjuntos'
  A=sintval(5.0,10.0)
  INT(1)=sintval(13.0,20.0)
  INT(2)=sintval(8.0,15.0)
  INT(3)=sintval(6.0,9.0)
  INT(4)=sintval(4.0,11.0)
  INT(5)=sintval(10.0,20.0)
  INT(6)=sintval(NEAREST(10.0,+1.0),20.0)
  INT(7)=sintval(NEAREST(10.0,-1.0),20.0)
  INT(8)=sintval(-5.0,3.0)
  INT(9)=sintval(2.0,7.0)
  INT(10)=sintval(0.0,NEAREST(5.0,-1.0))
  INT(11)=sintval(0.0,NEAREST(5.0,+1.0))
  INT(12)=sintval(0.0,5.0)
  do i=1, 12
    call exwrite(6,INT(i))
  end do
  write(6,*) ''
  do I=1,12
    call swrite(6,uni(A,INT(I)))
    call swrite(6,unid(A,INT(I)))
    call swrite(6,sinter(A,INT(I)))
  end do
end program

```

Fig 7.11 Programa de união e intersecção de intervalos em Fortran 90

As figuras 7.10 e 7.11 apresentam, respectivamente, os programas em Pascal XSC e Fortran 90. Os casos selecionados abrangem dois intervalos A e B onde eles são: disjuntos e  $A > B$ , disjuntos e  $A < B$ , A contido em B, A contém B, não disjuntos e  $a_2 > b_1$ , não disjuntos e  $b_2 > a_1$ , disjuntos por apenas um  $\epsilon$ , com um único número de intersecção, entre outros casos.

### 7.1.1.5 Radiciação e Potenciação - funções *sqr* e *sqrt*

O teste das funções *sqr* e *sqrt* foi efetuado num conjunto de dez intervalos degenerados baseados em números primos e quadrados perfeitos. As funções foram aplicadas em ordem alternada:  $sqr(sqrt(x))$  e  $sqrt(sqr(x))$ . (Em Fortran 90, o nome dessas rotinas é precedido por mais um *s*, indicando que a função é intervalar). O valor inicial foi mantido, ou seja, elas se comportaram como funções inversas, uma da outra. Neste caso foi verificada a comutatividade destas operações, pois a ordem em que foram aplicadas não alterou o resultado.

Estes testes foram desenvolvidos em Fortran 90 (figura 7.12) e em Pascal XSC (figura 7.13) e seus resultados confrontados. Tanto em Fortran quanto em Pascal XSC as funções foram validadas.

```

program Raiz_Quadrada (input,output);
use
  i_ari.mvi_ari;
var
  A,POT_RAIZ,RAIZ_POT:ivector[1..10];
  i:integer;
begin
  writeln('Teste de Aritmetica Basica - Raiz Quadrada e Potenciacao em PAscal XSC');
  A[1]:=1;
  A[2]:=2;
  A[3]:=3;
  A[4]:=5;
  A[5]:=7;
  A[6]:=13;
  A[7]:=16;
  A[8]:=36;
  A[9]:=49;
  A[10]:=169;
  for i:=1 to 10 do
    A[i]:=ival(pred(A[i].inf).succ(A[i].sup));
  for i:=1 to 10 do
    writeln ('Intervalo ',i,':',A[i]);
  for i:=1 to 10 do
    begin
      RAIZ_POT[i]:=sqrt(sqr(A[i]));
      POT_RAIZ[i]:=sqr(sqrt(A[i]));
    end;
  writeln('Potencia e Raiz');
  for i:=1 to 10 do
    writeln(i:'.POT_RAIZ[i]);
  writeln;
  writeln('Raiz e Potencia');
  for i:=1 to 10 do
    writeln(i:'.RAIZ_POT[i]);
end.

```

Fig 7.12 Programa teste das funções *sqr* e *sqrt* em Pascal XSC



```

program Raiz_Quadrada
use basico
type (interval), dimension(10) :: A, PTRZ, RZPT
integer I
print *, 'Teste Raiz Quadrada e Potenciacao'
print *, ''
A(1)=sintpt(1.0)
A(2)=sintpt(2.0)
A(3)=sintpt(3.0)
A(4)=sintpt(5.0)
A(5)=sintpt(7.0)
A(6)=sintpt(13.0)
A(7)=sintpt(16.0)
A(8)=sintpt(36.0)
A(9)=sintpt(49.0)
A(10)=sintpt(169.0)
do I=1,10
A(I)=sintval(NEAREST(A(I)%sup,+1.0),NEAREST(A(I)%mf,-1.0))
end do
do I=1,10
    print *, 'Intervalo', I
    call exwrite(6,A(I))
end do
print *, ''
do I=1,10
    RZPT(I)=ssqrt(sqrt(A(I)))
    PTRZ(I)=ssqr(sqrt(A(I)))
end do
print *, 'Potencia e Raiz'
do I=1,10
    print *, I
    call exwrite(6,PTRZ(I))
end do
print *, 'Raiz e Potencia'
do I=1,10
    print *, I
    call exwrite(6,RZPT(I))
end do
stop
end program

subroutine exwrite(A,B)
include 'inter.mc'
type (interval), intent (in) :: B
integer, intent (in) :: A
write(A, '(f22.17)') B%mf,B%sup
write(A, '*')
end subroutine exwrite

```

Fig 7.13 Programa teste das funções ssqr e ssqrt em Fortran 90

### 7.1.1.6 Funções trigonométricas

O teste das funções trigonométricas intervalares abrangeu as funções seno, cosseno e tangente no intervalo de  $-2\pi$  a  $2\pi$ , variando de  $\pi/12$ . Com esse teste, foi coberto um período do domínio das funções, incluindo inclusive valores impróprios, de descontinuidade, como no caso da tangente.

Mais uma vez foram desenvolvidos programas em Fortran 90 e em Pascal XSC, como mostram as figuras 7.14 e 7.15, para as funções seno e cosseno. A função tangente foi avaliada em separado, devido aos valores de descontinuidade. Os resultados obtidos foram satisfatórios.

```

program Trigonometria (input,output);
use
  i_ari,mv1_ari;
var
  mt,SENO,COSENO:vector[1..53];
  i:integer;
function p:real;
begin
  pi:=4*arctan(1);
end;
begin
  writeln('Tese de Aritmetica Basica - Seno e Coseno');
  writeln;
  for i:=1 to 49 do
    mt[i]:=(-2*pi)+((pi/12)*(i-1));
  mt[50]:=423*pi/180;
  mt[51]:=497*pi/180;
  mt[52]:=588*pi/180;
  mt[53]:=679*pi/180;
  for i:=1 to 49 do
    writeln('Intervalo ',i,' -360+((15*i)-15), graus =,mt[i]);
  writeln('Intervalo 50 423 graus =,mt[50]);
  writeln('Intervalo 51 497 graus =,mt[51]);
  writeln('Intervalo 52 588 graus =,mt[52]);
  writeln('Intervalo 53 679 graus =,mt[53]);
  writeln;
  for i:=1 to 53 do
    SENO[i]:=sin(mt[i]);
  for i:=1 to 53 do
    COSENO[i]:=cos(mt[i]);
  writeln('Seno: ');
  for i:=1 to 49 do
    writeln('Seno ',-360+((15*i)-15), ' graus =,SENO[i]);
  writeln('Seno 423 graus =,SENO[50]);
  writeln('Seno 497 graus =,SENO[51]);
  writeln('Seno 588 graus =,SENO[52]);
  writeln('Seno 679 graus =,SENO[53]);
  writeln;
  for i:=1 to 49 do
    writeln('Coseno ',-360+((15*i)-15), ' graus =,COSENO[i]);
  writeln('Coseno 423 graus =,COSENO[50]);
  writeln('Coseno 497 graus =,COSENO[51]);
  writeln('Coseno 588 graus =,COSENO[52]);
  writeln('Coseno 679 graus =,COSENO[53]);
end.

```

Fig 7.14 Programa teste das funções trigonométricas intervalares em Pascal XSC

```

program Trigonometria
use basico
use mvi
type(interval), dimension(53) :: int, SENO, COSENO
integer :: I,J
real :: pi
  print *, 'Tese de Arimetica Basica - Seno e Coseno'
  print *
  pi=4*atan(1.0)
  do I=1, 49
    int(I)=sintpt((-2*pi)+((pi/12)*(I-1)))
  end do
  int(50)=sintpt(423*pi/180)
  int(51)=sintpt(497*pi/180)
  int(52)=sintpt(588*pi/180)
  int(53)=sintpt(679*pi/180)
  do I=1, 49
    print *, 'Intervalo ', I, ': -360+((15*I)-15), grau = ', int(I)
  end do
  print *, 'Intervalo 50: 423 graus = ', int(50)
  print *, 'Intervalo 51: 497 graus = ', int(51)
  print *, 'Intervalo 52: 588 graus = ', int(52)
  print *, 'Intervalo 53: 679 graus = ', int(53)
  print *
  do I=1, 53
    SENO(I)=ssin(int(I))
  end do
  do I=1, 53
    COSENO(I)=scos(int(I))
  end do
  print *, 'Seno: '
  do I=1, 49
    print *, 'Seno ', -360+((15*I)-15), ' grau = ', SENO(I)
  end do
  print *, 'Seno 423 graus = ', SENO(50)
  print *, 'Seno 497 graus = ', SENO(51)
  print *, 'Seno 588 graus = ', SENO(52)
  print *, 'Seno 679 graus = ', SENO(53)
  print *
  do I=1, 49
    print *, 'Coseno ', -360+((15*I)-15), ' grau = ', COSENO(I)
  end do
  print *, 'Coseno 423 graus = ', COSENO(50)
  print *, 'Coseno 497 graus = ', COSENO(51)
  print *, 'Coseno 588 graus = ', COSENO(52)
  print *, 'Coseno 679 graus = ', COSENO(53)
  stop
end program

```

Fig 7.15 Programa teste das funções trigonométricas intervalares em Fortran 90

## 7.1.2 Influência do erro de arredondamento e representatividade

Neste item são analisados a influência dos erros de arredondamentos e a representatividade de números em ponto-flutuantes. Para tanto, foram desenvolvidos dois programas conforme [HAM93]. O primeiro deve testar diferentes maneiras de se calcular o valor da expressão  $z = x^4 - 4y^4 - 4y^2$ , onde  $x$  e  $y$  são reais. O segundo deve testar se um quociente  $z/n$  de dois números inteiros ( $z$  e  $n$ ) é representável como um número de ponto-flutuante, ou seja, verificar se um número racional é exatamente representável como um número real de máquina.

Apesar destes programas não envolverem intervalos, eles foram desenvolvidos com o objetivo de estabelecer um parâmetro entre os resultados produzidos em Fortran 90 e Pascal-XSC. Ambos os programas foram desenvolvidos nas duas linguagens e seus resultados confrontados.

### 7.1.2.1 Influência dos erros de arredondamentos

O primeiro programa trata da influência dos erros de arredondamentos durante o cálculo da expressão  $z = x^4 - 4y^4 - 4y^2$  para diferentes valores de  $x$  e  $y$ . Inicialmente foi desenvolvido um programa em Pascal-XSC, que aceita valores reais para  $x$  e  $y$  e calcula o valor de  $z$  pelos métodos descritos abaixo. O programa foi testado para os valores  $x=665857.0$  e  $y=470832.0$ . Para estes valores o valor exato da expressão  $z$  é igual a 1.

**método 1:** Calcular o valor da expressão  $z = x .x .x .x -4 . y . y . y . y -4 . y . y$ , usando: (a) as operações de multiplicação (\*) e subtração (-) em ponto-flutuante; (b) os operadores de multiplicação e subtração com arredondamento direcionado, representados por \*<, \*> e -< para dar um limite inferior para essa expressão e, (c) os operadores de multiplicação e subtração com arredondamento direcionado na outra direção, representados por \*>, \*< e -> para dar um limite superior para a expressão.

**método 2:** Calcular o valor da expressão  $z = x^2 . x^2 - 4 . y^2 . y^2 - 4 . y^2$  usando a função pré-definida *sqr* (eleva ao quadrado) e os operadores de multiplicação e subtração, sem arredondamentos direcionados.

**método 3:** Calcular o valor da expressão  $z = (x^2)^2 - (2 . y^2)^2 - (2 . y)^2$  usando a função pré-definida *sqr* (eleva ao quadrado) e os operadores de multiplicação e subtração, sem arredondamentos direcionados.

**método 4:** Calcular o valor da expressão  $z = (x^2)^2 - (2 . y)^2 . (y^2 + 1)$  usando a função pré-definida *sqr* (eleva ao quadrado) e os operadores de multiplicação e subtração, sem arredondamentos direcionados.

**método 5:** Calcular o valor da expressão de forma exata,  $z = \#*$  (a.a - b.b -c.c) com  $a = x^2$ ,  $b = 2.y^2$  e  $c = 2.y$ . O símbolo *#\** indica que devem ser utilizadas variáveis *dotprecision* para o cálculo exato da expressão.



program arredondamento (input, output);

```

var x,y,z: real;
    a,b,c: real;
begin
  writeln ('Influencia dos Erros de Arredondamento ');
  writeln;
  write ('x = '); read (x);
  write ('y = '); read (y);
  writeln;
  writeln ('Cálculo da expressão z = x ^4 - 4y^4 - 4y^2 ');
  writeln;
  z := x*x*x*x - 4*y*y*y*y - 4*y*y;
  writeln (' Cálculo 1: x*x*x*x - 4*y*y*y*y - 4*y*y           =', z);
  z := (x<x)*(x<x) -<4*>(y>y)*(y>y) -<4*>(y>y);
  writeln (' Cálculo 2: (x<x)*(x<x) -<4*>(y>y)*(y>y) -<4*>(y>y) =', z);
  z := (x>x)*(x>x) ->4*<(y<y)*(y<y) ->4*<(y<y);
  writeln (' Cálculo 3: (x>x)*(x>x) ->4*<(y<y)*(y<y) ->4*<(y<y) =', z);
  z := sqrt (x) * sqrt(x) - 4* sqrt(y) * sqrt(y) - 4*sqrt(y);
  writeln (' Cálculo 4: x^2 * x^2 - 4*y^2 *y^2 -4*y^2           =', z);
  z := sqrt(sqrt(x)) - sqrt(2*sqrt(y)) - sqrt (2*y);
  writeln (' Cálculo 5: (x^2)^2 - (2*y^2)^2 - (2*y)^2           =', z);
  z := sqrt (sqrt(x)) - sqrt (2*y) * (sqrt(y) + 1);
  writeln (' Cálculo 6: (x^2)^2 - (2*y)^2*(y^2+1)               =', z);
  a := sqrt (x);
  b := 2*sqrt(y);
  c := 2*y;
  z := #*(a*a - b*b - c*c);
  writeln (' Cálculo 7: #*( x^2*x^2 - (2*y^2) * (2*y^2) - (2*y) * (2*y)) =', z);
end.

```

#### Solução:

Influencia dos Erros de Arredondamento

$x = 665857,0$   $y = 470832,0$

Cálculo da expressão  $z = x^4 - 4y^4 - 4y^2$

Cálculo 1:  $x*x*x*x - 4*y*y*y*y - 4*y*y = 1.188556800000E+007$

Cálculo 2:  $(x<x)*(x<x) -<4*>(y>y)*(y>y) -<4*>(y>y) = -5.522329600000E+007$

Cálculo 3:  $(x>x)*(x>x) ->4*<(y<y)*(y<y) ->4*<(y<y) = 1.188556800000E+007$

Cálculo 4:  $x^2 * x^2 - 4*y^2 *y^2 -4*y^2 = 1.188556800000E+007$

Cálculo 5:  $(x^2)^2 - (2*y^2)^2 - (2*y)^2 = 1.188556800000E+007$

Cálculo 6:  $(x^2)^2 - (2*y)^2*(y^2+1) = 0.000000000000E+000$

Cálculo 7:  $\#*(x^2*x^2 - (2*y^2)*(2*y^2) - (2*y)*(2*y)) = 1.000000000000E+000$

Fig 7.16 Programa e execução do teste da influência dos erros de arredondamento em Pascal XSC

A expressão # permite que se obtenha o resultado exato. Isto se deve ao fato de que todos os operandos ( $a, b, c$ ) são exatos, desde que  $x=665875$  e  $y=470832$  sejam exatamente representáveis.

## program ARREDONDAMENTO

```

real x,y,z
real xd,xu,yd,yu
  print *,'Influencia dos erros de arredondamento em Fortran 90'
  print *,'X= 665857.0'
  print *,'Y= 470832.0'
  print *,'Calculo da expressao z=x^4 - 4y^4 - 4y^2'
  x=665857.0
  y=470832.0
  xd4=infla_dn(infla_dn(x*x)*infla_dn(x*x))
  xu4=infla_up(infla_up(x*x)*infla_up(x*x))
  yd4=infla_dn(infla_dn(y*y)*infla_dn(y*y))
  yu4=infla_up(infla_up(y*y)*infla_up(y*y))
  yd2=infla_dn(y*y)
  yu2=infla_up(y*y)
  z=x*x*x*x-4*y*y*y*y-4*y*y
  print *,'Calculo 1: x*x*x*x-4*y*y*y*y-4*y*y           =',z
  z=infla_dn(infla_dn(xd4-infla_up(4*yu4))-infla_up(4*yu2))
  print *,'Calculo 2: (x<x)*<(x<x)-<4*>(y>y)*>(y>y)-<4*>(y>y)='z
  z=infla_up(infla_up(xu4-infla_dn(4*yd4))-infla_dn(4*yd2))
  print *,'Calculo 3: (x>x)*>(x>x)->4*<(y<y)*<(y<y)->4*<(y<y)='z
  z=(x**2)*(x**2)-4*(y**2)*(y**2)-4*(y**2)
  print *,'Calculo 4: x^2*x^2-4*y^2*y^2-4*y^2           =',z
  z=((x**2)**2)-(((y**2)**2)**2)-((2*y)**2)
  print *,'Calculo 5: (x^2)^2-(2*y^2)^2-(2*y)^2         =',z
  z=((x**2)**2)-((2*y)**2)*(y**2+1)
  print *,'Calculo 6: (x^2)^2-(2*y)^2*(y^2+1)           =',z
contains
function infla_dn (A) result (C)
  real:: A,C
  C=NEAREST(A,-1.0)
end function infla_dn
function infla_up (A) result (C)
  real:: A,C
  C=NEAREST(A,+1.0)
end function infla_up
end program

```

Execução em Fortran 90

```

Influencia dos erros de arredondamento em Fortran 90
X= 665857.0
Y= 470832.0
Calculo da expressao z=x^4 - 4y^4 - 4y^2
Calculo 1: x*x*x*x-4*y*y*y*y-4*y*y           = 1073741824.
Calculo 2: (x<x)*<(x<x)-<4*>(y>y)*>(y>y)-<4*>(y>y) = -7336535040.011749
Calculo 3: (x>x)*>(x>x)->4*<(y<y)*<(y<y)->4*<(y<y) ) = 7695850496.011749
Calculo 4: x^2*x^2-4*y^2*y^2-4*y^2           = 1073741824.
Calculo 5: (x^2)^2-(2*y^2)^2-(2*y)^2         = 1073741824.
Calculo 6: (x^2)^2-(2*y)^2*(y^2+1)           = 0.E+0

```

Fig 7.17 Programa e execução do teste da influência dos erros de arredondamento em Fortran 90

Observa-se que em Fortran nenhuma das formas produziu o valor exato, antes produziram valores extremamente grandes.

### 7.1.2.1 Teste de Representatividade

Este segundo programa testa a representatividade de números racionais, definidos como o quociente de dois números inteiros  $z$  e  $n$  no conjunto dos números reais representáveis na máquina, ou seja, no conjunto de números em ponto-flutuante. O programa foi inicialmente desenvolvido em Pascal-XSC. Utilizou-se um par de números inteiros, do tipo *integer* ( $z$  e  $n$ , com  $n \neq 0$ ). O quociente  $z/n$  é exatamente representável como um número *real* de máquina (no conjunto de números em ponto-flutuante), se e somente se  $z < n = z > n$ . O programa foi desenvolvido para verificar um número arbitrário de

pares. No caso do quociente ser exatamente representável, são impressos os valores de  $z$ ,  $n$  e do quociente  $z/n$ . Se a condição não é verificada, então é impressa uma mensagem de advertência. Foi adotado  $n=0$  como critério de parada, ou seja condição de finalização. O programa também calcula uma percentagem dos pares com quociente exatamente representável.

```

program repesenta (input, output);
var n,z:integer;
    no_de_exatos, no_de_pares:integer;
    quociente:real;
begin
  writeln('Teste de representatividade');
  writeln;
  no_de_exatos:=0;
  no_de_pares:=0;
  write('Entre com z e n para o teste ');
  read(z,n);
  while n<>0 do
    begin
      quociente:=z/n;
      no_de_pares:=no_de_pares+1;
      if quociente=z/>n then
        begin
          no_de_exatos:=no_de_exatos+1;
          writeln('Quociente exatamente representavel !');
          writeln(z:1, '/', n:1, '=', quociente);
        end
      else
        writeln('Quociente nao e exatamente representavel !');
        writeln;
        write('Entre com z e n para o teste de quociente: ');
        read(z,n);
      end;
    if no_de_pares<>0 then
      begin
        writeln;
        writeln(no_de_pares, ' pares de dados que entraram');
        writeln(no_de_exatos, ' quocientes exatamente representaveis');
        writeln('Estes sao ', no_de_exatos/no_de_pares*100:5:1, '%');
      end;
    end;
  end.

```

### Solução:

Teste de representatividade

```

Entre com z e n para o teste de quociente:  0      0
Quociente exatamente representavel !
4/2 = 2.000000000000000E+000
Entre com z e n para o teste de quociente:  1      10
Quociente nao e exatamente representavel !
Entre com z e n para o teste de quociente:   3      5
Quociente nao e exatamente representavel !
Entre com z e n para o teste de quociente:  5      2
Quociente exatamente representavel !
5/2= 2.500000000000000E+000
Entre com z e n para o teste de quociente: 213     77
Quociente nao e exatamente representavel !
Entre com z e n para o teste de quociente:  1      0
5 pares de dados que entraram
2 quocientes exatamente representaveis - Estes sao 40.0%

```

Fig 7.18 Programa e execução do teste da representatividade em Pascal XSC

```

program REPRESENTA
integer, dimension(5) :: z,n
integer exatos,pares
real quociente,taxa,prox
  print *, 'Teste de Representatividade'
  print *
  exatos=0
  pares=0
  z(1)=4
  z(2)=1
  z(3)=3
  z(4)=5
  z(5)=213
  n(1)=2
  n(2)=10
  n(3)= 5
  n(4)=77
  n(5)=0
  do I=1, 5
    pares=pares+1
    quociente=z(I)/n(I)
    erro=abs(n(I)*quociente-z(I))
    print *, 'Quociente: ',z(I),', ',n(I)
    if (erro<7.3E-2466) then
      exatos=exatos+1
      print *, 'Quociente exatamente representavel !'
      print *, z(I), '/', n(I), '=', quociente
    else
      print *, 'Quociente nao e exatamente representavel !'
    end if
  end do
  if (pares/=0) then
    print *, 'Pares de dados que entraram: ',pares
    print *, 'Quocientes exatamente representaveis',exatos
    taxa=exatos/pares*100
    print *, 'Estes sao ',taxa,'% '
  end if
end program

```

```

Teste de Representatividade em Fortran 90
Quociente: 4, 2.
Quociente exatamente representavel !
4./2.=2.
Quociente: 1, 10.
Quociente nao e exatamente representavel !
Quociente: 3, 5.
Quociente exatamente representavel !
3./5.=0.5999999999999979
Quociente: 5, 2.
Quociente exatamente representavel !
5./2.=2.5
Quociente: 213, 77.
Quociente exatamente representavel !
213./77.=2.766233766233768
Pares de dados que entraram: 5.
Quocientes exatamente representaveis: 4,
Estes sao 79.99999999999955%

```

Fig 7.19 Programa e execucao do teste da representatividade em Fortran 90



### 7.1.3 Testes do Produto escalar

A mais importante operação na álgebra linear é o produto escalar, que aparece na multiplicação de vetores, multiplicação de matrizes e na multiplicação de vetores por matrizes. Estas operações por sua vez, são utilizadas para resolver, entre outros problemas, sistemas de equações lineares.

A forma como esta operação é implementada é de grande importância para a determinação da qualidade final do resultado, ainda mais em computadores vetoriais, onde a ordem como as operações são efetuadas geralmente foge do controle do programador, devido à vetorização. A essência do problema da implementação do produto escalar, reside no número de arredondamentos que são efetuados e no tamanho do registrador que acumula os produtos.

Na biblioteca *libavl.a* foram implementadas três formas diferentes de se efetuar o produto escalar. Apesar de nenhuma destas formas obedecerem às regras de semimorfismo ou utilizarem o controle do arredondamento, pretende-se com este exemplo demonstrar não só a importância desta operação como a fonte de erros que uma implementação não ótima pode representar nos cálculos.

Inicialmente serão mostrados alguns cálculos em Pascal-XSC do produto escalar calculado de forma usual (acadêmica) e com máxima exatidão. São feitas algumas variações para ilustrar os diferentes resultados que se podem obter. Por fim, estes cálculos serão feitos pelas diferentes formas possíveis em Fortran 90, utilizando a biblioteca de rotinas intervalares *libavl.a*, inclusive com a extensão intervalar das rotinas de produto escalar disponível na biblioteca BLAS, contida no arquivo *libsci.a* da Cray.

Antes de descrever o programa em Pascal-XSC, recorda-se a definição de produto escalar dada pela fórmula (2.37) no item 2.1.3. Mais detalhes sobre o produto escalar são dados no anexo 10.1 e no relatório de pesquisa que trata das limitações do processamento vetorial no Cray ([DIV95a]).

O produto escalar de dois vetores  $x$  e  $y$  é definido como o somatório dos produtos dos elementos correspondentes de  $x$  e  $y$ , podendo ser anotado por:  $\sum x_i y_i$ . Este produto escalar é calculado de forma usual e com máxima exatidão. Para maior clareza, foram desenvolvidas duas funções: a *scalp* que calcula o produto escalar à maneira comum e a função *Max\_Acc\_Scalp* que calcula o produto escalar com máxima exatidão, onde os produtos  $x_i y_i$  são acumulados em uma variável do tipo *dotprecision*. Neste caso só é feito um arredondamento no final, quando o resultado é armazenado em uma variável real.

Os vetores  $x$  e  $y$  serão inicializados no programa principal. Foram escolhidos arbitrariamente vetores de tamanho  $n = 5$ , ou seja com cinco elementos. Os valores para este teste são dados por (7.1). A função *Max\_Acc\_Scalp* deste programa simula a funcionalidade do operador  $*$  para o tipo *rvector* proporcionada pelo módulo *MV\_ARI*.

$$x = \begin{pmatrix} 2.718281828E10 \\ -3.141592654E10 \\ 1.414213562E10 \\ 5.772156649E9 \\ 3.010299957E9 \end{pmatrix}, y = \begin{pmatrix} 1.4862497E12 \\ 8.783669879E14 \\ -2.237492E10 \\ 4.773714647E15 \\ 1.85049E5 \end{pmatrix} \quad (7.1)$$

```

program Prod_Esc_Int(input,output);
use I_ari, mvt_ari;
var a,b,c : ivector [1..5];
    i : integer;
    s_n : interval;
begin
    a[1]:= intval (2.718281828e10);    a[2]:= intval (-3.141592654e10);    a[3]:= intval (1.414213562e10);
    a[4]:= intval (5.772156649e9);    a[5]:= intval (3.010299957e9);
    b[1]:= intval (1.4862497e12);    b[2]:= intval (8.783669879e14);    b[3]:= intval (-2.237492e10);
    b[4]:= intval (4.773714647e15);    b[5]:= intval (1.85049e5);
    writeln('Produto Escalar normal e com alta exatidao?');
    for i:=1 to 5 do writeln (a[i]);
        writeln;
    for i:=1 to 5 do writeln (b[i]);
        writeln;
    s_n := intval(0);
    c[i]:= intval(0);
    for i:= 1 to 5 do
        begin
            c[i] := a[i] * b[i];
        end;
    for i:= 1 to 5 do
        s_n := s_n + c[i];
    write ('Prod.es.comum :');
    write (s_n);
    writeln;
    write ('Prod.es.dotpr:');
    s_n := intval(0);
    s_n := ## (for i:=lbound(a) to ubound(a).sum (a[i] * b[i]));
    write (s_n);
end.

```

o primeiro vetor :

```

[ 2.7182818280000000E+010, 2.7182818280000000E+010 ]
[ -3.1415926540000000E+010, -3.1415926540000000E+010 ]
[ 1.4142135620000000E+010, 1.4142135620000000E+010 ]
[ 5.7721566490000000E+009, 5.7721566490000000E+009 ]
[ 3.0102999570000000E+009, 3.0102999570000000E+009 ]

```

o segundo vetor :

```

[ 1.4862497000000000E+012, 1.4862497000000000E+012 ]
[ 8.7836698790000000E+014, 8.7836698790000000E+014 ]
[ -2.2374920000000000E+010, -2.2374920000000000E+010 ]
[ 4.7737146470000000E+015, 4.7737146470000000E+015 ]
[ 1.8504900000000000E+005, 1.8504900000000000E+005 ]

```

Prod.es.comum :[ -8.6E+009, 8.7E+009 ]

Prod.es.dotpr:[ -1.0065710700000000E+008, -1.0065710700000000E+008 ]

Fig 7.20 Programa e execucao do produto escalar intervalar em Pascal XSC

A seguir foi desenvolvido um programa-exemplo em Fortran 90, que utiliza as rotinas disponíveis na biblioteca *libavia*. Como as rotinas da biblioteca são intervalares, os vetores  $x$  e  $y$  foram considerados como intervalos degenerados. A rotina *svmult* corresponde ao produto escalar usual, a rotina *sdot* é uma versão intervalar da rotina disponível na biblioteca BLAS para o cálculo do produto escalar em precisão simples. Por fim, a rotina *sddot*, sendo também uma extensão intervalar da biblioteca BLAS, calcula o produto escalar em dupla precisão, arredondando no final o resultado para uma variável de precisão simples. Os dados, os valores dos vetores foram atribuídos dentro do próprio programa, para facilitar a compreensão do programa.

```

program produto-escalar
use basico
use aplic
use mvi
type (interval) :: es1,es2,es3
type (interval), dimension(5) :: A,B
print *, 'Teste do produto escalar'
print *, ''
A(1)=sintpt(2.718281828e10)
A(2)=sintpt(-3.141592654e10)
A(3)=sintpt(1.414213562e10)
A(4)=sintpt(5.772156649e9)
A(5)=sintpt(3.010299957e9)
B(1)=sintpt(1.4862497e12)
B(2)=sintpt(8.783669879e14)
B(3)=sintpt(-2.237492e10)
B(4)=sintpt(4.773714647e15)
B(5)=sintpt(1.85049e5)
print *, 'Utilizando svmult:'
es1=svmult(A,B)
call swrite(6,es1)
print *, 'Utilizando svdot:'
es2=svdot(A,B)
call swrite(6,es2)
print *, 'Utilizando svddot:'
es3=svddot(a,b)
call swrite(6,es3)
end program ursula

```

Saida:

```

Teste do produto escalar
Utilizando svmult:
[-5.3683749301400E+11, 5.6267413477000E+11]
Utilizando svdot:
[ 1.9864223744000E+10, 1.9864223744000E+10]
Utilizando svddot:
[-1.0065710700000E+08, -1.0065710700000E+08]

```

Fig 7.21 Programa e execução do produto escalar em Fortran 90

Observa-se que os resultados obtidos contém o valor exato.

### 7.1.4 Cálculos com matrizes de intervalos

Neste exemplo são ilustradas as operações com matrizes de intervalos. Observa-se que nos programas em Pascal-XSC, as matrizes são apresentadas por linhas, enquanto que nos programas em Fortran 90, utilizando a biblioteca de rotinas intervalares *libavi.a*, as matrizes são apresentadas por colunas.

O teste desenvolvido aqui é simples, mas ilustra como introduzir e imprimir matrizes de intervalos e como operá-las. Mais uma vez é apresentado inicialmente o programa em Pascal-XSC e a seguir a versão do programa em Fortran 90.

$$A = \begin{pmatrix} [1,1] & [0,1] \\ [1,1] & [-1,1] \end{pmatrix}, B = \begin{pmatrix} [-1,2] & [3,4] \\ [2,2] & [-6,-4] \end{pmatrix} \quad (7.2)$$

Inicialmente é calculado o produto das matrizes intervalares A.B. São calculadas também algumas expressões matriciais intervalares para demonstrar que a multiplicação não é associativa e nem distributiva em relação à adição. Estas propriedades já foram demonstradas no capítulo dois, aqui através destes exemplos são novamente demonstradas.

Então o programa em Pascal-XSC inicialmente calcula A+B, A-B e A.B. Então são calculados os produtos A.(A.A) e (A.A).A e as expressões A.(B+A) e A.B+A.A. As operações utilizadas aqui são suportadas pelo módulo avançado MVI\_ARI do Pascal-XSC.

```

program Matrizes_Intervalares (input,output);
use i_ari,mvi_ari;
var A,B,C:matrix [1..2,1..2];
begin
  writeln('Calculos para Matrizes Intervalares');
  writeln;
  A[1,1]:=1;  A[1,2]:=intval(0,1);
  A[2,1]:=1;  A[2,2]:=intval(-1,1);
  B[1,1]:=intval(-1,2);  B[1,2]:=intval(3,4);
  B[2,1]:=2;  B[2,2]:=intval(-6,-4);
  writeln('A=');writeln;  writeln(A);
  writeln('B=');writeln;  writeln(B);
  writeln('A+B=');writeln;  writeln(A+B);
  writeln('A-B=');writeln;  writeln(A-B);
  writeln('A*B=');writeln;  writeln(A*B);
  writeln('A*(A*A)=');writeln;  writeln(A*(A*A));
  writeln('(A*A)*A=');writeln;  writeln((A*A)*A);
  writeln('A*(B+A)=');writeln;  writeln(A*(B+A));
  writeln('A*B+A*A=');writeln;  writeln(A*B+A*A);
end

```

#### Solução:

##### Cálculos para Matrizes Intervalares

A=	[ 1.000000000000000E+000, 1.000000000000000E+000 ]		0.0E+000, 1.0E+000 ]
	[ 1.000000000000000E+000, 1.000000000000000E+000 ]		-1.0E+000, 1.0E+000 ]
B=	[ -1.0E+000, 2.0E+000 ]		3.0E+000, 4.0E+000 ]
	[ 2.000000000000000E+000, 2.000000000000000E+000 ]		-6.0E+000, -4.0E+000 ]
A+B=	[ 0.0E+000, 3.0E+000 ]		3.0E+000, 5.0E+000 ]
	[ 3.000000000000000E+000, 3.000000000000000E+000 ]		-7.0E+000, -3.0E+000 ]



A-B=	[	-1.0E+000,	2.0E+000 ]		-4.0E+000,	-2.0E+000 ]
		-1.000000000000000E+000,	-1.000000000000000E+000 ]		3.0E+000,	7.0E+000 ]
A*B=		-1.0E+000,	4.0E+000 ]		-3.0E+000,	4.0E+000 ]
		-3.0E+000,	4.0E+000 ]		-3.0E+000,	1.0E+001 ]
A*(A*A)=		1.0E+000,	4.0E+000 ]		-2.0E+000,	4.0E+000 ]
		-1.0E+000,	4.0E+000 ]		-3.0E+000,	4.0E+000 ]
(A*A)*A=		0.0E+000,	4.0E+000 ]		-2.0E+000,	4.0E+000 ]
		-1.0E+000,	4.0E+000 ]		-2.0E+000,	4.0E+000 ]
A*(B+A)=		0.0E+000,	6.0E+000 ]		-4.0E+000,	5.0E+000 ]
		-3.0E+000,	6.0E+000 ]		-4.0E+000,	1.2E+001 ]
A*B+A*A=		0.0E+000,	6.0E+000 ]		-4.0E+000,	6.0E+000 ]
		-3.0E+000,	6.0E+000 ]		-4.0E+000,	1.2E+001 ]

Fig 7.22 Programa e execução de operações com matrizes intervalares em Pascal XSC

Pelos resultados fica demonstrado que a multiplicação de matrizes intervalares não é associativa e nem distributiva em relação a adição. Observa-se, ainda, que na figura 7.23 as operações entre matrizes intervalares são levadas a cabo pelo uso do módulo *mvi* da biblioteca *libavi.a*.

#### program Matrizes\_intervalares

```

use basico
use mvi
type(interval), dimension(2,2) :: A,B,C
  print *, 'Calculos para Matrizes Intervalares em Fortran 90'
  print *
  A(1,1)=smtp(1.0)
  A(1,2)=smrval(0.0,1.0)
  A(2,1)=smtp(1.0)
  A(2,2)=smrval(-1.0,1.0)
  B(1,1)=smrval(-1.0,2.0)
  B(1,2)=smrval(3.0,4.0)
  B(2,1)=smtp(2.0)
  B(2,2)=smrval(-6.0,-4.0)
  print *, 'A='
  call smwrite(6,A)
  print *, 'B='
  call smwrite(6,B)
  print *, 'A+B='
  call smwrite(6,A+B)
  print *, 'A-B='
  call smwrite(6,A-B)
  print *, 'A*B='
  call smwrite(6,A*B)
  print *, 'A*(A*A)='
  call smwrite(6,A*(A*A))
  print *, '(A*A)*A='
  call smwrite(6,(A*A)*A)
  print *, 'A*(B+A)='
  call smwrite(6,A*(A+B))
  print *, 'A*B+A*A='
  call smwrite(6,A*B+A*A)
end program

A=
[ 1.000000000000000E+00, 1.000000000000000E+00] [ 0.000000000000000E+00, 1.000000000000000E+00]
[ 1.000000000000000E+00, 1.000000000000000E+00] [-1.000000000000000E+00, 1.000000000000000E+00]

B=
[ -1.000000000000000E+00, 2.000000000000000E+00] [ 3.000000000000000E+00, 4.000000000000000E+00]
[ 2.000000000000000E+00, 2.000000000000000E+00] [-6.000000000000000E+00, -4.000000000000000E+00]

```

```

A+B=
|-7.3344154702194-2466, 3.000000000000E+00|| 3.000000000000E+00, 5.000000000000E+00|
| 3.000000000000E+00, 3.000000000000E+00|| -7.000000000000E+00, -3.000000000000E+00|

A-B=
|-1.000000000000E+00, 2.000000000000E+00|| -4.000000000000E+00, -2.000000000000E+00|
|-1.000000000000E+00, -1.000000000000E+00|| 3.000000000000E+00, 7.000000000000E+00|

A*B=
|-1.000000000000E+00, 4.000000000001E+00|| -3.000000000001E+00, 4.000000000001E+00|
|-3.000000000000E+00, 4.000000000001E+00|| -3.000000000001E+00, 1.000000000000E+01|

A*(A*A)=
| 9.999999999999E-01, 4.000000000001E+00|| -2.000000000001E+00, 4.000000000001E+00|
|-1.000000000001E+00, 4.000000000001E+00|| -3.000000000001E+00, 4.000000000001E+00|

(A*A)*A=
|-4.263256414560E-14, 4.000000000001E+00|| -2.000000000001E+00, 4.000000000001E+00|
|-1.000000000000E+00, 4.000000000001E+00|| -2.000000000001E+00, 4.000000000001E+00|

A*(B+A)=
|-1.4668830940439-2465, 6.000000000001E+00|| -4.000000000001E+00, 5.000000000001E+00|
|-3.000000000000E+00, 6.000000000001E+00|| -4.000000000001E+00, 1.200000000000E+01|

A*B+A*A=
|-3.5527136788005E-14, 6.000000000001E+00|| -4.000000000001E+00, 6.000000000001E+00|
|-3.000000000001E+00, 6.000000000001E+00|| -4.000000000001E+00, 1.200000000000E+01|

```

Fig 7.23 Programa e execução em Fortran 90 de operações com matrizes intervalares

## 7.1.5 Outros testes

Para ilustrar a influência da ordem das parcelas nos resultados, foi desenvolvido um teste, de um somatório longo. Este exemplo foi inicialmente analisado em [DIV95a], onde as parcelas positivas e negativas eram alternadas com o mesmo expoente e a unidade variava de posição. Desenvolveu-se, então, um novo programa onde as parcelas positivas e negativas são separadas por blocos em cada uma das somas  $S_i$ , ou seja,  $S_0$ , por exemplo, é definido pela fórmula (7.3).

$$\begin{aligned}
 S_{i1} := & 1 + 16^N + 16^{N-1} + \dots + 16^{-N} - 16^N - 16^{N-1} - \dots + 16^{-N} \\
 & + 16^N + 16^{N-1} + \dots + 16^{-N} - 16^N - 16^{N-1} - \dots + 16^{-N}
 \end{aligned}
 \quad (7.3)$$

O programa foi desenvolvido em Pascal XSC (ambiente do PC-486). Os resultados foram similares aos do modo sequencial do Cray. Um exemplo aparece quando  $N=28$ , onde  $S_0$  foi calculado como  $-1.912504 \text{ E} + 17$ . O resultado correto seria um. Desenvolveu-se, então, um programa em Pascal XSC utilizando as variáveis do tipo *dotprecision* para o cálculo de expressões exatas (*#-expressions*) e o resultado obtido foi o correto.

Tabela 7.1 Tabulação dos resultados do Cray para o programa do exemplo 7.1.5

N	Processamento Escalar		Processamento Vetorial	
	Real	Intervalar	Real	Intervalar
1	0.100000E+01	[ 9.999999999907E-01, 1.000000000007E+00]	0.100000E+01	[ 9.999999999907E-01, 1.000000000007E+00]
2	0.100000E+01	[ 9.999999997803E-01, 1.000000000183E+00]	0.100000E+01	[ 9.999999997803E-01, 1.000000000183E+00]
3	0.100000E+01	[ 9.999999953237E-01, 1.000000004094E+00]	0.100000E+01	[ 9.999999953237E-01, 1.000000004094E+00]
4	0.100000E+01	[ 9.999999065569E-01, 1.000000084130E+00]	0.100000E+01	[ 9.999999065569E-01, 1.000000084130E+00]
5	0.100000E+01	[ 9.999982068931E-01, 1.000000164409E+00]	0.100000E+01	[ 9.999982068931E-01, 1.000000164409E+00]
6	0.100000E+01	[ 9.999965945879E-01, 1.000002928574E+00]	0.100000E+01	[ 9.999965945879E-01, 1.000002928574E+00]
7	0.999997E+00	[ 9.999378206828E-01, 1.000054423262E+00]	0.999998E+00	[ 9.999378206828E-01, 1.000054423262E+00]
8	0.999953E+00	[ 9.988830566561E-01, 1.000992838557E+00]	0.999937E+00	[ 9.988830566561E-01, 1.000992838557E+00]
9	<b>0.999251E+00</b>	<b>[ 9.801757812509E-01, 1.017838541667E+00]</b>	<b>0.998992E+00</b>	<b>[ 9.801757812509E-01, 1.017838541667E+00]</b>
10	0.988020E+00	[ 6.515625000002E-01, 1.316666666668E+00]	0.983886E+00	[ 6.515625000002E-01, 1.316666666668E+00]
11	0.808333E+00	[ -4.950000000003E+00, 6.566666666670E+00]	0.625000E+00	[ -4.950000000003E+00, 6.566666666670E+00]
12	-0.106666E+01	[ -1.012000000001E+02, 9.706666666673E+01]	-0.200000E+01	[ -1.012000000001E+02, 9.706666666673E+01]
13	-0.170666E+02	[ -1.747200000001E+03, 1.681066666668E+03]	-0.200001E+01	[ -1.747200000001E+03, 1.681066666668E+03]
14	-0.273066E+03	[ -3.000320000002E+04, 2.894506666669E+04]	-0.320001E+02	[ -3.000320000002E+04, 2.894506666669E+04]
15	-0.436906E+04	[ -5.1281920000004E+05, 4.9588906666671E+05]	-0.512000E+03	[ -5.1281920000004E+05, 4.9588906666671E+05]
16	-0.699050E+05	[ -8.7293952000013E+06, 8.4585130666680E+06]	-0.655360E+05	[ -8.7293952000013E+06, 8.4585130666680E+06]
17	-0.111848E+07	[ -1.4805893120002E+08, 1.4372481706669E+08]	0.432537E+08	[ -1.4805893120002E+08, 1.4372481706669E+08]
18	-0.178957E+08	[ -2.5031606272004E+09, 2.4338148010671E+09]	0.000000E+00	[ -2.5031606272004E+09, 2.4338148010671E+09]
19	-0.286331E+09	[ -4.2198053683207E+10, 4.1088520465074E+10]	0.536870E+09	[ -4.2198053683207E+10, 4.1088520465074E+10]
20	-0.458129E+10	[ -7.095285972993E+11, 6.9177606580919E+11]	0.000000E+00	[ -7.095285972993E+11, 6.9177606580919E+11]
21	-0.733007E+11	[ -1.1902213370677E+13, 1.1618172866835E+13]	-0.687194E+11	[ -1.1902213370677E+13, 1.1618172866835E+13]
22	-0.117281E+13	[ -1.9923150695305E+14, 1.9468685889157E+14]	-0.109951E+13	[ -1.9923150695305E+14, 1.9468685889157E+14]
23	-0.187650E+14	[ -3.3284415996041E+15, 3.2557272306205E+15]	0.000000E+00	[ -3.3284415996041E+15, 3.2557272306205E+15]
24	-0.300240E+15	[ -5.5506865407351E+16, 5.4343435503614E+16]	0.281475E+15	[ -5.5506865407351E+16, 5.4343435503614E+16]
25	-0.480384E+16	[ -9.2413864353659E+17, 9.0552376507680E+17]	0.450360E+16	[ -9.2413864353659E+17, 9.0552376507680E+17]
26	-0.768614E+17	[ -1.5362679048889E+19, 1.5064840993532E+19]	0.450360E+16	[ -1.5362679048889E+19, 1.5064840993532E+19]
27	-0.122978E+19	[ -2.5502623681908E+20, 2.5026082793337E+20]	0.720575E+17	[ -2.5502623681908E+20, 2.5026082793337E+20]
28	-0.196765E+20	[ -4.2279937416950E+21, 4.1517471995237E+21]	0.115292E+19	[ -4.2279937416950E+21, 4.1517471995237E+21]
29	-0.314824E+21	[ -7.0009083108556E+22, 6.8789138433814E+22]	0.184467E+20	[ -7.0009083108556E+22, 6.8789138433814E+22]
30	-0.503719E+22	[ -1.1579242615999E+24, 1.1384051468040E+24]	0.295147E+21	[ -1.1579242615999E+24, 1.1384051468040E+24]

Tabela 7.2 Tabulação dos resultados em Pascal XSC para o programa do exemplo 7.1.5

N	Pascal-XSC		PC-486	
	Real	Exata	Intervalar	Intervalar exata
1	0.100000E+01	0.100000E+01	[9.999999999998E-001, 1.000000000001E+000]	[1.0, 1.0]
2	0.100000E+01	0.100000E+01	[9.999999999998E-001, 1.000000000001E+000]	[1.0, 1.0]
3	0.100000E+01	0.100000E+01	[9.999999999995E-001, 1.000000000001E+000]	[1.0, 1.0]
4	0.100000E+01	0.100000E+01	[9.999999999990E-001, 1.000000000001E+000]	[1.0, 1.0]
5	0.100000E+01	0.100000E+01	[9.99999999998E-001, 1.000000000001E+000]	[1.0, 1.0]
6	0.100000E+01	0.100000E+01	[9.99999999996E-001, 1.000000000001E+000]	[1.0, 1.0]
7	0.100000E+01	0.100000E+01	[9.99999999994E-001, 1.000000000001E+000]	[1.0, 1.0]
8	0.100000E+01	0.100000E+01	[9.99999999992E-001, 1.000000000001E+000]	[1.0, 1.0]
9	0.100000E+01	0.100000E+01	[9.99999999990E-001, 1.000000000001E+000]	[1.0, 1.0]
10	0.100002E+01	0.100000E+01	[9.96E-001, 1.01E+000]	[1.0, 1.0]
11	0.999774E+00	0.100000E+01	[9.3E-001, 1.1E+000]	[1.0, 1.0]
12	0.991961E+00	0.100000E+01	[7.1E-002, 2.5E+000]	[1.0, 1.0]
13	0.124196E+01	0.100000E+01	[1.7E+001, 3.1E+001]	[1.0, 1.0]
14	-4.75804E+00	0.100000E+01	[3.1E+002, 5.4E+002]	[1.0, 1.0]
15	2.72420E+01	0.100000E+01	[5.1E+003, 9.8E+003]	[1.0, 1.0]
16	2.72420E+01	0.100000E+01	[8.2E+004, 1.8E+005]	[1.0, 1.0]
17	2.46032E+04	0.100000E+01	[1.3E+006, 3.1E+006]	[1.0, 1.0]
18	-2.37541E+05	0.100000E+01	[2.1E+007, 5.3E+007]	[1.0, 1.0]
19	2.90819E+06	0.100000E+01	[3.7E+008, 9.1E+008]	[1.0, 1.0]
20	-1.14532E+08	0.100000E+01	[5.8E+009, 1.6E+010]	[1.0, 1.0]
21	-1.14532E+08	0.100000E+01	[9.5E+010, 2.7E+011]	[1.0, 1.0]
22	-8.70447E+09	0.100000E+01	[1.5E+012, 4.6E+012]	[1.0, 1.0]
23	-2.83582E+11	0.100000E+01	[2.2E+013, 8.0E+013]	[1.0, 1.0]
24	-5.78114E+12	0.100000E+01	[3.5E+014, 1.4E+015]	[1.0, 1.0]
25	-1.28926E+14	0.100000E+01	[5.5E+015, 2.4E+016]	[1.0, 1.0]
26	2.40435E+15	0.100000E+01	[8.4E+016, 3.9E+017]	[1.0, 1.0]
27	2.49223E+16	0.100000E+01	[1.3E+018, 6.7E+018]	[1.0, 1.0]
28	-1.91250E+17	0.100000E+01	[2.0E+019, 1.2E+020]	[1.0, 1.0]
29	2.11459E+18	0.100000E+01	[3.0E+020, 1.9E+021]	[1.0, 1.0]
30	-1.08566E+20	0.100000E+01	[4.6E+021, 3.2E+022]	[1.0, 1.0]

```

program teste1 (input,output);
use i_ari;
var
  s: array [0.. 244] of interval;
  N,CONSTA :integer;
  x,cont,y,oito_N :integer;
  soma :interval;
begin
for CONSTA:= 1 to 30 do
begin
S[0]:=intval(1);
y:= 1;
for N:= CONSTA downto -CONSTA do
begin
  S[y]:= intval(exp (N * ln(16)));
  y:= y+1;
end;
for N:= CONSTA  downto -CONSTA do
begin
  S[y]:= intval (-exp (N * ln(16)));
  y:= y+1;
end;
for N:= CONSTA downto -CONSTA do
begin
  S[y]:= intval(exp (N * ln(16)));
  y:= y+1;
end;
for N:= CONSTA downto -CONSTA do
begin
  S[y]:= intval(- exp (N * ln(16)));
  y:= y+1;
end;
oito_N := 8 * CONSTA + 4;
soma:= intval(0);
soma:= ## (for cont:=0 to (oito_N) sum (S[cont]));
  { writeln('S[', cont,'] ',S[cont]); }
writeln ('N: ',CONSTA,' Soma ',## ',soma);
end;
end.

```

Figura 7.24 Testes de somatórios em Pascal XSC



**program tabel1**

```

use basico
integer :: N, consta, x, cont, y
type (interval), dimension (0:244) :: S
type (interval) :: soma
consta = 1
do while (consta <= 30)
  y=0
  S(y) = sintpt(1.)
  y= y+1
  N= consta
  do while (consta >= -N)
    S(y) = sintpt (16.0 **N)
    N = N - 1
    y = y+1
  end do
  N= consta
  do while (consta >= -N)
    S(y) = sintpt(-(16.0 **N))
    N = N - 1
    y = y+1
  end do
  N= consta
  do while (consta >= -N)
    S(y) = sintpt (16.0 **N)
    N = N - 1
    y = y+1
  end do
  N= consta
  do while (consta >= -N)
    S(y) = sintpt(-(16.0 **N))
    N = N - 1
    y = y+1
  end do
  soma = sintpt(0.)
  do 202 cont=0,(8*consta +4)
    202 soma= soma + S(cont)
  write (6,210) consta
    210 format ('S =',i2, ' ')
  call swrite (6,soma)
  consta = consta + 1
end do
end

```

Figura 7.25 Testes de somatórios em Fortran 90

## 7.2 Validação do Uso da Matemática Intervalar

Uma vez verificada a biblioteca de rotinas intervalares, será validado seu uso para a resolução de problemas. Pretende-se, então validar o uso da matemática intervalar em supercomputadores vetoriais para a resolução de problemas. Foram escolhidos quatro problemas: resolução de equações algébricas, resolução de problemas instáveis representado pelo cálculo do determinante e da inversa de matrizes de Hilbert, cálculo da matriz inversa intervalar e a resolução intervalar de sistemas lineares.

### 7.2.1 Resolução de equação - Método de Newton Intervalar

Na resolução de equações, o método numérico mais conhecido é o método de Newton-Raphson (ou método da tangente). É um método iterativo. Um valor inicial é utilizado na função de iteração do método para gerar novas aproximações. Aplicando-se sucessivamente dos valores produzidos na fórmula, têm-se uma seqüência de valores produzidos que muitas vezes são convergentes. Quando há convergência, o limite é um dos zeros da equação. A função de iteração no método de Newton é dada pela fórmula (7.3), onde se têm que o valor atual é produzido pelo valor anterior menos o quociente do valor da função no ponto anterior pela derivada da função no ponto anterior.

$$x_{k+1} := x_k - f(x_k) / f'(x_k) \quad (7.3)$$

Uma generalização ingênua do método de Newton para o uso da matemática intervalar seria converter os valores reais por intervalos, e as avaliações da função e da derivada por extensões intervalares. Ou seja,  $x_0 \subseteq X$  e para  $f$  e  $f'$ , sejam  $F$  e  $F'$  avaliações intervalares com  $0 \notin F'(X_0)$ . Seja, ainda,  $\alpha$  um zero de  $F(X)$ , sendo  $\alpha$  único no intervalo.

$$X_{k+1} := X_k - F(X_k) / F'(X_k) \quad (7.4)$$

Este método é divergente para todo intervalo inicial  $X_0 \subseteq X$ , com  $\alpha \in X$ ,  $X_0 \neq \alpha$ , pois, verificando-se o diâmetro do intervalo, nota-se que o intervalo cresce a cada iteração, ou seja  $d(X_{k+1}) \geq d(X_k)$ , logo a seqüência de intervalos  $\{ X_k \}$  é divergente.

Com isto, verificou-se a necessidade de uma adaptação nos métodos numéricos na passagem da aritmética de ponto-flutuante para a aritmética intervalar. Para ilustrar esta passagem, a seguir será considerada uma das versões do método de Newton Intervalar, conhecido como método de Newton Intervalar Simplificado.

Sejam  $X$  e  $X_0$  intervalos, se:  $f \in F(X)$ ;  $X_0 \subseteq X$ ; o valor  $\alpha$  é zero de  $f$  determinado em  $X$ ,  $\alpha$  é único no intervalo; e se  $M$  é a avaliação intervalar  $F'$  em  $X_0$ . Então, define-se o operador Newtoniano pela fórmula (7.5), onde  $N_i$  e  $M$  são intervalos.

$$N_i(x) = x - f(x)/M, \text{ onde } 0 \notin M. \quad (7.5)$$

O método de Newton Intervalar simplificado é dado por:

$$X_{k+1} := Ni(X_k) \cap X_k \text{ para } k=1,2,\dots,n \quad (7.6)$$

Observa-se a propriedade: se  $x_0 \in X_0$  e  $Ni(x_0) \cap X_0 = \emptyset$ , então não existe raiz real de  $f$  em  $X_0$ . Observa-se, ainda, que são as variações na definição do operador Newtoniano que determinam e definem as versões do método de Newton intervalar. Como exemplo será calculado o zero contido no intervalo  $[2, 3]$  da equação algébrica dado por:  $f(x) = \sqrt{x} + (x+1) \cdot \cos(x) = 0$ , cuja derivada é calculada por  $f'(x) = 1/(2\sqrt{x}) + \cos(x) - (x+1)\sin(x)$ . O valor de  $\alpha$  é 2.059045253415145.

```

program inewt(input, output);
use i_ari;
var x,y:interval;
    i:integer;
function f(r:real):interval;
var x:interval;
begin
x:=r;
f:=sqrt(x)+(x+1)*cos(x)
end;
function deriv(x:interval):interval;
begin
deriv:=1/(2*sqrt(x))+cos(x)-(x+1)*sin(x)
end;
function criter(x:interval):boolean;
begin
criter:=((sup(f(inf(x)))*inf(f(sup(x))))<0) and not(0 in deriv(x)));
end;
begin
y:=intval(2,3);
for i:=1 to 6 do
begin
if criter(y) then
repeat
x:=y;
writeln(x);
y:=(mid(x)-f(mid(x))/deriv(x))*x;
until x=y
else
writeln('critério nao satisfeito');
end;
end.

```

#### Solução em Pascal XSC

[	2.0E+000,	3.0E+000	]
[	2.0E+000,	2.3E+000	]
[	2.05E+000,	2.07E+000	]
[	2.05903E+000,	2.05906E+000	]
[	2.059045253413E+000,	2.059045253417E+000	]
[	2.059045253415143E+000,	2.059045253415145E+000	]

Fig 7.26 Programa e execução do método de Newton intervalar simplificado em Pascal XSC

```

program newton
use basico
use mvi
use aplic
type(interval) :: x,y
integer I,J
  print *,'Metodo de Newton em Fortran 90'
  print *
  y=sintval(2.0,3.0)
  do I=1, 10
    if (criter(y)) then
      do
        x=y
        call swrite(6,x)
        y=sinter((rsubs(smed(x),(f(smed(x))/deriv(x))),x)
        if (x==y) then
          exit
        end if
      end do
    else print *,'Critério nao satisfeito!'
    end if
  end do
contains
function f (r) result (c)
real, intent(in) :: r
type(interval) :: c
type(interval) :: x
  x = sintpt(r)
  c = ssqrt(x)+(x+sintpt(1.0))*cos(x)
end function f
function deriv (x) result (c)
type(interval), intent(in) :: x
type(interval) :: c
  c=sinv(sintpt(2.0)*ssqrt(x))+cos(x)-(x+sintpt(1.0))*sin(x)
end function deriv
function criter (x) result (c)
type(interval), intent(in) :: x
logical :: c
  c=((ssup(f(sinf(x)))*sinf(f(ssup(x)))<0).AND.(NOT.(rins(0.0,deriv(x))))
end function criter
end program newton

```

#### Metodo de Newton em Fortran 90

[ 2.0000000000000E+00,	3.0000000000000E+00]
[ 2.0000000000000E+00,	2.2181371829539E+00]
[ 2.0514014623809E+00,	2.0647263299078E+00]
[ 2.0590377919369E+00,	2.0590530112333E+00]
[ 2.0590452534138E+00,	2.0590452534165E+00]
[ 2.0590452534151E+00,	2.0590452534152E+00]
[ 2.0590452534151E+00,	2.0590452534152E+00]

Fig 7.27 Programa e execução do método de Newton intervalar em Fortran 90



## 7.2.2 Matriz de Hilbert - inversa e determinante

As matrizes de Hilbert são matrizes onde todos os elementos são menores ou iguais a um, sendo calculados pela fórmula:

$$H_n = h[i,j] = [ 1 / ( i+j-1 ) ] \text{ para } i=1,\dots,n \text{ e } j=1,\dots,n. \quad (7.7)$$

As matrizes Hilbertianas costumam ser mal-condicionadas. Isso quer dizer que, tendo a matriz  $H_n$  elementos no máximo igual à unidade, a inversa,  $H_n^{-1}$ , terá elementos cujos valores são elevados. As matrizes Hilbertianas são usadas algumas vezes com o objetivo de submeter os programas a testes para verificar a amplitude dos erros de arredondamento.

O determinante de uma matriz Hilbertiana pode ser calculado diretamente pela expressão (7.8) e o determinante da matriz inversa será o recíproco do determinante, ou seja,  $\det(H_n^{-1}) = 1/\det(H_n)$ .

$$\det(H_n) = \frac{[ 1! * 2! * \dots * (n-1)! ]^3}{[ n! * (n+1)! * \dots * (2n-1)! ]} \quad (7.8)$$

Foram desenvolvidos programas em Pascal XSC e em Fortran 90 que calculam: matrizes de Hilbert de ordem  $n$  e suas inversas, o determinante pela fórmula (7.8) e como produto da diagonal principal da matriz triangularizada e de sua inversa, a norma euclidiana da matriz e inversa e o condicionamento. Os valores dos determinantes foram comparados pela fórmula do número de algarismos significativos (conforme [CLA89]). A ordem máxima calculada em Pascal XSC foi 14, pois para  $n=15$  ocorre *overflow*, já em Fortran a ordem máxima é maior. Devido ao condicionamento, a ordem máxima que pode ser realmente resolvida é da ordem 11, pois o condicionamento chega a ordem de  $10^{11}$ .

```

program h;
use mv_ari;
  matinv;
var n: integer;
    cond,na,ab:array[2..14] of real;
    norm:real;
const tamanho_max=14;
function fat(i:integer):real; {calcula o fatorial de i}
var j :integer;
    w:real;
begin
  w:=1;
  for j:= 1 to i do
    w:=w*j;
  fat:=w;
end;
procedure detformula (n:integer;var det:real);{acha o determinante pela*}
var i:integer;
    x,y:real;
begin
  x:=1;
  for i:= 1 to n-1 do
    x:=fat(i)*x;
  x:=x*x*x;
  y:=1;
  for i:= n to 2*n-1 do

```

```

    y:=fat(i)*y;
det:= x/y;
end;
procedure Detmega ( a : rmatrix;          (*encontra o determinante *)
                    var detm : real; n : integer);(*pelo metodo de Gauss *)
var c,i,j: integer;
    w :real;
begin
for c := 1 to n-1 do
    for i := c+1 to n do
        for j := c+1 to n do
            a[i,j]:= a[i,j]-(a[i,c]*a[c,j])/a[c,c];
        w:=1;
        for i :=1 to n do
            w:=a[i,i]*w;
        detm:=w;
    end;
end;
procedure norma (a:rmatrix; var norm:real;n:integer);(*encontra a norma da*)
var i,j:integer;          (*matriz *)
    aux,maior:real;
begin
maior:=0;
for i:= 1 to n do
    begin
    aux:=0;
    for j := 1 to n do
        aux:=sqr(a[i,j])+aux;
    if aux>maior then
        maior:=aux;
    end;
end;
norm:=sqrt(maior);
end;
procedure main(n:integer;var norm:real);    (*procedure principal *)
const
    eps=1e-16; k=3e-1;
var
i,j:integer;
det,detme,a,b:real;
H: rmatrix[1..n,1..n];
begin
for i := 1 to n do
    for j := 1 to n do
        h[i,j]:=1/(i+j-1);
    detformula(n,det);
    detmega(H,detme,a);
    norma(H,norm,n);
b:=eps+abs((det-detme)/det);
b:=log10(b);
a:=(k+b);
writeln(' ',det,' ',detme,' ',a);
end;
procedure inversa_main(n:integer;var norm:real);(*procedure principal inver*)
const
    eps=1e-16; k=3e-1;
var
i,j:integer;
det,detme,a,b:real;
H: rmatrix[1..n,1..n];
begin
for i := 1 to n do
    for j := 1 to n do
        b[i,j]:=1/(i+j-1);
    detformula(n,det);
det:=1/det;
matinv(H,H,i);
detmega(H,detme,a);
norma(H,norm,n);
b:=eps+abs((det-detme)/det);
b:=log10(b);
a:=(k+b);
writeln(' ',det,' ',detme,' ',a);
end;

```

```

end;
begin
  writeln('H(n) determinante formula      determinante metodo      asc');
  for n := 2 to tamanho_max do
    begin
      writeln('
      matric(n,norm):
      mat[n]:=norm;
      cond[n]:=norm;
      end;
    writeln('INVERSA');
    writeln('H(n) determinante formula      determinante metodo      asc');
    for n := 2 to tamanho_max do
      begin
        inversa:=matric(n,norm);
        mat[n]:=norm;
        cond[n]:=cond[n]*norm;
      end;
    writeln('
    writeln('DIM NORMA DE A      NORMA DE A-1      CONDICIONAMENTO');
    for n := 2 to tamanho_max do
      writeln('
      writeln('
      end;
    end;
  end;
end;

```

```

H(n) determinante formula      determinante metodo      asc
2 8.333333333333333E-002 8.333333333333333E-002 1.527424827315604E+001
3 4.629629629629630E-004 4.629629629629612E-004 1.410185577712623E+001
4 1.653439153439153E-007 1.653439153439159E-007 1.416068877210809E+001
5 3.749295132515087E-012 3.749295132502843E-012 1.118602268834914E+001
6 5.367299887358688E-018 5.367299886386974E-018 9.44221712277734E+000
7 4.835802623926117E-025 4.835802601447726E-025 8.032703323551813E+000
8 2.737050113791513E-033 2.737050098354242E-033 7.948712200575508E+000
9 9.720234311924999E-043 9.720269612220532E-043 5.139898392659116E+000
10 2.164179226431492E-053 2.164491737922188E-053 3.540427233512815E+000
11 3.01909534449335E-065 3.029564417843071E-065 2.159968167741181E+000
12 2.637780651253547E-078 2.784882186990695E-078 9.536214715026785E+001
13 1.442896518791137E-092 1.416307372587583E-092 1.434530793562271E+000
14 4.940314914590828E-108 3.096463048404885E-106 -2.103985613527100E+000

```

#### INVERSA

```

H(n) determinante formula      determinante metodo      asc
2 1.200000000000000E+001 1.200000000000000E+001 1.510222959448430E+001
3 2.160000000000000E+003 2.160000000000000E+003 1.411009938434061E+001
4 6.048000000000000E+006 6.047999999999994E+006 1.371559474289697E+001
5 2.667168000000000E+011 2.667168000000704E+011 1.22782821661498E+001
6 1.863134203392000E+017 1.863134203695574E+017 9.487979184660935E+000
7 2.067909047925770E+024 2.067909046344854E+024 8.816622510794135E+000
8 3.653568471257345E+032 3.653568566461989E+032 7.284059115022804E+000
9 1.028781784378570E+042 1.028779721925472E+042 5.3979391844991396E+000
10 4.620689394791469E+052 4.620341192771316E+052 3.822875489570427E+000
11 3.12250489706341E+064 3.305566007868199E+064 2.3950542534742E+000
12 3.791065794363045E+077 4.020135441663047E+077 9.187937745798156E+001
13 6.930503934113053E+091 3.025751979824074E+093 -1.949903839899107E+000
14 2.024162461883916E+107 3.081821756255484E+105 -2.9333369398765924E+001

```

#### DIM NORMA DE A

```

2 1.118033988749895E+000 NORMA DE A-1 CONDICIONAMENTO
3 1.16666666666667E+000 1.341640786499874E+001 1.500000000000001E+001
4 1.193151755273030E+000 8.184008797648440E+003 3.0999032106965024E+002
5 1.209797962930634E+000 2.333369546383791E+005 2.822905723979487E+005
6 1.221224340114823E+000 6.370687220345001E+006 7.780038296743770E+006
7 1.229551565471817E+000 1.953493898253675E+008 2.401920497096196E+008
8 1.235889174705481E+000 6.183101117111727E+009 7.641627736747749E+009
9 1.240873777290237E+000 1.869020938930798E+011 2.319219072325605E+011
10 1.244896674895769E+000 5.640516082773216E+012 7.021859716140482E+012
11 1.248211598238239E+000 1.858575818982882E+014 2.319895893459567E+014
12 1.250990263119942E+000 5.813468159076568E+015 7.272592061962603E+015
13 1.253353022170616E+000 1.027609352428242E+018 1.287957253636195E+018
14 1.255386728861088E+000 9.027671703452022E+016 1.133321924902844E+017

```

Fig. 7.28 Programa e execução da matriz de Hilbert - determinante

Tab. 7.3 - Cálculo do determinante de matrizes de Hilbert em Fortran 90.

ord	Det A - MEG	Det Form	Det Inv MEG	Det Inv Form
1	1.000000000000E+00	1.000000000000E+00	1.000000000000E+00	1.000000000000E+00
2	8.333333333333E-02	8.333333333333E-02	1.200000000000E+01	1.200000000000E+01
3	4.629629629628E-04	4.629629629628E-04	2.160000000007E+03	2.160000000001E+03
4	1.653439153436E-07	1.653439153439E-07	6.048000000009E+06	6.048000000182E+06
5	3.749295132383E-12	3.749295132515E-12	2.667168000093E+11	2.6671680001025E+11
6	5.367299866394E-18	5.367299887358E-18	1.8631342106692E+17	1.8631341987015E+17
7	4.8358024711384E-25	4.835802623926E-25	2.0679091132616E+24	2.0679090856291E+24
8	2.7370607140255E-33	2.7370501137915E-33	3.6535543215235E+32	3.6535721498905E+32
9	9.7218618291622E-43	9.7202343119250E-43	1.0286095580996E+42	1.0284742021290E+42
10	2.1721428229355E-53	2.1641792264315E-53	4.6037488393538E+52	4.6072306732859E+52
11	3.1685317057543E-65	3.0190953344493E-65	3.1560359588131E+64	2.8426426696385E+64
12	4.5323007754448E-78	2.6377806512535E-78	2.2063849015004E+77	2.3135544694845E+77
13	3.7227068253055E-91	1.4428965187911E-92	2.6862174404989E+90	4.3860455054512E+88
14	-3.3764962818327-104	4.9403149145908-108	-2.9616499368903+103	1.3035805984132+104
15	-1.1105610715367-117	1.0585427430697-124	-9.0044575271877+116	-4.2672729448600+117

Tab 7.2 - Testes de condicionamento com matrizes de Hilbert em Fortran 90.

Ord	Norma de A	Norma da Inversa	Condicionamento
1	1.000000000000E+00	1.000000000000E+00	1.000000000000E+00
2	1.2692955176440E+00	1.5231546211728E+01	1.9333333333333E+01
3	1.4136241839093E+00	3.7220558835150E+02	5.2615882107988E+02
4	1.5097340998183E+00	1.0342081802074E+04	1.5613793559701E+04
5	1.5809062632720E+00	3.0416042253600E+05	4.8084911702662E+05
6	1.6370223933024E+00	9.2356630319216E+06	1.5118987200251E+07
7	1.6831324888436E+00	2.8622065136273E+08	4.8174727728658E+08
8	1.7221431395613E+00	8.9966739394299E+09	1.5493560303659E+10
9	1.7558719097166E+00	2.8569137356625E+11	5.0163745769335E+11
10	1.7855271226510E+00	9.1041337888890E+12	1.6255677808305E+13
11	1.8119508009081E+00	2.6798243074927E+14	4.8557098002544E+14
12	1.8357520373814E+00	2.5754902930767E+15	4.7279615527716E+15
13	1.8573850093856E+00	3.6227875840492E+15	6.7289113508012E+15
14	1.8771970077480E+00	1.1928122713875E+16	2.2391436266536E+16
15	1.8954592794982E+00	1.1388433337653E+16	2.1586311648801E+16
16	1.9123875473542E+00	2.7546368422238E+16	5.2679331945519E+16
17	1.9281560786634E+00	2.8370059407055E+16	5.4701902497755E+16
18	1.9429075852587E+00	8.5133757000388E+16	1.6540702223762E+17
19	1.9567603489225E+00	6.0168122880478E+16	1.1773459712162E+17
20	1.9698134528699E+00	1.5603841408125E+17	3.0736656722173E+17
21	1.9821506908640E+00	7.9133223151918E+16	1.5685397294087E+17
22	1.9938435344365E+00	6.2259520050445E+16	1.2413574150970E+17
23	2.0049534171210E+00	1.1206602128676E+17	2.2468715232205E+17
24	2.0155335154173E+00	5.5732063097010E+16	1.1232984105538E+17
25	2.0256301534840E+00	3.7800688515723E+17	7.6570214479907E+17



### 7.2.3 Cálculo da Matriz inversa

O cálculo da matriz inversa apresentado neste item foi descrito por E.Hansen em [HAN65]. O cálculo da matriz inversa de  $A$ , onde  $A$  é uma matriz intervalar, inicia pelo cálculo da matriz pontual centro de  $A$  (veja Figura 7.30 - função **matcentro**). Utilizando a matriz centro de  $A$ , denominada  $AC$ , calcula-se a aproximação da inversa de  $AC$ .

No algoritmo da Figura 7.29 a fim de poupar memória, as matrizes  $L$  e  $U$  não são explicitamente alocadas. Assim, qualquer aparecimento dos componentes de  $L$  e  $U$  nas equações (7.5) a (7.7) serão substituídos por elementos correspondentes de  $A$ . O procedimento de inversão será finalizado se o elemento pivô for menor do que *tiny*. Caso contrário poderia ocorrer *overflow*, exceto no tempo de processamento da divisão pelo elemento pivô. O valor de *tiny* depende: do domínio exponencial e do formato de ponto-flutuante usado. A etapa de substituição progressiva do algoritmo ganha vantagem de liberar elementos zero no vetor unitário  $e^{(k)}$ .

Utilizando a matriz  $A$  e a aproximação da inversa da matriz centro de  $A$ , calcula-se a matriz inversa intervalar de  $A$  (veja Figura 7.30 - função **imatinv**). Este cálculo é realizado em 4 etapas:

- 1) Calcula-se o erro existente entre a multiplicação da matriz  $A$  pela aproximação da inversa da matriz centro de  $A$  menos a matriz identidade. Este cálculo é descrito no passo 1 da função **imatinv**, ou seja,  $E = I - A*B$ , onde  $I$  é a matriz identidade e  $B$  é a aproximação da inversa da matriz centro de  $A$ .
- 2) No passo 2 da função **imatinv**, são realizadas operações que destinam-se ao cálculo da matriz  $P$ , matriz esta que será utilizada para inflacionar a matriz inversa intervalar de  $A$ . Deve-se salientar que os valores de  $P$  são obtidos através da matriz erro  $E$  e seus valores são menores que o menor valor encontrado na matriz  $E$ .
- 3) No passo 3 da função **imatinv** é calculada a matriz  $S$ , esta matriz é obtida pela seguinte multiplicação:  $S = I+E*(I+E*(I+E))$ . Esta matriz será utilizada no passo seguinte para o cálculo da matriz inversa intervalar.
- 4) No passo 4 da função **imatinv** é calculada a matriz inversa intervalar de  $A$  através da multiplicação da aproximação da inversa do centro de  $A$  pela matriz  $S$  calculada no passo anterior.

**Algoritmo 7.1: {Procedure} *MatInv*(A,R,Err)**

```

1.  $Tiny := 10^{-200}$ ;  $p = (1, 2, \dots, n)^T$ ; Err := "No error";
2. If (n=2)           {Caso especial: n=2}
   then
     Det := (a11 · a22 - a21 · a12);
     if (|det| < Tiny) then return Err := "Matriz é provavelmente singular!";
     r11 := a22 / Det; r12 := -a12 / Det;
     r21 := -a21 / Det; r22 := a11 / Det;
3. {Caso usual: n≠2}
   else           {LU-Fatorização}

   for i=1 to n do
      $v_k := (a_{ki} - \sum_{j=1}^{i-1} a_{kj} \cdot a_{ji}); \quad (k=1, \dots, n);$ 
     |vi| = max {|vk|}, i ≤ k ≤ n;
     if (|vi| < Tiny) then return Err := "Matriz é provavelmente singular!";
     if (j ≠ i) then
       swap (pi, pj); swap (vi, vj); swap (aj, ai);
       aij := vj;

     for k=i+1 to n do
        $a_{ik} := (a_{ik} - \sum_{j=1}^{i-1} a_{ij} \cdot a_{jk});$ 
       aki := vk / vi;

4. {Cálculo da inversa coluna por coluna}
   for k = 1 to do
     (a) {Substituição progressiva, ver (7.6)}
       Encontra índice l com pl = k;
       yi := 0; (i= 1, ..., l-1)
       yl := 1;

        $y_i := - (\sum_{j=1}^{l-1} a_{ij} \cdot y_j); \quad (i=1+1, \dots, n)$ 

     (b) {retrossubstituição, ver (7.7)}

        $x_i := (y_i - \sum_{j=i+1}^n a_{ij} \cdot x_j) / a_{ii}; \quad i=n, n-1, \dots, 1,$ 

        $r_{*,k} := x; \quad \{k\text{-ésima coluna de R}\}$ 

5. {Retorna : inversa aproximada R e código de erro Err} return R, Err;

```

Fig. 7.29 Algoritmo 7.1 Procedure *MatInv*

```

Algoritmo 7.2: module iselas;
use i_ari, mvi_ari, mv_ari;
{Função que calcula a inversa intervalar de A}
global function imatinv (var A:imatrix; var B:rmatrix) : imatrix[1..ubound(A),1..ubound(A)];
var
  C, E, S, P: imatrix[1..ubound(A),1..ubound(A)];
  I, Z: rmatrix[1..ubound(A),1..ubound(A)];
  k,j,ordem:integer;
  r,bl,sumc: real;
begin
  ordem := ubound(A);
  {Definição da matriz identidade}
  I := id(I);
  {Passo 1}
  E := I - A*B;
  {Passo 2}
  for k:= 1 to ordem do
    for j:= 1 to ordem do
      if (abs(inf(E[k,j])) >= abs(sup(E[k,j]))) then Z[k,j]:= abs(inf(E[k,j]))
      else Z[k,j]:= abs(sup(E[k,j]));
    sumc:= 0; r:=0;
    for k:= 1 to ordem do
      for j:= 1 to ordem do
        sumc:= sumc + Z[j,k];
      if (r < sumc) then r:= sumc;
    bl:= ((r*r)*r)/(1-r);
    for k:= 1 to ordem do
      for j:= 1 to ordem do
        P[k,j]:= intval(-bl,bl);
  {Passo 3}
  S:= I + E*(I + E*(I + E));
  {Passo 4}
  C:= B*(S + P);
  imatinv := C;
end;
{Função que calcula a matriz pontual centro de A}
global function matcentro (var A:imatrix) : rmatrix[1..ubound(A),1..ubound(A)];
var
  AC : rmatrix[1..ubound(A),1..ubound(A)];
  k,j,ordem : integer;
begin
  ordem := ubound(A);
  for k:= 1 to ordem do
    for j:= 1 to ordem do
      AC[k,j]:= mid(A[k,j]);
  matcentro := AC;
end;
end.

```

Fig. 7.30 Módulo iselas em Pascal XSC

Estes algoritmos foram implementados em Fortran 90 e em Pascal XSC. Um exemplo de execução consta no próximo ítem, onde se trata da resolução intervalar de sistemas lineares. Para a verificação da inversa, ela foi multiplicada pela matriz A. O resultado obtido foi uma matriz intervalar que contém a identidade pontual.

### 7.2.4 Resolução de um sistema de equações

O procedimento para resolver o sistema de equações lineares  $Ax = b$  tem dois passos principais: cálculo de uma solução aproximada  $\tilde{x}$  e, então, cálculo da inclusão para o erro desta aproximação. É um método *a posteriori*. Para obter uma boa aproximação, usa-se (7.3) para o cálculo de uma iteração residual real simples (ver passo 2 da Figura 7.34). Visto que esta pré-iteração será seguida de um passo de verificação, podem-se aplicar algumas considerações heurísticas para melhorar o cálculo deste valor. Tenta-se prever se algumas das componentes da solução exata podem se anular. Sejam  $x$  e  $y$  duas iterações sucessivas. A heurística é que qualquer componente de  $y$  que é diminuída em mais do que  $n$  ordens de magnitude é uma boa candidata para entrada de um zero na solução exata. No algoritmo da Figura 7.31, aquelas componentes da iteração nova  $y$  são alteradas para zero. Isto revelou que  $n = 5$  é um bom valor para uso prático.

Algoritmo 7.3: {Procedure} *CheckForZeros*( $x, y$ )

1. {Inicialização} Factor :=  $10^5$ ;
2. {Atualizando entradas que possivelmente sejam zeros de  $y$ , i.e. aquelas entradas com  $|y_i/x_i| \leq 10^{-5}$ }  
if (  $|y_i|$ , Factor <  $|x_i|$  ) then  $y_i := 0$ ; ( $i = 1, \dots, n$ )
3. {Retorna o  $y$  atualizado} return  $y$ ;

Fig 7.31 Rotina *Checkforzeros*

Em geral, a iteração real será finalizada se o erro relativo de qualquer componente das duas iterações sucessivas for menor do que  $\delta$ . Caso contrário, interrompe-se a iteração depois de  $k_{max}$  passos. Aqui, usa-se  $\delta = 10^{-12}$  porque o formato de ponto-flutuante do PASCAL-XSC tem cerca de 12 dígitos corretos de mantissa da solução aproximada. O algoritmo 7.4 descreve o critério de parada para a iteração. Aqui faz-se uso de outra heurística. Se as componentes das duas iterações sucessivas diferem em sinal, ou se uma delas se anula, toma-se isto como um indicador da presença de um zero de entrada na solução exata. Assim, implicitamente, ajusta-se o erro relativo para zero.

**Algoritmo 7.4: {Function} Accurate(x,y)**

- {Inicialização : exatidão relativa desejada}  $\delta := 10^{-12}$  ;
- {Verifique se o erro relativo de y com respeito a x é menor ou igual a  $\delta$ }  
 $i := 1$  ;  
 repeat  
   if (  $x_i \cdot y_i \leq 0$  )  
     then ok := True {Erro relativo implicitamente aplicado a 0}  
     else ok := (  $|y_i - x_i| \leq \delta \cdot |y_i|$  ) ;  
    $i = i + 1$  ;  
 until ( not ok ) or (  $i > n$  ) ;
- {Retorna valor do flag ok} return Accurate := ok ;

Fig 7.32 Função *Accurate*

Para o passo de verificação, recorre-se à seção 7.1.2. Para acelerar o procedimento, as iterações são inflacionadas no começo de cada *loop* de iteração. Usou-se o valor constante de  $\epsilon=1000$  para a  $\epsilon$ -inflação. No algoritmo 7.5, a iteração pára depois de  $p_{\max}$  passos.

**Algoritmo 7.5: {Procedure} VerificationStep([x],[y],[C], IsVerified)**

- {Inicialização}  $\epsilon := 1000$  ;  $p_{\max} := 3$  ;  $p := 0$  ;  $[x]^{(0)} := [z]$  ;
- repeat  
    $[x]^{(p)} := [x]^{(p)} \triangleright \triangleleft \epsilon$  ; { $\epsilon$ -Inflação}  
    $[x]^{(p+1)} := \diamond ([z] + [C] \cdot [x]^{(p)})$  ;  
    $IsVerified := ([x]^{(p+1)} \subseteq [x]^{(p)})$  ;  
    $p := p + 1$  ;  
 until  $IsVerified$  or (  $p \geq p_{\max}$  ) ;  
    $[x] := [x]^{(p)}$  ;
- {Retorna a inclusão [x] e flag IsVerified} return [x], IsVerified;

Fig. 7.33 Rotina de verificação

Agora será fornecido o algoritmo completo baseado no algoritmo 7.3 e 7.5 para calcular uma inclusão verificada da solução de um sistema de equações lineares. O procedimento falha se o cálculo da inversa aproximada  $R$  falha ou se a inclusão no interior não pode ser estabelecida. Um número de condicionamento é retornado pela variável *Cond*, é necessária uma inclusão estreita  $[z]$  de  $z = R(b - A\bar{x})$  para iniciar o passo de verificação. Assim, no passo 3 primeiro calcula-se o resíduo  $d = (b - A\bar{x})$ . Se o cálculo for arredondado, uma inclusão deste erro é dada por  $[d] = \diamond (b - A\bar{x} - d)$ . Deste modo, têm-se  $b - A\bar{x} \in d + [d]$  e  $z \in \diamond (Rd + R[d])$ . No passo 4, obtém-se que  $\bar{x}$  é exata, mas não necessariamente a única solução do sistema se  $[z] = 0$ .



**Algoritmo 7.6: {Procedure} *LinSolve*( A,b,[x],Cond,Err)****1. {Cálculo de uma solução aproximada}**

Matinv(A,R,Err);

if (Err ≠ "No error") then return Err := "Matriz é provavelmente singular!"

Cond := ||A||<sub>∞</sub>.||R||<sub>∞</sub>;**2. {Iteração de resíduo real para uma solução aproximada}** $k_{\max} := 10; k := 0; \tilde{x}^{(0)} := R \cdot b;$ 

repeat

d := (b - A .  $\tilde{x}^{(k)}$ ); $\tilde{x}^{(k+1)} := (\tilde{x}^{(k)} + R \cdot d);$ CheckForZeros( $\tilde{x}^{(k)}, \tilde{x}^{(k+1)}$ );Success := Accurate( $\tilde{x}^{(k)}, \tilde{x}^{(k+1)}$ );

k := k + 1;

until Success or (k ≥  $k_{\max}$ ); $\tilde{x} = \tilde{x}^{(k)}$ ;**3. {Cálculo de inclusões [C] e [z] para C = I - RA e z = R(b - A  $\tilde{x}$ )}**[C] :=  $\diamond(I - R \cdot A)$ ;d := (b - A .  $\tilde{x}$ );[d] :=  $\diamond(b - A \cdot \tilde{x} - d)$ ;[z] :=  $\diamond(R \cdot d + R \cdot [d])$ ;**4. {Passo de verificação}**

if ([z] = 0) then

[x] :=  $\tilde{x}$ ; {Solução exata}

else

VerificationStep([x], [z], [C], IsVerified);

if (not IsVerified) then

Err := "Verificação falhou, o sistema está provavelmente mal-condicionado!";

return Cond, Err;

else

[x] :=  $\tilde{x} + [x]$ ; {Aproximação mais correção intervalar}

5. return [x], Cond, Err;

Fig 7.34 Rotina *linsolve*

Serão fornecidas as listagens dos módulos *matinv* para inversão de matriz e *linsys* para resolução de um sistema de equações lineares. Enfatiza-se que partes dos algoritmos da seção anterior, as quais são marcadas por (...) e  $\diamond(\dots)$  são implementadas usando expressões exatas. Códigos de erro são passados por parâmetros do tipo inteiro. O código de erro 0 significa que não ocorreu erro.

Como amostra de um dado de entrada, foi usado um sistema de Boothroyd/Dekker cujos elementos são inteiros definidos por

$$\tilde{a}_{ij} = \binom{n+i-1}{i-1} \binom{n-1}{n-j} \frac{n}{i+j-1}, \quad i, j = 1, \dots, n. \quad (7.9)$$

Para  $n = 10$ , o programa descrito pelo algoritmo 7.7 produz a seguinte solução:

**Entre a dimensão do sistema: 10**

**Entre matriz A:**

10	45	120	210	252	210	120	45	10	1
55	330	990	1848	2310	1980	1155	440	99	10
220	1485	4752	9240	11880	10395	6160	2376	540	55
715	5148	17160	4320	45045	40040	24024	9360	2145	220
2002	15015	51480	105105	140140	126126	76440	30030	6930	715
5005	38610	135135	280280	378378	343980	210210	83160	19305	2002
11440	90090	320320	672672	917280	840840	517440	205920	48048	5005
24310	194480	700128	1485120	2042040	1884960	1166880	466752	109395	11440
48620	393822	1432080	3063060	4241160	3938220	2450448	984555	231660	24310
92378	755820	2771340	5969040	8314020	7759752	4849845	1956240	461890	48620

**Entre vetor b: 1 2 3 4 5 6 7 8 9 10**

**Aproximação Naive de ponto-flutuante:**

```
3.769740075654227E -010
9.999999949022040E -001
-1.999999975192608E+000
2.999999904595484E+000
-3.999999691503490E+000
4.999999132005541E+000
-5.999997815384631E+000
6.999994975405571E+000
-7.999989266972989E+000
8.999978443156579E+000
```

**Solução verificada encontrada em:**

```
[ 0.000000000000000E+000, 0.000000000000000E+000 ]
[ 1.000000000000000E+000, 1.000000000000000E+000 ]
[-2.000000000000000E+000, -2.000000000000000E+000 ]
[ 3.000000000000000E+000, 3.000000000000000E+000 ]
[-4.000000000000000E+000, -4.000000000000000E+000 ]
[ 5.000000000000000E+000, 5.000000000000000E+000 ]
[-6.000000000000000E+000, -6.000000000000000E+000 ]
[ 7.000000000000000E+000, 7.000000000000000E+000 ]
[-8.000000000000000E+000, -8.000000000000000E+000 ]
[ 9.000000000000000E+000, 9.000000000000000E+000 ]
```

**Condição estimada: 1.1E+015**

Depois se efetua a iteração residual real do algoritmo 7.6, a solução aproximada já é exata! O passo de verificação é usado apenas para obter uma prova matemática deste fato. Em geral, a inclusão da solução não é um vetor intervalar “magro” ou “estrito”.

Para o cálculo de um sistema de equações onde as matrizes/vetores são intervalares, usa-se um método apresentado por Hansen em [HAN67]. De acordo com este método calcula-se a matriz inversa intervalar de  $A$ , conforme descrito no item 7.2.3 e, então, multiplica-se essa matriz inversa pela matriz de coeficientes  $B$  e o resultado obtido é a solução intervalar que contém a solução do sistema de equações.

```

Algoritmo 7.7: program hansen3 (input, output);
use i_ari, mvi_ari, matinv, mv_ari, iselas;
var
  A, AII: imatrix[1..3,1..3];
  AC, API: rmatrix[1..3,1..3];
  R, B: imatrix[1..3,1..1];
  coderro,k,j:integer;
begin
  A[1,1]:=-2; A[1,2]:=-20; A[1,3]:=-34;
  A[2,1]:=-30; A[2,2]:=23; A[2,3]:=1;
  A[3,1]:=-23; A[3,2]:=2; A[3,3]:=-2;
  B[1,1]:=1; B[2,1]:=2; B[3,1]:=3;
  {Calculo da matriz pontual centro de A}
  AC:= matcentro(A);
  {Calculo da aproximação da matriz pontual centro de A - Algoritmo 7.2, função matcentro}
  Matinv(AC,API,coderro);
  {Calculo da matriz inversa intervalar de A - Algoritmo 7.2, função imatinv}
  AII:= imatinv(A,API);
  {calculo do resultado do sistema de equações}
  R:= AII * B;
  writeln('Matriz A'); writeln(A);
  writeln('Matriz B'); writeln(B);
  writeln('Matriz Inversa de A'); writeln(AII);
  writeln('Matriz Resultado'); writeln(R);
end.

```

Fig 7.35 Programa Hansen3

Os resultados apresentados abaixo foram obtidos através da utilização das funções, procedimentos e programas apresentados nos algoritmos 7.1, 7.2 e 7.7. Essas funções, procedimentos e programas foram escritos em Pascal-XSC.

**Matriz A**

```

[-2.0E+000, -2.0E+000 ] | -2.0E+001, -2.0E+001 | | -3.4E+001, -3.40E+001 |
[-3.0E+001, -3.0E+001 ] | 2.3E+001, 2.3E+001 | | 1.0E+000, 1.0E+000 |
[-2.3E+001, -2.3E+001 ] | 2.0E+000, 2.0E+000 | | -2.0E+000, -2.0E+000 |

```

**Matriz B**

```

| 1.0E+000, 1.0E+000 |
| 2.0E+000, 2.0E+000 |
| 3.0E+000, 3.0E+000 |

```

**Matriz Inversa de A**

```

coluna 1
| 3.382663847780125E-003, 3.382663847780129E-003 |
| 5.849189570119800E-003, 5.849189570119805E-003 |
| -3.305144467935168E-002, -3.305144467935164E-002 |
coluna 2
| 7.61099365750528E-003, 7.61099365750529E-003 |
| 5.48273431994361E-002, 5.48273431994363E-002 |
| -3.269908386187458E-002, -3.269908386187454E-002 |
coluna 3
| -5.369978858350955E-002, -5.369978858350949E-002 |
| -7.202255109231858E-002, -7.202255109231850E-002 |
| 4.552501761804085E-002, 4.552501761804090E-002 |

```

**Matriz Resultado**

```
[ -1.424947145877380E-001, -1.424947145877377E-001 ]
[ -1.00563777307964E-001, -1.00563777307963E-001 ]
[ 3.81254404510217E-002, 3.81254404510220E-002 ]
```

**Resultado Pontual está contido no resultado intervalar:**

```
[-0.143]
[-0.101]
[ 0.0382]
```

**Matriz Resultado: A x Inversa de A é a identidade intervalar!****coluna 1**

```
[ 9.999999999999994E-001, 1.000000000000001E+000 ]
[ -1.6E-015, 1.7E-015 ]
[ -1.2E-015, 9.3E-016 ]
```

**coluna 2**

```
[-1.8E-016, 1.4E-016 ]
[ 9.999999999999992E-001, 1.000000000000001E+000 ]
[ -6.6E-016, 7.9E-016 ]
```

**coluna 3**

```
[ -9.2E-017, 1.1E-016 ]
[ -1.7E-016, 1.8E-016 ]
[ 9.999999999999994E-001, 1.000000000000001E+000 ]
```

### 7.3 Os resultados

Foi verificado, nos resultados produzidos pelos programas em Fortran 90, que no lugar do zero era produzido o valor 7.3344154702194E-2466 (em módulo). Com isto, verificou-se que a região de *underflow* ou o zero de máquina no Cray incluía este valor. O zero só é mantido na leitura. Após qualquer uma das operações aritméticas, o zero é armazenado pelo valor de *underflow*. Isto pode ser melhorado utilizando-se a rotina de impressão em formato de notação científica ao invés da rotina de impressão em formato livre do Fortran 90 para reais.

Verificou-se ainda que, em Fortran 90, os valores são arredondados para o número mais próximo de máquina, ou seja, o arredondamento simétrico, onde o dígito 15 da mantissa é analisado, caso seja um dos dígitos {0, 1, 2, 3, 4}. Então o número é truncado, mantendo os 14 primeiros dígitos significativos. Caso o dígito 15 da mantissa seja um dos dígitos {5, 6, 7, 8, 9}, então será adicionada uma unidade ao dígito 14 e truncado o número da mesma forma. Esta soma de uma unidade é feita em módulo. Isto poderia resultar na perda do valor exato no intervalo, portanto utilizaram-se as rotinas *infla-down* e *infla-up* de forma a simular os arredondamentos direcionados.

Na primeira versão destas rotinas tentou-se utilizar a rotina em fortran *nearst* que calcula o número de máquina mais próximo. Mas o resultado destas rotinas foi desastroso, pois o intervalo inflacionado perdia o valor exato em vários casos. Então projetaram-se novas rotinas que simulavam os arredondamentos direcionados. Esse procedimento baseou-se na idéia de somar ou subtrair uma unidade no décimo terceiro dígito menos significativo da mantissa. Este valor depende do sinal do número a ser inflacionado e do expoente. Pois na verdade, soma-se ou subtrai-se  $1.10^{13+u}$ , onde  $u$  é o expoente do valor a ser inflado. Desta forma, o resultado aproximou-se da definição dos arredondamentos monotônicos e obteve-se a garantia de que os resultados estariam sempre contidos dentro do intervalo.

Obtêm-se resultados com 13 a 14 dígitos significativos exatos com a biblioteca de rotinas intervalares, com a garantia de que o valor exato está contido no intervalo.

## 7.4 A avaliação do desempenho

Na questão da avaliação do desempenho, do ganho computacional com a utilização do supercomputador Cray Y-MP2E, há dois aspectos a serem considerados. O primeiro está relacionado à velocidade de processamento de 330 Megaflops por segundo por CPU, o que proporciona que os cálculos sejam efetuados muito rapidamente.

Por outro lado, deve-se que considerar o uso da vetorização. Utilizaram-se apenas a otimização e vetorização realizadas automaticamente pelo compilador F90, o que também proporcionou cálculos executados rapidamente. Este segundo aspecto, entretanto, tem alguns efeitos colaterais, como foi visto no produto escalar, onde a troca da ordem de parcelas em somatórios produz resultados diferentes. Com o uso de intervalos, este problema é controlado pois o resultado intervalar contém o valor exato. Pode acontecer que o diâmetro do intervalo seja muito grande, devido à falta da aritmética de alta exatidão, pois em Pascal XSC observou-se que o resultado exato pode ser obtido com o uso de expressões exatas e variáveis do tipo *dotprecision* como visto no ítem 7.1.5.

Resultados obtidos pela *libavi.a* são tão eficientes quanto os resultados obtidos pela biblioteca de rotinas voltadas para álgebra linear BLAS disponível na *libsci.a*, pois além de a utilizar, a *libavi.a* opera com intervalos que contêm a solução, proporcionando resultados com limites confiáveis.



## 8 CONCLUSÕES E CONSIDERAÇÕES

Neste capítulo é feito um resumo de toda a pesquisa desenvolvida, sendo caracterizados os resultados obtidos e problemas enfrentados. Considerando que este trabalho abriu um leque de problemas a serem estudados, os quais não foram todos abordados por estarem fora do escopo da proposta e desenvolvimento dessa Tese, foi feito um estudo geral desses problemas com o objetivo de proporcionar uma visão das áreas envolvidas, bem como sugerir um caminho para se abordar o problema, como consta no ítem proposta de continuidade.

### 8.1 Resumo do trabalho

Esta pesquisa foi iniciada através da identificação de duas das barreiras que a resolução de problemas numéricos em computadores enfrenta. Estas barreiras se referem à qualidade do resultado e ao porte do problema a ser resolvido, como foi visto no capítulo um. Observam-se tais questões apesar do avanço obtido na capacidade de processamento, desde os primeiros computadores até os supercomputadores atuais, onde se passou de um desempenho de algumas dezenas para bilhões de operações aritméticas em ponto-flutuante por segundo. Além do considerável aumento de capacidade de armazenamento dos dados.

Verificou-se a existência de uma grande lacuna entre o avanço tecnológico, com o desenvolvimento de computadores cada vez mais rápidos e poderosos, e a qualidade com que os cálculos são feitos. Com os supercomputadores (geralmente computadores vetoriais e/ou paralelos), os resultados são obtidos com extrema rapidez, mas não se sabe, em geral, quão confiáveis realmente são. Evoluiu-se no sentido de se resolver problemas cada vez maiores e mais complexos, enquanto que no sentido de qualidade (exatidão) do resultado foi dado o passo inicial, que foi a padronização da aritmética ordinária de ponto-flutuante.

Como a definição da aritmética da máquina ficava ao encargo do fabricante, cada sistema tinha as suas próprias características e defeitos. Cálculos efetuados em diferentes máquinas raramente produziam resultados compatíveis. Então, em 1980, a IEEE adotou o padrão de aritmética binária de ponto-flutuante, conhecida como padrão IEEE 754<sup>1</sup>. Várias de suas características são necessárias para se ter resultados garantidos. Entre elas, pode-se destacar a forma de representação dos números de ponto-flutuante; os intervalos de representatividade, os símbolos especiais de mais e menos infinito e o símbolo "not-a-number" (NaN) utilizados no tratamento de *underflow* e *overflow* e de outras exceções e, por fim, as operações com máxima exatidão, pois são definidas de forma a terem apenas um arredondamento, garantindo que resultados difiram do valor exato apenas na última casa da mantissa.

---

<sup>1</sup>ver [DIV94]

Isto foi um passo na direção certa no sentido de resolver a questão da qualidade numérica dos resultados, mas este padrão não especificou tudo. Nada foi mencionado para aritmética complexa, por exemplo. Uma falha grave resultante deste padrão, é que com sua adoção os fabricantes criaram a idéia dele ser suficiente. Erros produzidos são muitas vezes considerados como pequenas falhas.

Outras providências necessárias para melhorar a qualidade do resultado são: aritmética de alta exatidão, garantida pelos arredondamentos direcionados para cima e para baixo; pela definição das operações aritméticas básicas segundo as regras de semimorfismo; pelo uso da matemática intervalar e do cálculo do produto escalar com máxima exatidão e o uso de algoritmos com verificação automática do resultado. Mas infelizmente, nem todas estas inovações estão disponíveis para aqueles pesquisadores e cientistas que necessitam resolver problemas numéricos em suas aplicações. Isto se deve a alguns fatores, como por exemplo a disponibilidade da aritmética de alta exatidão nas máquinas digitais ou em linguagens de programação.

O trabalho de conscientização da comunidade matemática, da comunidade científica em geral e dos fabricantes de computadores sobre a necessidade da aritmética computacional seguir uma definição rigorosa, que garanta a qualidade do resultado e a verificação automática dos resultados, tem sido feito pelas sociedades científicas como **GAMM e IMACS**, através da publicação da **Proposta de Aritmética de ponto-flutuante vetorial exata** (veja [IMACS91]).

A pesquisa evoluiu para a proposta de desenvolvimento de uma aritmética de alta exatidão e alto desempenho, que torne disponíveis operações com intervalos e a própria matemática intervalar, aos usuários do supercomputador vetorial Cray Y-MP2E. A pesquisa sobre aritmética de alta exatidão e algoritmos com verificação automática do resultado foram relatadas no capítulo dois.

Através da proposta da biblioteca de rotinas de alta exatidão e alto desempenho, que tornasse disponível a matemática intervalar em ambiente vetorial, iniciaram-se estudos para sua concretização. A primeira meta era identificar os requisitos para tal tarefa. Como por exemplo, quais as características das linguagens de programação voltadas para a computação científica (conhecidas com extensões XSC) e ferramentas computacionais que foram desenvolvidas para proporcionar os avanços de qualidade, que dizem respeito à forma de implementação da aritmética intervalar de maneira que os resultados sejam produzidos com a máxima exatidão. A extensão do Pascal, PASCAL-XSC, desenvolvido na Alemanha (Karlsruhe), é um exemplo de linguagem de programação de propósitos gerais, mas voltada à implementação de algoritmos numéricos com resultados verificados matematicamente, ou seja autovalidáveis. Pascal XSC está disponível em várias plataformas e sistemas operacionais. Os resultados são compatíveis em qualquer plataforma. A questão de linguagens foi abordada no item 2.3, mas a descrição do Pascal XSC foi realizada no capítulo três. Os módulos avançados foram analisados no capítulo seis.

Através destes estudos se concluiu que o avanço prático do uso da matemática intervalar na resolução de problemas reais está, em parte, limitado pela inexistência de ferramentas computacionais que possibilitem um uso efetivo de intervalos. Observou-se que algumas das ferramentas existentes são limitadas, pois o porte dos problemas que se podem resolver é pequeno. Isto dificulta a disseminação prática e o uso de intervalos nas engenharias e em outras áreas.

Para a caracterização de aritmética de alto desempenho foi desenvolvido um estudo sobre suas necessidades. Como protótipo desta aritmética de alto desempenho, foi desenvolvido um estudo, uma especificação e, posteriormente, implementada uma biblioteca de rotinas intervalares no supercomputador Cray Y-MP2E, denominada *libavi.a*. Esta proposta foi apresentada no capítulo cinco, onde a *libavi.a* foi descrita em detalhes.

Com a *libavi.a* definiu-se a aritmética de alto desempenho, composta do processamento de alto desempenho (vetorial) e da matemática intervalar. Não se tem a aritmética de alta exatidão e alto desempenho, pois no ambiente vetorial, como do supercomputador Cray Y-MP2E com a linguagem de programação Fortran 90, a aritmética não segue o padrão da IEEE 754 na especificação do tamanho da palavra nem na forma como os arredondamentos e operações aritméticas em ponto-flutuante são efetuadas, assim sendo, foi necessário desenvolver rotinas que simulassem tais operações.

Verificaram-se, também, as limitações do processamento vetorial na realização do produto escalar e de somas acumuladas. As diferenças nos resultados foram identificadas como resultantes da forma como os cálculos são efetuados. As diferenças de resultados foram encontradas não só em relação a máquinas diferentes, mas também entre o processamento seqüencial e vetorial. Para solucionar esta limitação seriam necessários acumuladores longos, de precisão maior do que a disponível no Cray ou então a simulação por software da aritmética inteira de precisão infinita. Infelizmente isto ultrapassou os objetivos desta pesquisa, apesar de estabelecer certas limitações à biblioteca de rotinas intervalares quanto à aritmética de alta exatidão e ao desenvolvimento de algoritmos que verificam automaticamente o resultado.

Em relação ao tempo de execução, estes métodos podem ser mais demorados do que os métodos reais disponíveis nas bibliotecas matemáticas e científicas da Cray, entretanto sabe-se que os resultados produzidos por ela são confiáveis, pois são verificados automaticamente; além do fato de que são utilizadas as operações reais e as rotinas disponíveis no Cray para se implementar as operações envolvendo intervalos.

As operações sobre dois intervalos no computador resultam de operações sobre dois extremos apropriadamente escolhidos dos operandos intervalares, onde o cálculo do extremo inferior é arredondado para baixo e o cálculo do extremo superior para cima. Dessa forma, o resultado certamente contém todos os resultados da operação aplicada individualmente aos elementos do primeiro e do segundo operandos.



Para obtenção de uma aritmética de alto desempenho, a aritmética deve ser incrementada com outros elementos. Todas as operações com números de ponto-flutuante devem ser supridas de arredondamentos direcionados, isto é, arredondamentos para baixo (para o número de máquina anterior), para cima (para o número de máquina posterior) e simétrico (para o número de máquina mais próximo). Uma aritmética intervalar para números reais e complexos em ponto-flutuante pode ser construída com estas operações.

A biblioteca *libavi.a* é um conjunto de rotinas intervalares que reúne as características da matemática intervalar no ambiente do supercomputador vetorial Cray Y-MP. A *libavi.a* foi desenvolvida em Fortran 90, o que possibilitou as características de modularidade, sobrecarga de operadores e funções, uso de *arrays* dinâmicos na definição de vetores e matrizes e a definição de novos tipos de dados próprios a análise matemática.

Como pode ser observado na figura 5.5, a biblioteca foi organizada em quatro módulos, estes são: *básico*, *mvi*, *aplic* e *ci*. O módulo *básico* contém a aritmética intervalar básica, sendo por isso utilizado por todos os demais módulos. O módulo *aplic* contém os demais módulos, pois se utiliza deles. O módulo de intervalos complexos, contém o módulo *básico*. Os tipos, vetores e matrizes de intervalos complexos foram definidos, mas não foram desenvolvidas as rotinas que os manipulam. Elas são a extensão do módulo *mvi* para complexos, o que na figura 5.5 representaria um retângulo incluindo *mvi* e *ci*. A seguir é descrito o conteúdo de cada um dos módulos:

- módulo *básico* - intervalos reais - 53 rotinas. Estas rotinas foram organizadas dentro deste módulo em seis conjuntos, de acordo com o tipo de operação que realiza. Esses conjuntos são: funções de transferência, operações relacionais, operações entre conjuntos, operações aritméticas, funções básicas e rotinas de entrada e saída.

- módulo *mvi* - matrizes e vetores de intervalos - 151 rotinas. Neste módulo são implementadas as operações entre vetores de intervalos, matrizes de intervalos e as rotinas que manipulam diferentes tipos de dados com intervalos, vetores e matrizes de intervalos. Existem, portanto, três seções neste módulo. As duas primeiras, que implementam as operações com vetores e com matrizes de intervalos, são compostas, cada uma, por seis conjuntos de rotinas. Esses conjuntos são: funções de transferência, operações relacionais e entre conjuntos, operações aritméticas, funções básicas elementares, funções pré-definidas e rotinas de entrada e saída. A terceira seção, onde estão operações entre diferentes tipos de dados, comporta sete conjuntos de operações, que são: operações de reais com intervalos, operações de reais com vetores de intervalos, operações de reais com matrizes de intervalos, operações de intervalos com vetores de intervalos, operações de intervalos com matrizes de intervalos, operações entre vetores e matrizes de intervalos e operações entre vetor/matriz real com vetor/matriz de intervalos.

- módulo *ci* - intervalos complexos - 58 rotinas. Este módulo implementa intervalos complexos. Da mesma forma que o módulo básico, ele foi organizado em seis conjuntos de rotinas que são: funções de transferência, operações relacionais, operações

entre conjuntos, operações aritméticas, funções elementares entre intervalos complexos e rotinas de entrada e saída.

- módulo *aplic* - aplicações da álgebra linear - 29 rotinas. Este módulo estende algumas das rotinas reais da biblioteca BLAS para intervalos. Este conjunto de rotinas inclui operações entre vetores, entre matriz e vetores e entre matrizes da álgebra linear. Estas rotinas incluem os três níveis da BLAS. Estão incluídas algumas rotinas que determinam normas de matrizes, vetores e condicionamento.

Além da aritmética vetorial intervalar (operações, funções e avaliação de expressões), sentiu-se a necessidade de providenciar bibliotecas que tornassem disponíveis a estes usuários os métodos intervalares para solução de seus problemas. Inicialmente, foi especificada a biblioteca científica aplicada *libselint.a*, composta por algumas rotinas intervalares de resolução de equações algébricas e sistemas de equações lineares. O que na figura 5.5 representa o retângulo mais externo. Esta biblioteca intervalar aplicada, *libselint.a* deve ser composta por quatro módulos, são estes:

- Módulo *dirint*, deve incluir os métodos baseados em operações algébricas intervalares e propriedades intervalares; também são conhecidos como métodos diretos intervalares.

- Módulo *refint*, deve incluir os métodos baseados em inclusões ou refinamentos intervalares da solução e do erro; também podem ser chamados de métodos híbridos, uma vez que se pode utilizar a solução inicial calculada por métodos pontuais ou a matriz inversa pontual.

- Módulo *itrint*, deve incluir os métodos iterativos intervalares, conhecidos também como métodos de relaxação. Eles também se baseiam em inclusões monotônicas.

- Módulo *equalg*, deve conter rotinas do caso particular de sistema linear de ordem um, ou seja, resolução algébrica de equações por métodos intervalares, como as versões intervalares de Newton.

Observa-se que desta biblioteca aplicada só foram implementadas algumas rotinas visando verificar e validar o uso da biblioteca intervalar e da matemática intervalar em supercomputadores. Este capítulo está em fase de pesquisa, busca-se identificar metodologias de desenvolvimento e métodos intervalares de resolução de equações e sistemas lineares.

Por fim, foram desenvolvidos vários testes que verificaram a biblioteca de rotinas intervalares quanto à sua correção e compatibilidade com a documentação. Em especial, as rotinas que implementam as operações e funções aritméticas intervalares e operações entre conjuntos foram verificadas por exaustão, como pode ser encontrado no capítulo sete. Todos os resultados obtidos através de programas que utilizavam a *libavi.a* foram comparados com os resultados produzidos por programas análogos em Pascal XSC.



Para a determinação das rotinas que fazem parte da biblioteca *libavi.a*, foram analisadas as bibliotecas matemática e científica (*libm.a* e *libsci.a*) disponíveis no Cray, a biblioteca *Numerals* que havia sido estudada e documentada pelo Grupo de Matemática Computacional da UFRGS e os módulos avançados do Pascal XSC, como pode ser visto no capítulo seis.

A verificação do uso da Matemática Intervalar no supercomputador vetorial se deu através da resolução de problemas numéricos implementados em Fortran 90, utilizando a *libavi.a*, e seus resultados foram confrontados com o de outras bibliotecas. Maiores detalhes serão apresentados a seguir.

## 8.2 Comentário sobre resultados obtidos

Neste item são feitos alguns comentários sobre os resultados obtidos com esta pesquisa. O primeiro resultado prático foi o desenvolvimento da biblioteca de rotinas intervalares *libavi.a* com sua documentação. Os comentários que podem ser feitos se referem às características desta biblioteca.

Como o propósito do desenvolvimento desta biblioteca foi o de providenciar uma ferramenta útil para programação científica (em ambiente vetorial), as rotinas implementadas necessitavam de certas características como: exatidão, eficiência, confiabilidade, validação, facilidade de uso, boa documentação, modificabilidade, completude e transportabilidade.

A exatidão destas rotinas foi obtida pelo uso das rotinas disponíveis no Fortran 90 para manipulação de números reais (números em ponto-flutuantes), especialmente as funções básicas elementares. Utilizou-se para obter melhor exatidão algumas rotinas em dupla precisão, especialmente para tentar se obter um produto escalar mais próximo do ótimo. Utilizaram-se, ainda, algumas das rotinas da biblioteca BLAS para a implementação das operações entre vetores e matrizes de intervalos. A extensão para rotinas intervalares se deu através da aplicação de princípios de conversão real-intervalo e das propriedades de semimorfismo. Manteve-se a exatidão disponível do supercomputador Cray para processamento científico, mas se acrescentou, com intervalos, a garantia dos resultados.

Em relação à eficiência, evoluiu-se mais neste conceito, pois inicialmente eficiência significava um uso otimizado da memória, por ser esta muito cara e escassa. Com a evolução tecnológica, eficiência se vinculou à idéia de velocidade de processamento. Perdeu-se um pouco da idéia de qualidade para se ter velocidade. Com a matemática intervalar (acrescida de aritmética de alta exatidão) é resgatado o conceito de eficiência como qualidade e garantia do resultado calculado com rapidez. A biblioteca *libavi.a* produz resultados um pouco mais lentos do que a aritmética de ponto-flutuante ordinária, uma vez que os cálculos são efetuados para os extremos dos intervalos, ganha-se, contudo, na garantia dos resultados.

As qualidades de confiabilidade e validação, dizem respeito à biblioteca ser consistente com a documentação, ou seja, a execução do programa realiza exatamente o que seu propósito descrito na documentação define, resolvendo, assim, a classe de problemas a que se propõe.

Sobrecarga de funções e operadores é uma das facilidades da biblioteca de rotinas intervalares. A sobrecarga possibilita que o usuário utilize o nome genérico de uma função, independente do tipo de argumentos desta função. A tarefa de identificar a rotina correta que opera com tais argumentos é transferida ao compilador. Isto facilita a programação. Foram sobrecarregados os símbolos das operações aritméticas de soma, subtração, multiplicação, divisão e os símbolos relacionais existentes no Fortran 90. As demais funções matemáticas foram sobrecarregadas ao nome usual da notação matemática.

O uso de *arrays* dinâmicos na definição de vetores e matrizes de intervalos contribui para a simplificação e facilidade de uso da biblioteca. *Arrays* dinâmicos permitem não só um uso otimizado da memória, através do uso de vetores e matrizes de tamanho aleatório, garantindo alocação e liberação do espaço da memória, mas também facilitam a programação, uma vez que o usuário apenas utiliza a sua definição segundo a compatibilidade do sistema.

A otimização e vetorização das rotinas da *libavi.a* se deram como que por herança das características da máquina, do compilador e das rotinas que são utilizadas pelas bibliotecas, como as rotinas da BLAS. O uso destas rotinas, além de proporcionar esta vantagem, também produziu a mesma qualidade em termos de exatidão dos resultados, adicionado ao uso de intervalos, o que proporciona resultados contidos dentro do intervalo solução.

A modularidade é outra característica da biblioteca de rotinas intervalares. Em cada módulo são definidas as operações que manipulam certo tipo de elementos ou dados. Isto facilita o entendimento, a manutenção e produz maior estruturação na biblioteca.

Outra característica é quanto à documentação. O manual de utilização seguiu os padrões da documentação das bibliotecas da Cray Research Inc. Nele são apresentadas: a descrição dos módulos, uma introdução a cada conjunto de rotinas e uma descrição detalhada para cada uma das 290 rotinas implementadas.

Outro resultado, que pode ser considerado como parcial, foi a especificação da biblioteca intervalar científica aplicada *libselint.a*, contendo métodos numéricos intervalares, voltados à resolução de problemas de equações e sistemas lineares. Também foram iniciados estudos para a identificação de metodologias de desenvolvimento de métodos intervalares. A continuação desta pesquisa já está em andamento.

Verificou-se ainda que, em Fortran 90, os valores são arredondados para o número mais próximo de máquina, ou seja, o arredondamento simétrico, onde o dígito 15 da mantissa é analisado. Caso seja um dos dígitos {0, 1, 2, 3, 4} então o número é truncado, mantendo-se os 14 primeiros dígitos significativos. Caso o dígito 15 da mantissa seja um dos dígitos {5, 6, 7, 8, 9}, então será adicionada uma unidade ao dígito 14 e truncado o número da mesma forma. Esta soma de uma unidade é feita em módulo. Isto poderia resultar na perda do valor exato no intervalo, portanto utilizaram-se as rotinas *infla-down* e *infla-up* de forma a simular os arredondamentos direcionados.

Na primeira versão destas rotinas tentou-se utilizar a rotina em fortran *nearst* que calcula o número de máquina mais próximo. Mas o resultado destas rotinas foi desastroso, pois o intervalo inflacionado perdia o valor exato em vários casos. Então projetaram-se novas rotinas que simulavam os arredondamentos direcionados, baseando-se na idéia de somar ou subtrair uma unidade no décimo-terceiro dígito menos significativo da mantissa. Este valor depende do sinal do número a ser inflacionado e do expoente. Pois, na verdade, soma-se ou subtrai-se  $1 \cdot 10^{-13+u}$ , onde  $u$  é o expoente do valor a ser inflado. Desta forma, o resultado aproximou-se da definição dos arredondamentos monotônicos e obteve-se a garantia de que os resultados estariam sempre contidos dentro do intervalo.

Resultados com 13 a 14 dígitos significativos exatos podem ser obtidos com a biblioteca de rotinas intervalares, com a garantia de que o valor exato está contido no intervalo.

Na questão da avaliação do desempenho, do ganho computacional com a utilização do supercomputador Cray Y-MP2E, há dois aspectos a serem considerados. O primeiro está relacionado à velocidade de processamento de 330 Megaflops por segundo por CPU, o que proporciona que os cálculos sejam efetuados muito rapidamente.

Por outro lado, deve-se considerar o uso da vetorização. Utilizou-se apenas a otimização e vetorização que é realizada automaticamente pelo compilador F90, o que também proporcionou cálculos executados rapidamente. Este segundo aspecto, entretanto, tem alguns efeitos colaterais, como foi visto no produto escalar, onde a troca da ordem de parcelas em somatórios produz resultados diferentes. Com o uso de intervalos, este problema é controlado pois o resultado intervalar contém o valor exato. Pode acontecer que o diâmetro do intervalo seja muito grande, devido à falta da aritmética de alta exatidão, pois em Pascal XSC se observou que o resultado exato pode ser obtido com o uso de expressões exatas e variáveis do tipo *doubleprecision* como visto no item 7.1.5.

Resultados obtidos pela *libavi.a* são tão eficientes quanto os resultados obtidos pela biblioteca de rotinas voltadas para álgebra linear BLAS disponível na *libsci.a*, pois além de a utilizar, a *libavi.a* opera com intervalos que contém a solução, proporcionando resultados com limites confiáveis.

### 8.3 Limitações do trabalho

Algumas das limitações da biblioteca de rotinas intervalares são decorrentes da arquitetura do Supercomputador Cray e de sua aritmética de ponto-flutuante, como por exemplo, o fato de não se ter na *libavia* operações aritméticas com máxima exatidão, condição necessária para se ter aritmética de alta exatidão e desempenho.

A representação dos números reais no Cray pode ser feita em precisão simples (em uma palavra de 64 *bits*) ou em dupla precisão (duas palavras). Na precisão simples os *bits* são divididos em três regiões: sinal (*bit* 63), expoente (*bits* 62-48) e mantissa (*bits* 47-0). Os quarenta e oito *bits* da mantissa proporcionam, aproximadamente, 15 dígitos decimais de precisão. Em dupla precisão, apenas se aumenta o número de dígitos da mantissa, que passa dos 48 para 96, proporcionando uma mantissa decimal de 29 dígitos. Apesar da representação dos números em ponto-flutuante no Cray contar com mais dígitos do que a especificação do padrão IEEE 754, a forma como as unidades funcionais de ponto-flutuante realizam as operações aritméticas resulta em perda da exatidão nos cálculos.

Através das rotinas que simulam os arredondamentos direcionados, garantiram-se treze dígitos decimais nos intervalos de reais. Estas rotinas simulam o comportamento dos arredondamentos, mas não são arredondamentos. Elas não testam se um determinado valor já é representado e aplicam o arredondamento no décimo-terceiro dígito da mantissa. Isto se constitui em outra limitação, pois sempre se tem um intervalo maior do que se poderia ter, não sendo de máxima exatidão. No Cray enfrenta-se esta limitação.

A falta das propriedades das operações aritméticas definidas sobre o conjunto dos números em ponto-flutuante, que garantem a estrutura algébrica de semimorfismo, introduz erros e, algumas vezes, é a causa de resultados inesperados em cálculos efetuados no supercomputador; especialmente no modo vetorial, onde a ordem que as operações são efetuadas são alteradas.



## 8.4 Melhorias que podem ser feitas

Entre as possíveis melhorias em relação à biblioteca de rotinas intervalares, destacam-se as que apontam para uma exatidão mais próxima da ótima. Isto implica na implementação das rotinas de arredondamento direcionado, do produto escalar ótimo e do controle de erros nas operações aritméticas.

Outra melhoria diz respeito à visualização dos resultados no que tange às rotinas de entrada e saída de dados. Não se utilizaram as potencialidades do Fortran 90 para se obter melhor exatidão nos resultados impressos, pois as rotinas de impressão mostram valores em formato livre. Pretende-se desenvolver rotinas que possibilitem ao usuário definir o formato dos dados.

Em relação à documentação, no manual de utilização das rotinas os exemplos dados na descrição detalhada foram os mais simples. Na nova versão devem ser incluídos exemplos mais elaborados, incluindo situações problemáticas.

## 8.5 Propostas para continuidade

Entre as propostas de continuidade desta pesquisa estão: a implementação das melhorias que podem ser feitas à biblioteca de rotinas intervalares; o desenvolvimento do módulo de matrizes e vetores de intervalos complexos; a implementação da biblioteca intervalar científica *libselint.a* para sistemas lineares; a implementação da biblioteca de rotinas intervalares para integração numérica e a disseminação da matemática intervalar e da biblioteca entre pesquisadores da UFRGS e CESUP, que se utilizam do Cray para resolver problemas.

Em relação às **melhorias para a *libavi.a*** estão o desenvolvimento de uma aritmética de alto desempenho e alta exatidão no ambiente do supercomputador vetorial Cray Y-MP, o que implica o estudo e a implementação dos arredondamentos direcionados, do controle automático do erro nas operações aritméticas básicas, do produto escalar exato e, ainda, a incorporação destas características à biblioteca *libavi.a*; e a implementação do módulo de rotinas sobre vetores e matrizes de intervalos complexos.

Para o **projeto de bibliotecas intervalares aplicativas** devem ser estudados os problemas de resolução de equações algébricas, solução de sistemas lineares e integração numérica. O estudo compreende métodos pontuais básicos e métodos intervalares. Busca-se não só a resolução de problemas, mas também identificar as metodologias de desenvolvimento de métodos intervalares de forma a se obter resultados com alta exatidão, entre limites confiáveis (um intervalo) e que sejam verificados automaticamente pelos computadores.



Como resultado da análise dos métodos intervalares existentes na literatura para solução de equações algébricas e sistemas lineares deve-se **especificar, implementar e documentar a biblioteca aplicativa intervalar *libselint.a***.

Do estudo de métodos intervalares existentes na literatura sobre integração numérica, visando a descrição, caracterização, identificação dos princípios dos métodos intervalares, deve-se especificar, implementar e documentar a **biblioteca aplicativa intervalar *libintnum.a***, abrangendo este capítulo.

**Novos métodos intervalares** poderão ser desenvolvidos e implementados e seus resultados comparados, uma vez que se têm a aritmética intervalar e rotinas básicas disponíveis no supercomputador.

Outra proposta de continuidade é o estudo do desempenho da matemática intervalar na resolução de problemas numéricos, através de ferramentas computacionais, em ambientes: seqüencial (Pascal-XSC em PC486dx ou estações de trabalho Sun), vetorial (Biblioteca de rotinas intervalares *libavi.a* no Cray Y-MP2E), paralelo (Biblioteca paralela a ser desenvolvida em plataforma de Transputers) e distribuídos (rede de estações de trabalho). Quer-se estudar a resolução de problemas numéricos em diferentes ambientes computacionais, identificando as características específicas de cada ambiente, as vantagens e desvantagens dos diferentes tipos de processamento sobre a resolução de problemas com aritmética de alta exatidão.

Entre estas questões, destaca-se a análise do paralelismo existente em cada uma das operações e funções intervalares, para se determinar se a plataforma de hardware adotada é suficiente, ou seja, se existem processadores suficientes para que o paralelismo inerente da operação seja utilizado. Ao mesmo tempo, deve-se considerar, em cada operação, como utilizar cada processador, de forma que haja um equilíbrio de tarefas para cada processador.

A solução de problemas práticos que envolvam sistemas de equações lineares por métodos numéricos e intervalares como exemplos, para que sejam feitas comparações quanto a tempo de processamento, exatidão, convergência e controle de instabilidades; e assim, seja disseminado entre usuários do supercomputador o uso da matemática intervalar.

Algumas destas questões e propostas já se encontram em andamento na forma de trabalho de pesquisa de dissertação ou de projeto de pesquisa do Grupo de Matemática Computacional em Conjunto com o Grupo de Arquiteturas de Computadores e Processamento de Alto Desempenho do PGCC da UFRGS, com apoio do CNPq e do Protem.

## 9 BIBLIOGRAFIA

- [ADA93] ADAMS, E.; KULISCH, U. (Eds.). **Scientific Computing with automatic result verification.** San Diego: Academic Press, 1993. 612p.
- [AHO74] AHO, A. V.; HOPCROFT, J. E.; ULLMAN, J. D. **The design and analysis of computer algorithms.** Reading: Addison Wesley, 1974. 470p.
- [AKL89] AKL, S. G. **The design and analysis of parallel algorithms.** Englewood Cliffs: Prentice Hall, 1989. 401p.
- [ALB73] ALBRECHT, P. **Análise numérica: um curso moderno.** Rio de Janeiro: LTC, 1973. 240p.
- [ALE80] ALEFELD, G.; GRIGORIEFF, R. D. (Eds.). **Fundamentals of Numerical Computation (Computer-Oriented Numerical Analysis).** Wien: Springer Verlag, 1980. (COMPUTING SUPPLEMENTUM 2, Seminar held in Berlin, June 5-8, 1979).
- [ALE83] ALEFELD, G.; HERZBERGER, J. **Introduction to interval computation.** New York: Academic Press, 1983. 333p.
- [AYR71] AYRES Jr., F. **Matrizes.** São Paulo: McGraw-Hill, 1971. 286p.
- [BAR87] BARROSO, L.C. et al. **Cálculo numérico (com aplicações).** São Paulo: Habra, 1987. 367p.
- [BER89] BERTSEKAS, D. P.; TSITSIKLIS, J. V. **Parallel and distributed computations numerical methods.** Englewood Cliff: Prentice Hall, 1989. 715p.

- [BOH91a] BOHLENDER, G. Vector extension of the IEEE standard for floating point arithmetic. In: KAUCHER, E.; MARKOV, S. M.; MAYER, G. (Eds.). **Computer Arithmetic, Scientific Computation and Mathematical Modelling**. Basel: J.C.Baltzer, 1991. p.3-12. (Proceedings of SCAN'90, IMACS Annals on Computing and Applied Mathematics, Albena, Sept 24-28, 1990, 12).
- [BOH91c] BOHLENDER, G.; KROFEL, A. A Survey of pipeline hardware support for accurate scalar products. In: KAUCHER, E.; MARKOV, S. M.; MAYER, G. (Eds.). **Computer Arithmetic, Scientific Computation and Mathematical Modelling**. Basel: J.C.Baltzer, 1991. p.22-43. (Proceedings of SCAN'90, IMACS Annals on Computing and Applied Mathematics, Albena, Sept 24-28, 1990, 12).
- [BON91] BONDELI, S. Divide and conquer: a parallel algorithm for the solution of a tridiagonal linear system of equations. **Parallel Computing**. Amsterdam, v.17, n.4&5, p.419-434, July 1991.
- [CAP85] CAPPELLO, P. R. A mesh automation for solve dense linear systems. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, St Charles, Aug. 20-23, 1985. **Proceedings...** New York: IEEE, 1985. p.418-425.
- [CAR69] CARNAHAN, B.; LUTHER, H. A.; WILKES, J. O. **Applied Numerical Methods**. New York: John Wiley & Sons, 1969. 604p.
- [CLA79] CLAUDIO, D. M. **Beiträge zur struktur der recherearithmetik**. Karlsruhe: Universität Karlsruhe, 1979. 85p. (Dr dissertation).
- [CLA83] CLAUDIO, D. M.; ROYO DOS SANTOS, J. A. **Microcomputadores e minicalculadoras seu uso em ciências e engenharia**. São Paulo: E. Blucher, 1983. 423p.

- [CLA89] CLAUDIO, D. M.; MARINS, J. M. **Cálculo numérico computacional**. São Paulo: Atlas, 1989. 464p.
- [CRA90a] CRAY RESEARCH, Inc. **CF77 Compiling System. Fortran reference manual.**, n.sr.3071, versão 5.0. [S.l : s.n.], 1990. v.1.
- [CRA90b] CRAY RESEARCH, Inc. **Software Training. CF77 & SCC features and optimization.** [S.l : s.n.], 1990. (TR-OPT revisão D).
- [CRA91] CRAY RESEARCH, Inc. **UNICOS Math and Scientific Library Reference Manual**, n.sr-2081, versão 6.0. [S.l : s.n.], 1991. v.3.
- [CRA93] CRAY RESEARCH, Inc. **CF90 Fortran language reference manual.**, n.sr.3902, versão 1.0. [S.l : s.n.], 1993.
- [DEI86] DEIF, A. **Sensitivity analysis in linear systems**. Berlin: Springer-Verlag, 1986. 224p.
- [DEM85] DEMMEL, J. W.; KRUCKEBERG, F. An Interval Algorithm for Solving Systems of Linear Equations to Prespecified Accuracy. **Computing**, New York, v.34, p.117-129, 1985.
- [DEW83] DEW, P. M.; JAMES, K. R. **Introduction to numerical computation in Pascal**. Hong Kong: MacMillan Press, 1983. 291p.
- [DIV86] DIVERIO, T. A. **Software Numérico Aplicativo e Instrucional**. Porto Alegre: CPGCC da UFRGS, 1986. (Dissertação de mestrado).
- [DIV87] DIVERIO, T. A.; CLAUDIO, D. M.; TOSCANI, L. V. **Fundamentos de Matemática Computacional**. Porto Alegre: DC Luzzato, 1987. 194p.

- [DIV90] DIVERIO, T. A. et al. **Levantamento bibliográfico de métodos numéricos.** Porto Alegre: GMC-PGCC da UFRGS, 1990. 24p.
- [DIV90a] DIVERIO, T. A. et al. **LEPMAC** - Caderno de Exercícios do Sistema de Software do Laboratório. Porto Alegre: GMC-PGCC da UFRGS, 1990. 103p.
- [DIV90b] DIVERIO, T. A. **LEPMAC** - Material Didático de Apoio - módulo de REATS. Porto Alegre: GMC-PGCC da UFRGS, 1990. 62p.
- [DIV90c] DIVERIO, T. A. **LEPMAC** - Material Didático de Apoio - módulo de SELAS. Porto Alegre: GMC-PGCC da UFRGS, 1990. 61p.
- [DIV91] DIVERIO, T. A. **Processamento vetorial e vetorização de algoritmos na máquina Convex C210.** Porto Alegre: CPGCC da UFRGS, 1991. 100p. (RP,160).
- [DIV91a] DIVERIO, T. A. et al. **Versões intervalares do método de Newton.** Porto Alegre: CPGCC da UFRGS, 1991. 86p. (RP, 161).
- [DIV92] DIVERIO, T. A. et al. **Introdução a teoria dos intervalos.** Porto Alegre: CPGCC da UFRGS,1992. 39p. (RP, 172).
- [DIV93] DIVERIO, T. A. et al. **Supercomputador Cray:** arquitetura, características e acesso remoto. Porto Alegre: CPGCC da UFRGS, 1993. 86p. (RP, 210).
- [DIV93a] DIVERIO, T. A.; HOLBIG, C. A. **Aritmética Intervalar - uma nova abordagem.** Porto Alegre: CPGCC da UFRGS, 1993. (Tradução do Kulisch).



- [DIV94] DIVERIO, T. A.; HOLBIG, C. A.; NORONHA, C. R. **Sistema de Ponto-Flutuante e o Padrão IEEE 754.** Porto Alegre: CPGCC da UFRGS, 1994. 58p. (RP, 225).
- [DIV94a] DIVERIO, T. A.; HOLBIG, C. A.; FERNANDES, U. A. L. **Sistemas de Equações Lineares: Instabilidade, Análise sensível e métodos intervalares.** Porto Alegre: CPGCC da UFRGS, 1994. 56p. (RP, 228).
- [DIV94b] DIVERIO, T. A.; FERNANDES, U. A. L. **Aplicações de Intervalos.** Porto Alegre: CPGCC da UFRGS, 1994. 64p. (RP, 235).
- [DIV94c] DIVERIO, T. A.; PORCIUNCULA, C. B.; NAVAU, P. O. A. **Uso de Transputers para definição de uma arquitetura intervalar paralela.** Porto Alegre: CPGCC da UFRGS, 1994. 66p. (RP, 236).
- [DIV94d] DIVERIO, T. A. **PASCAL XSC - Pascal para computação científica: descrição, instalação e aplicações.** Porto Alegre: CPGCC da UFRGS, 1994. 65p. (RP, 245).
- [DIV95] DIVERIO, T. A. **Uso efetivo da matemática intervalar em supercomputadores vetoriais.** Porto Alegre: CPGCC da UFRGS, 1995. 290p. (Tese de doutorado).
- [DIV95a] DIVERIO, T. A.; FERNANDES, U. A. L.; DAHMER, A. **Limitações do Processamento vetorial no Cray Y-MP2E.** Porto Alegre: CPGCC da UFRGS, 1995. 60p. (RP, 248).
- [DIV95b] DIVERIO, T. A. **LIBAVIA Manual de utilização.** Porto Alegre: CPGCC da UFRGS, 1995. 430p.
- [DIV95c] DIVERIO, T. A. **LIBAVIA Verificação e Validação.** Porto Alegre: CPGCC da UFRGS, 1995. 217p.

- [DOR81] DORN, W. S.; McCracken, D. D. **Cálculo numérico - com estudos de casos em Fortran IV.** Rio de Janeiro: Campus, 1981, 568p.
- [DUF86] DUFF, I. S.; ERISMAN, A. M.; REID, J. K. **Direct methods for sparse matrices.** Oxford: Oxford University Press, 1986. 341p.
- [DUF87] DUFF, I. S. The influence of vector and parallel processors on numerical analysis. In: **The state of the art in numerical analysis.** Oxford: Clarendon Press, 1987.
- [FOR67] FORSYTHE, G.; MOLLER, C. B. **Computer solution of linear algebraic systems.** Englewood Cliffs: Prentice Hall, 1967. 148p.
- [FOR77] FORSYTHE, G. E.; MALCOLM, M. A.; MOLER, C. B. **Computer Methods for mathematical computations.** Englewood Cliffs: Prentice Hall, 1977. 259p.
- [GEO93] GEÖRG, S.; HAMMER, R.; NEAGA, M. et al. **PASCAL-XSC: A PASCAL Extension for Scientific Computation and Numerical Data Processing.** Karlsruhe: IAM-University of Karlsruhe, 1993. 38p.
- [HAH85] HAHN, W.; MOHR, K.; SCHAUER, U. Some Techniques for Solving Linear Equations Systems with Guarantee. **Computing.** New York, v.34, p.375-379, 1985.
- [HAM90] HAMMER, R. How reliable is the arithmetic of vector computers? In: ULLRICH, C. (Ed.). **Contributions to computer arithmetic and self-validating Numerical Methods.** Basel: J.C Baltzer, 1990. p.467-482. (Proceedings of SCAN'89, Basel, Oct 2-6, 1989, submitted papers, 7).

- [HAM93] HAMMER, R.; HOCKS, M.; KULISCH, U. et al. **Numerical Toolbox for Verified Computing I: basic numerical problems.** Berlin: Springer-Verlag, 1993. 337p.
- [HAN65] HANSEN, E. Interval arithmetic in matrix computation Part 1. **SIAM J Numer Anal.** Philadelphia, v.2, p.308-320, 1965.
- [HAN67] HANSEN, E.; SMITH, R. Interval arithmetic in matrix computation Part 2. **SIAM J Numer Anal.**, Philadelphia, v.4, p.1-9, 1967.
- [HAN69] HANSEN, E. On the Solution of Linear Algebraic Equations with Interval Coefficients. **Linear Algebra Appl.**, New York, v.2, p.153-165, 1969.
- [HAN81] HANSEN, E.; SENGUPTA, S. Bounding Solutions of Systems of Equations Using Interval Analysis. **Bit**, Copenhagen, v.21, p.203-221, 1981.
- [HEL78] HELLER, D. A survey of parallel algorithms in numerical linear algebra. **SIAM Review**, New York, v.20, n.4, p.740-777, Oct, 1978.
- [HOL95] HÖLBIG, C. A. **Sistemas lineares: aplicabilidade, resolução e verificação automática.** Porto Alegre: CPGCC da UFRGS, 1995. 70p. (TI, 472).
- [IMACS91] IMACS, GAMM. Resolution on computer arithmetic. In: KAUCHER, E.; MARKOV, S. M.; MAYER, G. (Eds.), **Computer Arithmetic, Scientific Computation and Mathematical Modelling.** Basel: J.C.Baltzer, 1991. p.477-479. (Proceedings of SCAN'90, IMACS Annals on Computing and Applied Mathematics, Albena, Sept 24-28, 1990, 12).
- [KAU87] KAUCHER, R.; KULISCH, U.; ULLRICH, C. (Eds.). **Computer arithmetic: Scientific computing and programming languages.** Stuttgart: B.G Teubner Verlag, 1987.

- [KAU91] KAUCHER, E.; MARKOV, S. M.; MAYER, G. (Eds.). **Computer Arithmetic, Scientific Computation and Mathematical Modelling.** Basel: J.C.Baltzer, 1991. v.12. (Proceedings of SCAN'90, IMACS Annals on Computing and Applied Mathematics, Albena, Sept 24-28, 1990).
- [KEA90] KEARFOTT, R. B. Preconditioners for the Interval Gauss-Seidel Method. **Siam J Numer Anal**, Philadelphia, v.27, p.804-822, 1990.
- KIR88a] KIRCHNER, R.; KULISCH, U. W. Accurate arithmetic for vector processors. **Journal of Parallel and Distributed Computing.** Philadelphia, v.5, n.S, p.250-270, 1988.
- [KIR88b] KIRCHNER, R.; KULISCH, U. Arithmetic for vector processors. In: MOORE, R. E. (Ed.). **Reliability in Computing: The role of interval methods in scientific computing.** San Diego: Academic Press, 1988. p.3-41. (Proceedings of the conference at Colombus, Ohio, Sept 8-11, 1987, 19).
- [KLA91] KLATTE, R.; KULISCH, U.; NEAGA, M. et al. **PASCAL-XSC:** Language Reference with Examples. Berlin: Springer-Verlag, 1991. 38p. (folheto).
- [KLA92] KLATTE, R.; KULISCH, U.; NEAGA, M. et al. **PASCAL-XSC** language reference with examples. Berlin: Springer Verlag, 1992. 344p.
- [KLA93] KLATTE, R.; JULISCH, U.; WIETHOFF, A. et al. **C-XSC:** A C++ class library for extended scientific computing. Berlin: Springer Verlag, 1993.
- [KUL81] KULISCH, U. W.; MIRANKER, W. L. **Computer arithmetic in theory and practice.** New York: Academic Press, 1981. 249p.

- [KUL83] KULISCH, U. W.; MIRANKER, W. L. **A new approach to scientific computation.** New York: Academic Press, 1983. 384p.
- [KUL86] KULISCH, U. W.; MIRANKER, W. L. The arithmetic of the digital computer: a new approach. **SIAM REVIEW**, Philadelphia, v.28, n.1, p.1-40, 1986.
- [KUL87] KULISCH, U.; MIRANKER, W. R. **Computer arithmetic theory and practice.** New York: Academic Press, 1987.
- [KUL88] KULISCH, U.; STETTER, H. J. (Eds.). Scientific computation with automatic result verification. Wien: Springer Verlag, 1988. (COMPUTING SUPPLEMENTUM 6, Seminar held in Karlsruhe, Sept 30- Oct 2, 1987).
- [LEY93] LEYSER, M. **Propriedades algébricas reais e intervalares: estudo comparativo.** Porto Alegre: CPGCC da UFRGS, 1993. 40p. (TI, 310).
- [LOP87] LOPEZ, J. G. **Introdução a matemática intervalar.** Porto Alegre: UFRGS, 1987. 86p. (TI, s.n.)
- [MAY88] MAYER, G. Enclosing the Solutions of Systems of Linear Equations for Interval Iterative Process. In: KULISCH, U.; STETTER, H. J. (Eds.). **Scientific computation with automatic result verification.** Wien: Springer Verlag, 1988. p.804-822. (COMPUTING SUPPLEMENTUM 6, Seminar held in Karlsruhe, Sept 30- Oct 2, 1987).
- [MAY91] MAYER, G. Old e News Aspects for the Interval Gaussiam Algorithm. In: KAUCHER, E.; MARKOV, S. M.; MAYER, G. (Eds.). **Computer Arithmetic, Scientific Computation and Mathematical Modelling.** Basel: J.C.Baltzer, 1991. p.329-349. (Proceedings of SCAN'90, IMACS Annals on Computing and Applied Mathematics, Albena, Sept 24-28, 1990, 12).



- [MET85] METCALF, M. **Fortran Optimization.** New York: Academic Press, 1985.
- [MOD88] MODI, J. J. **Parallel algorithms and matrix computations.** New York: Oxford, 1988. 260p.
- [MON71] MONTEIRO, J. **Elementos de álgebra.** Rio de Janeiro: IMPA Livro Técnico, 1971. 572p.
- [MOO66] MOORE, R. E. **Interval Analysis.** Englewood Cliffs: Prentice Hall, 1966.
- [MOO79] MOORE, R. E. **Methods and applications for interval analysis.** Philadelphia: SIAM, 1979. 190p.
- [MOO85] MOORE, R. E. **Computational functional analysis.** Chichester: Ellis Horwood, 1985. 156p.
- [MOO88] MOORE, R. E. (Ed.). **Reliability in Computing: The role of interval methods in scientific computing.** San Diego: Academic Press, 1988. v.19. (Proceedings of the conference at Columbus, Ohio, Sept 8-11, 1987).
- [NAV90] NAVAUX, P. O. A. **Processadores Pipeline e processamento vetorial.** São Paulo: IME-USP, 1990. 98p. (ESCOLA DE COMPUTAÇÃO, 7, USP. São Paulo, 12-20, Jul, 1990)..
- [NEU85] NEUMAIER, A. Interval Iteration for Zeros of Systems of Equations. **Bit**, Copenhagen, v.25, p.256-273, 1985.
- [NEU90] NEUMAIER, A. **Interval methods for systems of equations.** Cambridge: Cambridge University Press, 1990. 260p.
- [ORT88] ORTEGA, J. M. **Introduction to parallel and vector solution of linear systems.** New York: Plenum Press, 1988.

- [RAL60] RALSTON, A.; WILF, H. S. **Mathematical methods for digital computers.** New York: John Wiley & Sons, 1960. v.1.
- [RAL65] RALSTON, A. **A first course in numerical analysis.** Tokyo: McGraw Hill, 1965. 578p.
- [RAT82] RATSCHEK, H.; SAVER, W. **Linear Interval Equations. Computing.** New York, v.28, p.105-115, 1982.
- [RAT87] RATSCHEK, H. **Interval Mathematics.** Freiburg: Universitat Freiburg, 1987. 44p.
- [RAT88] RATSCHEK, H.; ROKNE, J. **New computer methods for global optimization.** Chichester: Ellis Horwood, 1988. 229p.
- [RIC83] RICE, J. R. **Numerical Methods, Software, and Analysis.** Singapore: McGraw-Hill, 1983. 483p.
- [ROD80] RODRIQUE, G.; GIROUX, E. D.; PRATT, M. **Perspective on large-scale scientific computations. Computing,** New York, v.13, n.10, p.65-80, Oct, 1980.
- [ROH88] ROHN, J. **Solving Systems of Linear Equations.** In: MOORE, R. E. (Ed.). **Reliability in Computing: The role of interval methods in scientific computing.** San Diego: Academic Press, 1988. p.171-182. (Proceedings of the conference at Columbus, Ohio, Sept 8-11, 1987, 19).
- [RUM80a] RUMP, S. M. **Kleine Fehlerschranken bei Matrixproblemen.** Karlsruhe: Universitat Karlsruhe, 1980. (Dr dissertation).
- [RUM80b] RUMP, S. M.; KAUCHER, E. **Small Bounds for the Solution of Systems of Equations.** In: ALEFELD, G.; GRIGORIEFF, R. D. (Eds.). **Fundamentals of Numerical Computation (Computer-Oriented Numerical Analysis).** Wien: Springer Verlag, 1980.

- p.157-164. (COMPUTING SUPPLEMENTUM 2. Seminar held in Berlin, June 5-8, 1979).
- [RUM83] RUMP, S. M. Solving Algebraic Problems with High Accuracy. In: KULISCH, U. W.; MIRANKER, W. L. **A new approach to scientific computation.** New York: Academic Press, 1983. 384p. p.51-120.
- [RUM90] RUMP, S. M. Rigorous sensivity analysis for systems of linear and nonlinear equations. **Mathematics of Computation**, Providence, v.54, n,190, p.721-736, Apr. 1990.
- [RUM94] RUMP, S. M. **Verification Methods for Dense and Sparce Systems of Equations.** Hamburg: Technische Universität Hamburg-Harburg, 1994. 73p.
- [SAD80] SADOSKY, M. **Cálculo Numérico e Gráfico.** Rio de Janeiro: Interciência, 1980. 306p.
- [SAN76] SANTOS, V. R. B. **Curso de cálculo numérico.** Rio de Janeiro: Livro Técnico, 1976. 263p.
- [SCH87] SCHWANDT, B. Iterative Methods for System of Equations with Interval Coefficients and Linear Form. **Computing**, New York, v.38, p.143-161, 1987.
- [SIL87] SILVA, E. N. **Uma especificação para aritmética de intervalos.** Recife: CCEN Informática da UFPe, 1987. 306p. (Dissertação de mestrado).
- [ULL90a] ULLRICH, C. (Ed.). **Computer Arithmetic and self-validating numerical methods.** Boston: Academic Press, 1990. (Proceedings of SCAN'89. Basel. Oct 2-6. 1989. invited papers).

- [ULL90b] ULLRICH, C. (Ed.). **Contributions to computer arithmetic and self-validating Numerical Methods.** In: IMACS Annals on Computing and Applied Mathematics. Basel: J.C Baltzer, 1990. v.7 (Proceedings of SCAN'89, Basel, Oct 2-6, 1989, submitted papers).
- [VAN78] VANDERGRAFT, J. S. **Introduction to numerical computations.** New York: Academic Press, 1978. 344p.
- [WES68] WESTLAKE, J. R. **A Handbook of Numerical Matrix Inversion and Solution of Linear Equations.** New York: Willy, 1968.

## 10 ANEXOS

### 10.1 Recomendação IMACS/GAMM

Avanços em tecnologia computacional são agora tão profundos que a capacidade aritmética e o repertório dos computadores podem e deveriam ser expandidos. Os arredondamentos e operações aritméticas elementares (incluindo arredondamentos e operações intervalares) de qualidade proporcionados pelo Padrão IEEE para aritmética de ponto-flutuante [2,3] deveriam ser estendidos aos espaços produtos da computação (vetores, matrizes e números complexos, etc.). Esta proposta dá os detalhes essenciais para fazer isto. A nova capacidade computacional expandida é obtida com um custo modesto. Extraordinariamente, a nova metodologia não implica em penalizar o desempenho. Além disso, técnicas são agora disponíveis para que com esta capacidade expandida, o computador por si mesmo possa ser usado para avaliar a qualidade e a confiabilidade dos resultados calculados para uma grande faixa de aplicações.

#### 10.1.1 Escopo

Esta proposta é um documento oficial da GAMM e IMACS. Ela define as propriedades matemáticas que as operações aritméticas em ponto-flutuante, especialmente operações vetoriais compostas, deveriam satisfazer (veja também [17]). Isto pode ser visto como uma extensão da abordagem dos Padrões IEEE para aritmética em ponto-flutuante (754 e 854) [2, 3] puramente escalar para aritmética vetorial/matricial [6, 7, 8, 12, 22, 25, 26, 27]. Contudo, melhor que ser prescritivo ao nível de bit, esta proposta aplica-se a muitas plataformas de *hardware* e formatos em ponto-flutuante. Ela pode também ser usada para especificar o comportamento matemático das operações matemáticas em linguagens de programação. Um sistema equipado com a aritmética proposta é uma plataforma ideal sobre a qual técnicas de verificação automática do resultado podem ser realizadas [1, 16, 21, 23, 24, 36, 37]. Os objetivos principais desta proposta são a confiança e a segurança dos resultados numéricos. Parte das características propostas podem ser proporcionadas em *software*, se bem que um bom suporte de *hardware* é fortemente recomendado. O *hardware* existente e implementações de *software* demonstram que aritmética de ponto-flutuante vetorial exata é praticável usando a tecnologia atual - com um custo modesto e, geralmente, sem penalizar o tempo de execução.

#### 10.1.2 Definição de aritmética computacional

Um formato de ponto-flutuante é definido por sua base  $b$  (*radix*), sua precisão  $p$  (tamanho da mantissa), e seu expoente mínimo e máximo,  $e_{min}$  e  $e_{max}$ . Estes quatro parâmetros definem um formato de ponto-flutuante  $F(b, p, e_{min}, e_{max})$  e, então, o conjunto finito de números reais representáveis com este formato são os **números em**



**ponto-flutuante**  $F$ . Por conveniência, é assumido que um formato particular em ponto-flutuante é dado.

Um arredondamento  $\text{rnd}(x)$  é função monotônica, não decrescente, de números reais  $\mathbb{R}$  sobre o conjunto dos números em ponto-flutuante  $F \subset \mathbb{R}$ , que deixa todos elementos de  $F$  inalterados. A monotonicidade elimina comportamentos irregulares de um arredondamento. Formalmente, as duas seguintes propriedades devem valer:

$$\text{rnd}(x) = x \quad \text{se } x \in F \quad (\text{projeção}) \quad (10.1)$$

$$x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y) \quad \text{para } x, y \in \mathbb{R} \quad (\text{monotonicidade}) \quad (10.2)$$

Geralmente arredondamentos usados também são anti-simétricos:

$$\text{rnd}(-x) = -\text{rnd}(x) \quad \text{para todos } x \in \mathbb{R} \quad (\text{anti-simetria}) \quad (10.3)$$

Note que (10.3) impõe a propriedade da simetria sobre  $F$ , assim se  $x \in F$ , então  $-x \in F$ . Exemplos de arredondamentos anti-simétricos são os arredondamentos para zero (truncamento), o arredondamento longe de zero (para cima), e certos arredondamentos para o mais próximo. Há muitas maneiras de definir um arredondamento para o número em ponto-flutuante mais próximo.

Outros arredondamentos importantes são os **arredondamentos direcionados** para  $-\infty$  (para baixo, denotado por  $\nabla$ ) e para  $+\infty$  (para cima, denotado por  $\Delta$ ), que são ambos igualmente definidos por (10.1), (10.2), e a propriedade adicional

$$\nabla x \leq x \leq \Delta x \quad \text{para todos } x \in \mathbb{R} \quad (\text{arredondamento direcionado}) \quad (10.4)$$

Um overflow de expoente poderia ser mapeado para  $+\infty$  ou  $-\infty$ , conforme o caso. Os arredondamentos direcionados não são anti-simétricos, mas satisfazem a regra:  $\nabla(-x) = -\Delta(x)$ .

Arredondamentos são definidos exatamente da mesma maneira para **números complexos** e para os **espaços produtos** básicos (vetores/matrizes reais e complexos). Em todos esses espaços, a relação de ordem  $\leq$  é definida por componente, induzindo a uma ordem parcial sobre esses conjuntos. Se  $T$  denota um destes conjuntos e  $S$  seu subconjunto representável no computador, um arredondamento de  $T$  em  $S$  satisfaz as mesmas propriedades (10.1), (10.2), (10.3) com  $\mathbb{R}$  substituído por  $T$  e  $F$  por  $S$  (exemplo, se  $T = \mathbb{R}^n$ , então  $S = \mathbb{F}^n$ ):

$$\text{rnd}(x) = x \quad \text{se } x \in S \quad (\text{projeção}) \quad (10.5)$$

$$x \leq y \Rightarrow \text{rnd}(x) \leq \text{rnd}(y) \quad \text{para } x, y \in T \quad (\text{monotonicidade}) \quad (10.6)$$

$$\text{rnd}(-x) = -\text{rnd}(x) \quad \text{para todos } x \in T \quad (\text{anti-simetria}) \quad (10.7)$$

Note que (10.7) impõe a propriedade de simetria em  $S$ . A teoria da aritmética computacional mostra que esses arredondamentos são equivalentes a aplicar arredondamentos análogos a componentes individuais do vetor, ou da matriz, ou nas partes real e imaginária de um número complexo separadamente [26, 27].

Para os correspondentes espaços intervalares - os intervalos sobre os números reais e complexos, assim como os intervalos sobre os vetores/matrizes reais e complexos - a relação de ordem é a relação de subconjunto  $\subseteq$  (em (10.6)). Um arredondamento de qualquer conjunto intervalar  $T$  em seu subconjunto  $S$  representável no computador é definido pelas propriedades (10.5), (10.6) (com  $\leq$  trocado por  $\subseteq$ ), mais a propriedade adicional

$$x \subseteq \text{rnd}(x) \quad \text{para todos } x \in T \quad (\text{inclusão}) \quad (10.8)$$

Estes arredondamentos intervalares são também anti-simétricos, isto é, eles satisfazem (10.8).

Quando uma operação aritmética em um dos espaços básicos (os números reais e complexos, vetores/matrizes e os correspondentes espaços intervalares) é modelado no computador, ela será definida como segue:

Seja  $T$  um dos espaços segundo consideração em que operações aritméticas são definidas e, seja  $S$  seu subconjunto representável no computador. Se  $\circ$  é uma operação aritmética em  $T$ , a correspondente operação no computador  $\boxplus$  em  $S$  é dada por

$$r \boxplus s := \text{rnd}(r \circ s) \quad \text{para todos } r, s \in S \quad (\text{operação arredondada}) \quad (10.9)$$

onde  $\text{rnd}$  é um arredondamento. Isto é, a operação no computador deve ser efetuada **como se** o resultado exato fosse primeiro calculado e, então, arredondado com o arredondamento selecionado. O arredondamento  $\text{rnd}$  unicamente determina a operação arredondada  $\boxplus$  no computador. Com a exceção de  $\nabla$  e  $\Delta$ , geralmente os arredondamentos usados são anti-simétricos. Lembrar que no caso de intervalos, o arredondamento deve adicionalmente satisfazer o requerimento de inclusão (10.8). Um mapeamento de  $T$  em  $S$  satisfazendo (10.5), (10.6), (10.7), e (10.9) (e (10.8) no caso de intervalos) é chamado de semimorfismo. Muitas propriedades de ambas as estruturas de ordem e estruturas algébricas são invariantes sob um semimorfismo. Para mais detalhes, veja [26, 27].

É facilmente visto que a propriedade (10.9) garante que todas as operações aritméticas são exatas a pelo menos 1 *ulp* (unidade na última casa decimal).  $1/2$  *ulp* é alcançada no caso de arredondamento para o mais próximo. Esta exatidão é alcançada por operações elementares em ponto-flutuante definidas por várias implementações incluindo todas aquelas de acordo com os padrões IEEE (754 ou 854) [2,3].

Uma análise cuidadosa mostra que todas as operações aritméticas em consideração (com uma exceção) podem ser expressas em termos das quatro operações aritméticas elementares  $+$ ,  $-$ ,  $*$ ,  $/$  para números em ponto-flutuante mais uma operação adicional (que é uma operação composta), o produto escalar (exato) de dois vetores com componentes em ponto-flutuante (denotado por  $\cdot$ ). Esta quinta operação é definida na caixa abaixo. Para implementar as correspondentes operações intervalares, a teoria da aritmética computacional mostra que as mesmas cinco operações ( $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\cdot$ ) com os arredondamentos dirigidos para  $-\infty$  (para baixo,  $\nabla$ ) e para  $+\infty$  (para cima,  $\Delta$ ) são suficientes. A exceção, divisão complexa (intervalar), pode ser realizada iterativamente [26, 27, 9].

Resumindo, operações semimórficas para números reais e complexos, vetores e matrizes assim como para intervalos reais e complexos, vetores e matrizes intervalares podem ser realizadas em termos de 15 operações aritméticas fundamentais em aritmética de ponto-flutuante:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\cdot$ , cada uma com os arredondamentos  $\lfloor \cdot \rfloor$ ,  $\nabla$ , e  $\Delta$ , onde  $\lfloor \cdot \rfloor$  tolera um arredondamento anti-simétrico (tipicamente para o mais próximo). Os símbolos  $\lfloor \cdot \rfloor$ ,  $\nabla$  e  $\Delta$  (onde  $\circ \in \{+, -, *, /, \cdot\}$ ) representam as operações definidas por (10.9) usando um arredondamento anti-simétrico  $\lfloor \cdot \rfloor$  (semimorfismo), o arredondamento para baixo  $\nabla$ , e o arredondamento para cima  $\Delta$ , respectivamente.

Essas 15 operações fundamentais são encontradas em uma variedade de sistemas de programação [14, 15, 16, 23, 24]. Doze das 15 operações são proporcionadas pelos padrões IEEE. Adicionando somente mais três, as operações aritméticas em todos os espaços produtos básicos da matemática podem ser realizados com 1 (ou 1/2) *ulp* de exatidão (em cada componente).

Em análises numéricas, o produto escalar é "onipresente". Ele aparece em aritmética complexa, aritmética vetorial e matricial, aritmética de múltipla precisão, técnicas de refinamento iterativo, entre muitos outros lugares.

Para completar a especificação da aritmética computacional, será dado agora uma definição formal de uma **operação de produto escalar exato**.

Dados dois vetores  $x$  e  $y$  com  $n$  componentes em ponto-flutuante cada e arredondamento prescrito  $\text{rnd}$ , o resultado  $s$  em ponto-flutuante da operação de produto escalar (aplicada para  $x$  e  $y$ ) é definida por (10.10) onde todas as operações aritméticas são exatas. Em outras palavras,  $s$  será calculado **como se** um resultado intermediário  $\bar{s} = x \cdot y$  fosse correto para precisão infinita e com faixa de expoente ilimitada e, então, arredondado para o formato em ponto-flutuante desejado de acordo com o arredondamento  $\text{rnd}$  selecionado.

$$s := \text{rnd}(\bar{s}) := \text{rnd}(x, y) = \text{rnd}\left(\sum_{i=1}^n x_i * y_i\right), \quad (10.10)$$

Essa definição da operação de produto escalar garante a mais alta exatidão possível (para o dado arredondamento e o formato destino em ponto-flutuante). Ela é análoga à definição da aritmética escalar em ponto-flutuante dada nos padrões IEEE[2, 3].

É importante perceber que a ordem em que as operações elementares são realizadas quando determinam o produto escalar não é especificada, e que esta acomoda o processamento *pipelinizado* (e, potencialmente, paralelo). A reordenação das parcelas não tem influência nos resultados calculados desde que todos os resultados intermediários sejam exatos.

Em contraste, note que o cálculo tradicional do produto escalar de dois vetores com  $n$  componentes cada ( em aritmética de ponto-flutuante comum com multiplicações e adições arredondadas) envolve  $2n-1$  arredondamentos. Se ocorrerem cancelamentos catastróficos, um grande número de dígitos significativos podem ser perdidos [13, 35]. Isto pode acontecer mesmo se um formato de precisão estendida dos dados for usada para o somatório. Perda de parte da exatidão, uma considerável quantidade de tempo de processamento pode ser requerido para realizar passos intermediários gratuitos tais como composição, decomposição, normalização e, arredondamento de valores em ponto-flutuante intermediários. Além disso, operações desnecessárias de carregar e armazenar podem ter de ser realizados no modo tradicional.

### 10.1.3 Opções de implementação

Implementações das operações elementares em ponto-flutuante com diferentes arredondamentos são triviais. A realização computacional das novas operações de produto escalar podem ser realizadas de diferentes maneiras [14, 8]. Uma maneira de somar  $n$  números ou produtos é via acumulação de ponto-fixo. A escala que acompanha cálculos em ponto-fixo é eliminada por uma implementação de *hardware* que tem tido uso por mais do que dez anos. Este mecanismo emprega um registrador longo de ponto-fixo (acumulador) abrangendo duas vezes a faixa de expoente do formato em ponto-flutuante na qual os componentes do vetor são dados (veja por exemplo [14]). Se um número relativamente pequeno de dígitos extras é proporcionado na frente (parte mais significativa) semelhante a um registrador para coletar "vai-um" extras, o resultado intermediário em tal registrador é sempre exato e não pode ser overflow ou underflow para qualquer número praticável de somadores. Somente um arredondamento ocorre, e esse é no fim do processo de acumulação. Este método tem a vantagem de ser mais simples, direto e sempre proporciona a resposta exata desejada.

Implementações existentes e estudos detalhados mostram que uma implementação de *hardware* de um registrador longo é rotineira e inexpressiva em unidades adicionais de silício e podem ser feitas para executar tão rápido quanto a acumulação em ponto-flutuante convencional. As operações fundamentais que são necessárias são: inicialização do registrador longo, adição de um produto de dois números em ponto-flutuante para o registrador e, arredondar o conteúdo do registrador para um número em ponto-flutuante (usando o arredondamento escolhido). Outras operações desejadas incluem: adição de um



número em ponto-flutuante para o registrador, adição e comparação de dois registradores longos. Extensões de linguagens existentes de programação proporcionam e exploram essas operações [14, 15, 16, 23, 24].

Um número de estudos em pesquisas de uma solução que irá requerer menos silício que um registrador longo estão abaixo desta. Uma alternativa é uma janela (registrador curto) onde somente um segmento do registrador longo é realmente provido em *hardware*. Isto pode ser justificado desde que os cálculos envolvendo expoentes muito longos ou muito curtos raramente ocorrem. Na sua forma mais simples, a janela é um seguimento de tamanho fixo cobrindo uma faixa fixa de expoentes. Uma variante mais sofisticada permite a janela ser transferida para a esquerda (para acomodar os valores intermediários maiores), mas nunca para a direita, desta maneira mantendo uma janela na parte mais significativa do produto escalar. Contudo, qualquer somador que não caiba completamente requer tratamento caro, provavelmente através de *software* especial.

Há métodos iterativos bem conhecidos para computar somas e produtos escalares exatamente que usam aritmética em ponto-flutuante somente[5, 33]. Um destes procede pela troca sucessiva do conjunto completo de somadores, usando uma técnica de correção. Em um número finito de passos, o conjunto de somadores é dirigido para um estado onde nenhum somador coincide com outro num sentido de ponto-fixa, e então o somatório é corretamente completado. É importante notar e explorar o fato que a soma corretamente arredondada pode estar disponível muito mais cedo no processo, frequentemente depois de uma interação. Veja [5] para um exemplo de critério de parada. Esses métodos exigem operações exatas +, -, \* que podem ser fornecidas representando o resultado com 2 números em ponto-flutuante, uma parte de alta-ordem e uma parte de baixa-ordem [9]. Contudo, esses métodos iterativos têm algumas desvantagens maiores: o número de iterações e a quantidade de memória requerida para obter uma resposta exata não são conhecidos *a priori*, e *overflows* e *underflows* intermediários não podem ser evitados sem uma degradação significativa do desempenho.

Do ponto de vista do usuário, o bem-definido, comportamento prognostical do registrador longo é altamente desejável. Qualquer apoio parcial de *hardware* exige *software* complementar. Fabricantes que se recusam a proporcionar uma implementação completa do produto escalar em *hardware* devem estar conscientes do fato de que pode ser uma penalidade substancial no desempenho em aplicações essenciais.

#### 10.1.4 Referências bibliográficas da recomendação

- [1] Adams, E.; Kulisch, U.(eds.): Scientific Computing with Automatic Result Verification. Academic Press, Orlando, 1992.
- [2] ANSI /IEEE: IEEE Standard for Binary Floating-Point Arithmetic. ANSI/ IEEE Std 754-1985, New York, 1985.
- [3] ANSI/IEEE: IEEE Standard for Radix-Independent Floating-Point Arithmetic. ANSI/IEEE Std 854-1987, New York, 1987.



- [4] Bleher, J.H.; Rump, S.M.; Kulisch, U.; Metzger, M.; Ullrich, Ch.; Walter, W.: FORTRAN-SC: A Study of a FORTRAN Extension for Engineering/Scientific Computation with Access to ACRITH. Computing 39, 93-110, Springer-Verlag, 1987.
- [5] Bohlender, G.: Genaue Berechnung mehrfacher Summen, Produkte und Wurzeln von Gleitkommazahlen und allgemeine Arithmetik in höheren Programmiersprachen. Ph.D. thesis, Univ. Karlsruhe, 1978.
- [6] Bohlender, G.: What do we need beyond IEEE Arithmetic ? In [36], 1-36, 1990
- [7] Bohlender, G.: A vector Extension of the IEEE Standard for Floating-Point Arithmetic. In [21], 3-12, 1991.
- [8] Bohlender, G.; Knöfel, A.: A Survey of Pipelined Hardware Support for Accurate Scalar Products. In [21], 3-12, 1991.
- [9] Bohlender, G.; Kornerup, P.; Matula, D.W.; Walter, W.V.: Semantics for Exact Floating Point Operations, Proc. of 10th IEEE Symp. on Computer Arithmetic (ARITH 10) in Grenoble, 22-26, IEEE Comp. Soc., 1991.
- [10] Cordes, D.: Runtime System for a PASCAL-XSC Compiler. In [21], 151-160, 1991.
- [11] Dekker, T.J.: A Floating-Point Technique for Extending the Available Precision. Numerical Mathematics 18, 224-242, 1971.
- [12] Hahn, W.; Mohr, K.: APL/PCXA, Erweiterung der IEEE Arithmetik für Technisch wissenschaftliches Rechnen. Hanser Verlag, München, 1989.
- [13] Hammer, R.: How Reliable is the arithmetic of Vector Computers ? In [37], 467-482, 1990.
- [14] IBM: System/370 RQP, High-Accuracy Arithmetic. SA22-7093-0, IBM Corp., 1984.
- [15] IBM: High-Accuracy Arithmetic Subroutine Library (ACRITH), General Information Manual. 3rd ed., GC33-6163-02, IBM Corp., 1986.
- [16] IBM: High-Accuracy Arithmetic- Extended Scientific Computation (ACRITH-XSC), General Information. GC33-6461-01, IBM Corp., 1990.
- [17] IMACS, GAMM: Resolution on Computer Arithmetic. In Mathematics and Computers in Simulation 31, 297-298, 1989; in Zeitschrift für Angewandte Mathematik und Mechanik 70, no.4, p. T5, 1990; in [36], 301-302, 1990; in [37], 523-524, 1990; in [21], 477-478, 1991.
- [18] ISO: Language Independent Arithmetic Standard (LIA-1). Second Committee Draft Standard (Version 4.0), ISO/IEC CD 10967-1, 1992.
- [19] Kahan, W.: Further Remarks on Reducing Truncation Errors. Comm. ACM 8, no. 1, 40, 1965.
- [20] Kahan, W.: Double Precision IEEE Standard 754 Floating-Point Arithmetic. Mini-Course on "The Regrettable Failure of Automated Error Analysis", Conf. on Computers and Mathematics, MIT, June 13, 1989.
- [21] Kaucher, E.; Markov, S.M.; Mayer, G.(eds): Computer Arithmetic, Scientific Computation and Mathematical Modelling. IMACS Annals on Computing Applied Mathematics 12, J.C. Baltzer, Basel, 1991.
- [22] Kirchner, R.; Kulisch, U.: Arithmetic for Vector Processors. Proc. of 8th IEEE Symp. on Computer Arithmetic (ARITH8) in Como, 256-269. IEEE Comp. Soc., 1987.

- [23] Klatte, R.; Kulisch, U.; Lawo, C.; Rauch, M.; Wiethoff, A.: C-XSC: A C++ Class Library for Extended Scientific Computing. Springer-Verlag, Berlin, Heidelberg, New York, 1993.
- [24] Klatte, R.; Kulisch, U.; Neaga, M.; Ratz, D.; Ullrich, Ch.: PASCAL-XSC Sprachbeschreibung mit Beispielen. Springer-Verlag, Berlin, Heidelberg, New York, 1991.
- [25] Knöfel, A.: Fast Hardware Units for The Computation of Accurate Dot Products. Proc. of 10th IEEE Symp. on Computer Arithmetic (ARITH 10), 70-74, IEEE Comp. Soc., 1991.
- [26] Kulisch, U.: Grundlagen des numerischen Rechnens : Mathematische Begründung der Rechnerarithmetik. Reihe Informatik 19, Bibl. Inst. Mannheim, 1976.
- [27] Kulisch, U.; Miranker, W.L.: Computer Arithmetic in Theory and Practice. Academic Press, New York, 1981.
- [28] Linnainmaa, S.: Analysis of Some Known Methods of Improving the Accuracy of Floating-Point Sums. BIT 14, 167-202, 1974.
- [29] Lohner, R.; Wolff v. Gudenberg, J.: Complex Interval Division with Maximum Accuracy. Proc. of 7th IEEE Symp. on Computer Arithmetic.
- [30] Moller, O.: Quasi Double Precision in Floating-Point Addition. BIT 5, 37-50, 1965.
- [31] Müller, M.; Rüb, Ch.; Rülling, W.: Exact Addition of Floating Point Numbers. Sonderforschungsb. 124, FB 14, Informatik, Univ. des Saarlandes, Saarbrücken, 1990.
- [32] NAG: Basic Linear Algebra Subprograms (BLAS). The Numerical Algorithms Group Ltd, Oxford, 1990.
- [33] Pichat, M.: Correction d'une somme en arithmétique à virgule flottante. Numer. Math. 19, 400-406, 1972.
- [34] Priest, D. M.: Algorithms for Arbitrary Precision Floating Point Arithmetic. Proc. of 10th IEEE Symp. on Computer Arithmetic (ARITH 10), 132-143, IEEE Comp. Soc., 1991.
- [35] Ratz, D.: The Effects of the Arithmetic of Vector Computers on Basic Numerical Methods. In [37], 499-514, 1990.
- [36] Ullrich, Ch.(ed.): Computer Arithmetic and Self-Validating Numerical Methods. Academic Press, San Diego, 1990.
- [37] Ullrich, Ch.(ed.): Contributions to Computer Arithmetic and Self-Validating Numerical Methods. IMACS Annals on Computing and Applied Mathematics 7, J.C. Baltzer, Basel, 1990.

## 10.2 Livros existentes sobre matemática intervalar e tópicos relacionados

- [ADA87] ADAMS, E; ANSORGE, R.; CIROSSMANN, Ch; ROSS, H. G. **Discretization in Differential Equations and Enclosures**. Mathematische Firschung, Band 36, Berlin: Akademie Verlag, 1987.
- [ADA93] ADAMS,E; KULISCH,U. (Eds) **Scientific Computing with automatic result verification**. San Diego: Academic Press, 1993.
- [ALB77] ALBRECHT, R; KULISCH, U. (Eds) **Grundlagen der Computerarithmetik**. Computing Supplementum 1, Wien: Spring Verlag, 1977.
- [ALB93] ALBRECHT,R; ALEFELD,G; STETTER,H.J (Eds) **Validation numerics- Theory and applications**. Computing Supplementum 9 Wien: Springer Verlag, 1993.
- [ALE80] ALEFELD,G.; GRIGORIEFF,R.D. (Eds) **Fundamentals of numerical computation(computer-oriented numerical analysis)**. Proceedings of seminar held in Berlin, june 5-8, 1979. Computing suplementum 2, Wien: Springer Verlag, 1980.
- [ALE83] ALEFELD, G; HERZBERGER, J. **An introduction to interval computations**. New York: Academic Press, 1983.
- [AME89] AMES,W.F.(Ed) **Numerical and applied mathematics**. Basel: J.C Baltzer Scientific Publishing, 1989.
- [AND91] ANDREEV, A. S; DIMOV, I. T; MARKOV, S. M; ULLRICH,Ch. (Eds) **Mathematical Modelling and scientific computations**. Sofia: Bulgarian Academy Of Sciences, 1991.
- [ATA92] ATANASSOVA,L; HERZBERGER,J (Eds) **Computer Arithmetic and enclosure methods**. Proceedings os SCAN 91. North Holland: Elsevier Science Publishers B.V, Amsterdam, 1992.
- [BAU87] BAUCH,H; JAHN,K.U.; OELSCHOGEL,D; SUSSE,H; WIEBIGUE,V. **Intervall mathematik, theory und anwendungen**. Leipzig: BG Teubner Verlagsgesellschaft, 1987.
- [BOH87] BOHLENDER,G; RALL,L.B; ULLRICH,Ch; WOFF,V.; GUTENBERG,J. **PASCAL-SC A computer language for scientific computation**. Perspectives in computing, v.17, Orlando: Academic Press, 1987.
- [BRE92] BREZINSKY, C; KULISCH, U. (Eds) **Computational and applied mathematics I. Algorithms and Theory**. See also [Vic91]. Proceedings of the 13th IMACS World Congress, Dublin: Ireland, Elsevier Science Publischers B V, North Holland: 1992.
- [BUC86] BUCHBERGER, B; KUTZLER, B.; FREILMEIER, M; KRATZ, M; KULISCH, U; RUMP, S. M. **Rechnerorientierte Verfahren**. Stuttgart: B.G Teubner Verlag, 1986.
- [EIN92] EINARSSON, B; FOSDICK, L. D; GAFFNEY, P; HOUSTIN, E. (Eds) - **Programming environments for high level scientific problem solving**. Proceedings of IFIP WG 25 Conference. Karlsruhe, Sept.23-27, 1991. Elsevier Publischers B.V, 1992.

- [FOR84] FORD, B; RAULT, J. C; THOMASSED,F. (Eds) **Tools, methods and Languages for Scientific and Engineering computation**. Elsevier: North Holland, 1984.
- [GRI91] GRIEWANK,A; CORLISS,G (Eds) **Automatic Differential of Algorithms:Theory, implementation and applications**. Proceedings of workshop on automatic differentiation at Breckenridge, Philadelphia: SIAM,1991.
- [HAM93] HAMMER,R; HOCKS,M; KULISCH,U; RATZ,D. **Numerical Toolbox for Verified Computing I: basic numerical problems**. Berlin, Springer Verlag, 1993.
- [HWA79] HWANG,K. **Computer Arithmetic: Principles, Architecture and Design**. New York: J.Wiley, 1979.
- [JAH91] JAHN, K-U (Ed) **Computerarithmetik mit ergebnisverifikation problemseminar, technische Hochschule Leipzig**. March 13-15,1991. Proceedings in Wissenschaftliche zeitschrift der technischen Hochschule. Leipzig: Jahrgang 15, heft 6, 1991.
- [KAU84] KAUCHER, E; MIRANKER, W. L. **Self-Validating numerics for functions Space Problems**. New York: Academic Press, 1984.
- [KAU87] KAUCHER, E; KULISCH, U; ULLRICH, Ch (Eds) **Computerarithmetik: scientific computing and programming languages**. Stuttgart: BG Teubner Verlag, 1987.
- [KAU91] KAUCHER,E; MARKOV,S.M; MAYER,G (Eds) **Computer Arithmetic, Scientific Computation and Mathematical Modelling**. Proceedings of SCAN'90 Albena, Sept 24-28, 1990. IMACS Annals on Computing and Applied Mathematics, v.12. Basel: J.C.Baltzer, 1991.
- [KLA92] KLATTE, R; KULISC, U; NEAGA, M; RATZ, D; ULLRICH, Ch **PASCAL-XSC language reference with examples**. Berlin: Springer Verlag, 1992.
- [KLA93] KLATTE,R; JULISCH,U;WIETHOFF,A; LAWOC; RAUCH,M.- **C-XSC A C++ class library for extended scientific computing**. Berlin: Springer Verlag, 1993. (livro)
- [KUL81] KULISCH, U; MIRANKER,W.L. **Computer arithmetic in theory and Practice**. New York: Academic Press, 1981.
- [KUL82] KULISCH, U; ULLRICH, Ch (Eds) **Wissenschaftliches Rechnen und Programmiersprachen**. Berichte des German Chapter of the ACM, Band 10. Seminar held in Karlsruhe at April 2-3, 1982.
- [KUL83] KULISCH, U; MIRANKER, W. L. (Eds) **A new approach to scientific computation**. Proceedings of symposium held at IBM Research Center, Yorktown Heights, 1982. New York: Academic Press, 1983.
- [KUL88] KULISCH, U; STETTER,H.J (Eds) **Scientific computation with automatic result verification**. Seminar held in Karlsruhe Sept.30-Oct.2, 1987. Computing supplementum 6, Wien: Springer Verlag, 1988.
- [KUL89] KULISCH, U. (Ed)- **Wissenschaftliches Rechner mit Ergebnisverifikation eine einfuhrung**. v.58, Akademie Verlag, Berlin und Vieweg Verlagsgesellschaft, Wiesbaden, 1989.



- [MEY91] MEYER,R.K; SCHMIDT,D.S (Eds) **Computer Aided proofs in analysis** **Spring Verlag**, (the IMACS volumes in Mathematics and its applications, v.25. New York: 1991.
- [MIT93] MITCALF, M; RED,J. **Fortran 90 explained**. Oxford, Oxford University Pres, 1993.
- [MIR86] MIRANKER,W.L; TOUPIN,R.A. **Accurate scientific computations**. Symposium held in Bad Neuenahr (Germany) March 12-14, 1985. Lecture Notes in Computer Science n.235. Berlin: Springer Verlag, 1986.
- [MOO66] MOORE, RAMON,E **Interval Analysis**. Englewood Cliffs, Prentice Hall, 1966.
- [MOO79] MOORE, R E. **Methods and applications of interval analysis**. Philadelphia: SIAM, 1979.
- [MOO88] MOORE,R E (Ed) **Reliability in Computing: The role of interval methods in scientific computing**. PERSPECTIVES IN COMPUTING V 19. Proceedings of the conference at Columbus, Ohio, Sept 8-11,1987. San Diego: Academic Press, 1988.
- [NEU90] NEUMAIER A **Interval methods for systems of equations**. Cambridge: Cambridge University Press, 1990.
- [NIC80] NICKEL,K (Ed) **Interval Mathematics 1980**. Proceedings of the international symposium Freiburg 1980. New York: Academic Press, 1980.
- [NIC85] NICKEL,K (Ed) **Interval Mathematics 1985**. Proceedings of the international Symposium, Freiburg 1985. Vienna: Springer Verlag, 1986.
- [NIC88] NICKEL,K.(ed) **Freiburger Intervallberichte from 78/1 to 87/10**. Author index, subject index, Institut für Angewandte Mathematik, Universität Freiburg, Herman Helder Strasse 10, D-79104, Freiburg: 1988.
- [RAT88] RATSCHKE,H; RONKE,J. **New computer methods for Global Optimization**. Chichester, Ellis Hor Wood, 1988.
- [RIC71] RICE, JOHN R. (Ed) **Mathematical software**. Proceedings of symposium held at Perdue University, april 1970. New York: Academic Press, 1971.
- [RIC77] RICE, J.R. (ed) **Mathematical Software III**. Proceedings New York: Academic Press, 1977.
- [RUS86] RUSCHITZKA, M. (Ed) **Computer systems: performance and simulation**. Proceedings of 11th IMACS word congress on system simulation and scientific computation, Aug. 5-9, 1985. Oslo (Italy) Preprints see [Wah85]. Elsevier Science publisher BV, North Holland, 1986.
- [SYD88] SYDOW,A.(Ed) **Third international symposium on systems analysis and simulation**. Humboldt University, Berlin: sept.12-16, 1988.



- [ULL89] ULLRICH,Ch; WOLFF, V.; Gudenberger,J. (Eds) **Accurate Numerical Algorithms, a collection of diamond research papers, project 1072.** DIAMOND. Berlin: Springer Verlag, 1989.
- [ULL90a] ULLRICH,Ch (Ed) **Computer Arithmetic and self-validating numerical methods.** Proceedings of SCAN'89, invited papers. Basel, Oct 2-6, 1989. San Diego: Academic Press, 1990.
- [ULL90b] ULLRICH,Ch (Ed) **Contributions to computer arithmetic and self-validating Numerical Methods.** Proceedings of SCAN'89, submitted papers. Basel, Oct 2-6, 1989. IMACS Annals on Computing and Applied Mathematics, v.7. Basel: J.C Baltzer, 1990.
- [VIC91] VICHNEVETSKY, R; MILLER, J. J. H (Eds) **IMACS'91**, 13th World Congress on computation and applied Mathematics. Proceeding in 4 volumes. See also [Bre92]. July 22-26, 1991. Trinity College, Dublin: Ireland, 1991.
- [WAH85] WAHLSTROM, B; HENSIKSEN, R; SUNDBY,N.P (Eds) **Proceedings of the 11th IMACS World Congress on system simulation and scientific computation.** 5 volumes. Aug 5-9, 1985. Oslo. Proceedings publische in [Rus86].

### 10.3 Tabela das constantes de máquina

A tabela abaixo mostra os valores das constantes de máquina definidas para o ambiente do Pascal XSC (PC486) e para o Cray. Elas são utilizadas para definir os domínios das funções elementares intervalares definidas pela tabela 5.5.

constante	valor em Pascal XSC	valor no Cray
sqrmax	1.340 780 792 994 259 6 . $10^{154}$	5.215 361 924 162 1 . $10^{1232}$
expmax	7.097 827 128 933 84 . $10^2$	5.676 872 886 111 . $10^3$
expl0max	3.082 547 155 599 167 . $10^2$	2.465 434 568 904 . $10^3$
one-eps	0.999 999 999 999 999 9	0.999 999 999 999 996 40
one+eps	1.000 000 000 000 000 2	1.000 000 000 000 007 00
maxreal	1.797 693 134 862 315 8 . $10^{308}$	0.273 . $10^{2466}$
minreal	4.940 656 458 412 465 4 . $10^{-324}$	0.367 . $10^{-2465}$

## 10.4 Extrato da listagem da *libavi.a*

```

libavi.f90
include 'inter.inc'
interface assignment (=)
interface operator (+)
interface operator (-)
interface operator (*)
interface operator (/)
interface operator (==)
interface operator (/=)
interface operator (<)
interface operator (>)
interface operator (<=)
interface operator (>=)
interface sin
interface cos
interface tan
interface exp
interface ln
interface log
interface sqr
interface sqrt
interface atan
subroutine scopy (A,B)
    type (interval), intent(out) :: A
    type (interval), intent(in) :: B
subroutine sswap (A,B)
    type (interval), intent (inout) :: A,B
function sintpt (A) result (C)
    real, intent(in) :: A
    type (interval) :: C
function sintval (A,B) result (C)
    real, intent(in) :: A,B
    type (interval) :: C
function sinf (A) result (C)
    type (interval), intent(in) :: A
    real :: C
function ssup (A) result (C)
    type (interval), intent(in) :: A
    real :: C
function smod (A) result (C)
    type (interval), intent(in) :: A
    real :: C
    real :: M1,M2
function smax (A) result (C)
    type (interval), intent(in) :: A
    real :: C
    real :: M1,M2
function smin (A) result (C)
    type (interval), intent(in) :: A
    real :: C
    real :: M1,M2

```

```

function sdis (A,B) result (C)
    type (interval), intent(in) :: A,B
    real :: C
    real :: x1,x2,M1,M2
function sdia (A) result (C)
    type (interval), intent(in) :: A
    real :: C
function sdiar (A) result (C)
    type (interval), intent (in) :: A
    real :: C
function sraio (A) result (C)
    type (interval), intent(in) :: A
    real :: C
function smed (A) result (C)
    type (interval), intent(in) :: A
    real :: C
    real :: x
function sequ (A,B) result (C)
    type (interval), intent(in) :: A,B
    logical :: C
function sless (A,B) result (C)
    type (interval), intent(in) :: A,B
    logical :: C
function sgre (A,B) result (C)
    type (interval), intent(in) :: A,B
    logical :: C
function sleq (A,B) result (C)
    type (interval), intent(in) :: A,B
    logical :: C
function sgeq (A,B) result (C)
    type (interval), intent(in) :: A,B
    logical :: C
function rins (A,B) result (C)
    real, intent(in) :: A
    type (interval), intent(in) :: B
    logical :: C
function sneq (A,B) result (C)
    type (interval), intent(in) :: A,B
    logical :: C
function sxiny (A,B) result (C)
    type (interval), intent (in) :: A,B
    logical :: C
function sxcy (A,B) result (C)
    type (interval), intent (in) :: A,B
    logical :: C
function sxmy (A,B) result (C)
    type (interval), intent (in) :: A,B
    logical :: C
function siv (A,B) result (C)
    type (interval), intent (in) :: A,B
    logical :: C
function suni (A,B) result (C)
    type (interval), intent(in) :: A,B
    type (interval) :: C

```

```

    real :: M1,M2
function sunid (A,B) result (C)
    type (interval), intent(in) :: A,B
    type (interval) :: C
    real :: M1,M2
function sinter (A,B) result (C)
    type (interval), intent(in) :: A,B
    type (interval) :: C
    type (interval) :: D
    real :: M1,M2
function sadd (A,B) result (C)
    type (interval), intent(in) :: A,B
    type (interval) :: C
function smt (A) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
function ssub (A,B) result (C)
    type (interval), intent(in) :: A,B
    type (interval) :: C
function sneg (A) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
function smult (A,B) result (C)
    type (interval), intent(in) :: A,B
    type (interval) :: C
    real :: M1,M2,M3,M4
function sdiv (A,B) result (C)
    type (interval), intent(in) :: A,B
    type (interval) :: C
    type (interval) :: D
    real :: M1,M2,M3,M4
function sinv (A) result (C)
    type (interval) :: A
    type (interval) :: C
function sabs(A) result (C)
    type (interval), intent (in) :: A
    type (interval) :: C
function ssqr (A) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
    real :: X
function ssqrt (A) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
function spotn (A,n) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
    integer :: n
    logical :: ck_int
function sexp (A) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
function sln (A) result (C)
    type (interval), intent(in) :: A

```



```

    type (interval) :: C
function slog (A) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
function ssin (A) result (C)
    type (interval), intent(in) :: A
    type (interval) :: C
    integer :: q1,q2
    real :: PI, i1,i2
function quadrante(X) result (Y)
    real :: X
    integer :: Y
function scos (A) result (C)
    type (interval) :: A,X
    type (interval) :: C
    real :: PI
function stan (A) result (C)
    type (interval) :: A,X,Y
    type (interval) :: C
function sarctan (A) result (C)
    type (interval) :: A
    type (interval) :: C
    type (interval), intent(in) :: intv1
    type (interval), intent(in), optional :: intv2, intv3, intv4, intv5
    type (interval),intent(in) :: Entrada
function infla_down (A) result (C)
    real:: A,C
function infla_up (A) result (C)
    real:: A,C
function sblow(A,B) result (C)
    type (interval), intent (in) :: A
    real, intent (in) :: B
    type (interval) :: C
subroutine sread(A,B)
    type (interval), intent (out) :: B
    integer, intent (in) :: A
subroutine swrite(A,B)
    type (interval), intent (in) :: B
    integer, intent (in) :: A
interface null
interface snull
interface smnull
interface svum
interface smid
interface operator (+)
interface operator (-)
interface operator (*)
interface operator (/)
interface operator (==)
interface operator (/=)
interface operator (>)
interface operator (<)
interface operator (>=)
interface operator (<=)

```

```

interface sin
interface cos
interface tan
interface arctan
interface exp
interface ln
interface log
interface sqr
interface sqrt
subroutine smcopy(A,B)
    type(interval), dimension(:,:), intent (out) :: A
    type (interval), dimension(:,:), intent (in) :: B
subroutine smswap(A,B)
    type(interval), dimension(:,:), intent (inout) :: A,B
    real :: AUX1
function smintval(A,B) result (C)
    real, dimension(:,:), intent (in) :: A,B
    type (interval), dimension(size(A,1),size(A,2)) :: C
function smintpt(A) result (C)
    real, dimension(:,:), intent (in) :: A
    type (interval), dimension(size(A,1),size(A,2)) :: C
function sminf(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smsup(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smmod(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smmed(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smmin(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smmax(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smdia(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smdiar(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smraio(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(size(A,1),size(A,2)) :: C
function smdis(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    real, dimension(size(A,1),size(A,2)) :: C
function smequ(A,B) result (C)
    type (interval), dimension(:,:), intent (in) :: A,B
    logical :: C

```

```

function smless(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smgre(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smleq(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smgeq(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smneq(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function rminsm (A,B) result (C)
    real, dimension (:,:), intent (in) :: A
    type (interval), dimension (:,:), intent (in) :: B
    logical :: C
function smxiny(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smxcy(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smxmy(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smiv(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B
    logical :: C
function smuni (A,B) result (C)
    type (interval) , dimension (:,:), intent (in) :: A,B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smunid (A,B) result (C)
    type (interval) , dimension (:,:), intent (in) :: A,B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function sminter (A,B) result (C)
    type (interval) , dimension (:,:), intent (in) :: A,B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smadd (A,B) result (C)
    type (interval), dimension (:,:), intent(in) :: A,B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smsub (A,B) result (C)
    type (interval), dimension (:,:), intent(in) :: A,B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smmt(A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1),size(A,2)) :: C
function smneg(A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1),size(A,2)) :: C
function smmult(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A,B

```

```

    type (interval), dimension(size(A,1),size(B,2)) :: C
function smabs (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smsqr (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smsqrt (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smexp (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smln (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smlog (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smpotn (A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    integer :: B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smarctan (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smtan (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smsin (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smcos (A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smtransp(A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,2), size(A,1)) :: C
function smnullm(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    type (interval), dimension(size(A,1), size(A,2)) :: C
function smnulld(A,B) result (C)
    integer, intent (in) :: A,B
    type (interval), dimension(A,B) :: C
function smidm(A) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    type (interval), dimension(size(A,1), size(A,2)) :: C
function smidd(A) result (C)
    integer, intent (in) :: A
    type (interval), dimension(A,A) :: C
function smbrow(A,B) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension (size(A,1),size(A,2)) :: C

```

```

subroutine smread(A,B)
  integer, intent (in) :: A
  type (interval), dimension(:,,:), intent (out) :: B
subroutine smwrite(A,B)
  integer, intent (in) :: A
  type (interval), dimension(:,,:), intent (in) :: B
subroutine svcopy(A,B)
  type (interval), dimension (:), intent(out) :: A
  type (interval), dimension (:), intent (in) :: B
subroutine svswap(A,B)
  type (interval), dimension (:), intent (inout) :: A,B
  real :: AUX1
function svintval(A,B) result (C)
  real, dimension(:), intent (in) :: A,B
  type (interval), dimension (size(A)) :: C
function svinf(A) result (C)
  type (interval), dimension(:), intent (in) :: A
  real,dimension (size(A)) :: C
function svsup(A) result (C)
  type (interval), dimension(:), intent (in) :: A
  real, dimension(size(A)) :: C
function svintpt(A) result (C)
  real, dimension(:), intent (in) :: A
  type (interval),dimension(size(A)) :: C
function svmod(A) result (C)
  type (interval), dimension (:), intent (in) :: A
  real, dimension(size(A)) :: C
function svmax(A) result (C)
  type (interval), dimension (:), intent (in) :: A
  real, dimension(size(A)) :: C
function svmin(A) result (C)
  type (interval), dimension (:), intent (in) :: A
  real, dimension(size(A)) :: C
function svdis(A,B) result (C)
  type (interval), dimension(:), intent (in) :: A,B
  real ,dimension(size(A)) :: C
function svdia(A) result (C)
  type (interval), dimension(:), intent (in) :: A
  real, dimension(size(A)) :: C
function svraio(A) result (C)
  type (interval), dimension(:), intent (in) :: A
  real, dimension(size(A)) :: C
function svdiar(A) result (C)
  type (interval), dimension(:), intent (in) :: A
  real, dimension(size(A)) :: C
function svmed(A) result (C)
  type (interval), dimension(:), intent (in) :: A
  real ,dimension(size(A)) :: C
function svequ(A,B) result (C)
  type (interval), dimension(:), intent (in) :: A,B
  logical :: C
function svless(A,B) result (C)
  type (interval), dimension(:), intent (in) :: A,B
  logical :: C

```



```

function svgre(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    logical :: C
function svleq(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    logical :: C
function svgeq(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    logical :: C
function rvinsv(A,B) result (C)
    real, dimension(:), intent (in) :: A
    type (interval), dimension(:), intent (in) :: B
    logical :: C
function svneq(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    logical :: C
function svxiny(A,B) result (C)
    type (interval), dimension(:), intent(in) :: A,B
    logical :: C
function svxcy(A,B) result (C)
    type (interval), dimension(:), intent(in) :: A,B
    logical :: C
function svxmy(A,B) result (C)
    type (interval), dimension(:), intent(in) :: A,B
    logical :: C
function sviv(A,B) result (C)
    type (interval), dimension(:), intent(in) :: A,B
    logical :: C
function svuni(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    type (interval), dimension(size(A)) :: C
function svinter(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    type (interval), dimension(size(A)) :: C
function svunid(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    type (interval), dimension(size(A)) :: C
function svadd(A,B) result (C)
    type (interval), dimension (:), intent (in) :: A,B
    type (interval),dimension (size(A)) :: C
function svmt(A) result (C)
    type (interval), dimension(:), intent (in) :: A
    type (interval), dimension(size(A)) :: C
function svsub(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    type (interval), dimension(size(A)) :: C
function svneg(A) result (C)
    type (interval), dimension(:), intent (in) :: A
    type (interval), dimension(size(A)) :: C
function svmult(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    type (interval) :: C
function svabs(A) result (C)
    type (interval), dimension (:), intent (in) :: A

```

```

    type (interval), dimension (size(A)) :: C
function svsqrt(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svsqrt(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svexp(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svpotn(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A
    integer, intent (in) :: B
    type (interval), dimension(size(A)) :: C
function svln(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svlog(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svsin(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svcos(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svtan(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svarctan(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svnorm(A) result (C)
    type (interval), dimension(:), intent (in) :: A
    type (interval) :: C, AUX1
function svblow(A,B) result (C)
    type (interval), dimension (:), intent(in) :: A
    real, intent (in) :: B
    type (interval), dimension(size(A,1)) :: C
function svnullv(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svnulld(A) result (C)
    integer, intent (in) :: A
    type (interval), dimension(A) :: C
function svumv(A) result (C)
    type (interval), dimension (:), intent (in) :: A
    type (interval), dimension (size(A)) :: C
function svumd(A) result (C)
    integer, intent (in) :: A
    type (interval), dimension(A) :: C
subroutine svread(A,B)
    type (interval), dimension(:), intent (out) :: B
    integer, intent(in) :: A

```

```

subroutine svwrite(A,B)
  type (interval), dimension(:), intent (in) :: B
  integer, intent (in) :: A
function saddr(A,B) result (C)
  type (interval), intent (in) :: A
  real, intent (in) :: B
  type (interval) :: C
function radds(A,B) result (C)
  real, intent (in) :: A
  type (interval), intent (in) :: B
  type (interval) :: C
function ssubr(A,B) result (C)
  type (interval), intent (in) :: A
  real, intent (in) :: B
  type (interval) :: C
function rsubs(A,B) result (C)
  real, intent (in) :: A
  type (interval), intent (in) :: B
  type (interval) :: C
function rmults(A,B) result (C)
  real, intent (in) :: A
  type (interval), intent (in) :: B
  type (interval) :: C
function smultr(A,B) result (C)
  type (interval), intent (in) :: A
  real, intent(in) :: B
  type (interval) :: C
function rdivs(A,B) result (C)
  real, intent (in) :: A
  type (interval), intent (in) :: B
  type (interval) :: C
function sdivr(A,B) result (C)
  type (interval), intent (in) :: B
  type (interval) :: C
function runids(A,B) result (C)
  real, intent (in) :: A
  type (interval), intent (in) :: B
  type (interval) :: C
function sunidr(A,B) result (C)
  type (interval), intent (in) :: A
  real, intent (in) :: B
  type (interval) :: C
function raddsv(A,B) result (C)
  real, intent (in) :: A
  type (interval), dimension (:), intent (in) :: B
  type (interval), dimension (size(B)) :: C
  type (interval) :: AUX1
function svaddr(A,B) result (C)
  type (interval), dimension (:), intent (in) :: A
  real, intent (in) :: B
  type (interval), dimension (size(A)) :: C
  type (interval) :: AUX1
function rsubsv(A,B) result (C)

```

```

    real, intent (in) :: A
    type (interval), dimension (:), intent (in) :: B
    type (interval), dimension (size(B)) :: C
    type (interval) :: AUX1
function svsubr(A,B) result (C)
    type (interval), dimension (:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension (size(A)) :: C
    type (interval) :: AUX1
function rmultsv(A,B) result (C)
    real, intent (in) :: A
    type (interval), dimension (:), intent (in) :: B
    type (interval), dimension (size(B)) :: C
    type (interval) :: AUX1
function svmult(A,B) result (C)
    type (interval), dimension (:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension (size(A)) :: C
    type (interval) :: AUX1
function svdivr(A,B) result (C)
    type (interval), dimension (:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension (size(A)) :: C
    type (interval) :: AUX1
function raddsm(A,B) result (C)
    real, intent (in) :: A
    type (interval), dimension(:,:), intent (in) :: B
    type (interval), dimension(size(B,1), size(B,2)) :: C
    type (interval) :: AUX1
function smaddr(A,B) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension(size(A,1), size(A,2)) :: C
    type (interval) :: AUX1
function rsubsm(A,B) result (C)
    real, intent (in) :: A
    type (interval), dimension(:,:), intent (in) :: B
    type (interval), dimension(size(B,1), size(B,2)) :: C
    type (interval) :: AUX1
function smsubr(A,B) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension(size(A,1), size(A,2)) :: C
    type (interval) :: AUX1
function rmultsm(A,B) result (C)
    real, intent (in) :: A
    type (interval), dimension(:,:), intent (in) :: B
    type (interval), dimension(size(B,1), size(B,2)) :: C
    type (interval) :: AUX1
function smmult(A,B) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension(size(A,1), size(A,2)) :: C
    type (interval) :: AUX1

```

```

function smdivr(A,B) result (C)
    type (interval), dimension(:,,:), intent (in) :: A
    real, intent (in) :: B
    type (interval), dimension(size(A,1), size(A,2)) :: C
    type (interval) :: AUX1
function ssubsv(A,B) result (C)
    type (interval), intent (in) :: A
    type (interval), dimension(:,), intent (in) :: B
    type (interval), dimension(size(B)) :: C
function svsubs(A,B) result (C)
    type (interval), intent (in) :: B
    type (interval), dimension(:,), intent (in) :: A
    type (interval), dimension(size(A)) :: C
function svmults(A,B) result (C)
    type (interval), intent (in) :: B
    type (interval), dimension(:,), intent (in) :: A
    type (interval), dimension(size(A)) :: C
function smultsv(A,B) result (C)
    type (interval), intent (in) :: A
    type (interval), dimension(:,), intent (in) :: B
    type (interval), dimension(size(B)) :: C
function saddsv(A,B) result (C)
    type (interval), intent (in) :: A
    type (interval), dimension(:,), intent (in) :: B
    type (interval), dimension(size(B)) :: C
function svadds(A,B) result (C)
    type (interval), intent (in) :: B
    type (interval), dimension(:,), intent (in) :: A
    type (interval), dimension(size(A)) :: C
function svdivs(A,B) result (C)
    type (interval), dimension (:),intent (in) :: A
    type (interval), intent(in) :: B
    type (interval), dimension(size(A)) :: C
function smadds (A,B) result (C)
    type (interval), dimension (:,:), intent(in) :: A
    type (interval), intent (in) :: B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function saddsm (A,B) result (C)
    type (interval), intent (in) :: A
    type (interval), dimension (:,:), intent(in) :: B
    type (interval), dimension (size(B,1), size(B,2)) :: C
function smsubs (A,B) result (C)
    type (interval), dimension (:,:), intent(in) :: A
    type (interval), intent (in) :: B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function ssubsm (A,B) result (C)
    type (interval), dimension (:,:), intent(in) :: B
    type (interval), intent (in) :: A
    type (interval), dimension (size(B,1), size(B,2)) :: C
function smmults (A,B) result (C)
    type (interval), dimension (:,:), intent(in) :: A
    type (interval), intent (in) :: B
    type (interval), dimension (size(A,1), size(A,2)) :: C
function smultsm (A,B) result (C)

```



```

type (interval), dimension (:,:), intent(in) :: B
type (interval), intent (in) :: A
type (interval), dimension (size(B,1), size(B,2)) :: C
function smdivs (A,B) result (C)
type (interval), dimension (:,:), intent(in) :: A
type (interval), intent (in) :: B
type (interval), dimension (size(A,1), size(A,2)) :: C
function smmultsv(A,B) result (C)
type (interval), dimension(:,:), intent (in) :: A
type (interval), dimension(:), intent (in) :: B
type (interval), dimension(size(A,1)) :: C
function svmultsm(A,B) result (C)
type (interval), dimension(:), intent (in) :: A
type (interval), dimension(:,:), intent (in) :: B
type (interval), dimension(size(B,2)) :: C
function rvaddsv(A,B) result (C)
real, intent (in), dimension(:) :: A
type (interval), intent (in), dimension (:) :: B
type (interval), dimension (size(A)) :: C
function svaddrv(A,B) result (C)
real, intent (in), dimension(:) :: B
type (interval), intent (in), dimension (:) :: A
type (interval), dimension (size(A)) :: C
function svsubrv(A,B) result (C)
type (interval), dimension(:), intent (in) :: A
real, dimension (:), intent (in) :: B
type (interval), dimension(size(A)) :: C
function rvmultsv(A,B) result (C)
type (interval), dimension(:), intent (in) :: B
real, dimension (:), intent (in) :: A
type (interval) :: C
function svmultrv(A,B) result (C)
type (interval), dimension(:), intent (in) :: A
real, dimension (:), intent (in) :: B
type (interval) :: C
function smadddrn (A,B) result (C)
type (interval), dimension (:,:), intent(in) :: A
real, dimension (:,:), intent (in) :: B
type (interval), dimension (size(A,1), size(A,2)) :: C
function rmaddsm (A,B) result (C)
type (interval), dimension (:,:), intent(in) :: B
real, dimension (:,:), intent (in) :: A
type (interval), dimension (size(A,1), size(A,2)) :: C
function smsubrn (A,B) result (C)
type (interval), dimension (:,:), intent(in) :: A
real, dimension (:,:), intent (in) :: B
type (interval), dimension (size(A,1), size(A,2)) :: C
function rmsubsm (A,B) result (C)
type (interval), dimension (:,:), intent(in) :: B
real, dimension (:,:), intent (in) :: A
type (interval), dimension (size(A,1), size(A,2)) :: C
function smmultrn(A,B) result (C)
type (interval), dimension (:,:), intent (in) :: A
real, dimension (:,:), intent (in) :: B

```

```

    type (interval), dimension(size(A,1),size(B,2)) :: C
function rmmultsm(A,B) result (C)
    type (interval), dimension (:,:), intent (in) :: B
    real, dimension (:,:), intent (in) :: A
    type (interval), dimension(size(A,1),size(B,2)) :: C
function smmultrv(A,B) result (C)
    type (interval), dimension(:,:), intent (in) :: A
    real, dimension(:), intent (in) :: B
    type (interval), dimension(size(A,1)) :: C
function rvmultsm(A,B) result (C)
    real, dimension(:), intent (in) :: A
    type (interval), dimension(:,:), intent (in) :: B
    type (interval), dimension(size(B,2)) :: C
function rmmultsv(A,B) result (C)
    type (interval), dimension (:), intent (in) :: B
    real, dimension(:,:), intent(in):: A
    type (interval), dimension(size(A,1)) :: C
function svmultrm(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A
    real, dimension(:,:), intent (in) :: B
    type (interval), dimension(size(B,2)) :: C
interface assignment (=)
interface operator (+)
interface operator (-)
interface operator (*)
interface operator (/)
interface operator (==)
interface operator (/=)
interface operator (>)
interface operator (<)
interface operator (<=)
interface operator (>=)
interface scompl
interface scintval
interface sin
interface cos
interface tan
interface exp
subroutine sccopy(A,B)
    type (scomplex), intent(out) :: A
    type (scomplex), intent (in) :: B
function screinf(A) result (C)
    type (scomplex), intent (in) :: A
    real :: C
function scresup(A) result (C)
    type (scomplex), intent (in) :: A
    real :: C
function sciminf(A) result (C)
    type (scomplex), intent (in) :: A
    real :: C
function scimsup(A) result (C)
    type (scomplex), intent (in) :: A
    real :: C
function scomiia(A,B) result (C)

```

```

    type (interval), intent(in) :: A,B
    type (scomplex) :: C
function scomri (A,B) result (C)
    real, intent (in) :: A
    type (interval), intent (in) :: B
    type (scomplex) :: C
function scomir(A,B) result (C)
    type (interval), intent (in) :: A
    real, intent (in) :: B
    type (scomplex) :: C
function scomi(A) result (C)
    type (interval), intent (in) :: A
    type (scomplex) :: C
function svalcc(A,B) result (C)
    complex, intent (in) :: A,B
    type(scomplex) :: C
function svalrc(A,B) result (C)
    real, intent (in) :: A
    complex, intent (in) :: B
    type (scomplex) :: C
function svalcr(A,B) result (C)
    complex, intent (in) :: A
    real, intent (in) :: B
    type (scomplex) :: C
function svalc(A) result (C)
    complex, intent (in) :: A
    type (scomplex) :: C
function scinf(A) result (C)
    type (scomplex), intent (in) :: A
    complex :: C
function scsup(A) result (C)
    type (scomplex), intent (in) :: A
    complex :: C
function scre(A) result (C)
    type (scomplex), intent(in) :: A
    type (interval) :: C
function scim(A) result (C)
    type (scomplex),intent(in) :: A
    type (interval) :: C
subroutine scswap(A,B)
    type (scomplex), intent (inout) :: A,B
    type (interval) :: C
function scmed(A) result (C)
    type (scomplex), intent (in) :: A
    complex :: C
function scabs(A) result (C)
    type (scomplex), intent (in) :: A
    type (interval) :: C
function sconj(A) result (C)
    type (scomplex), intent (in) :: A
    type (scomplex) :: C
function scraio(A) result (C)
    type (scomplex), intent (in) :: A
    type (interval) :: C

```

```

function scdia(A) result (C)
  type (scomplex), intent (in) :: A
  complex :: C
function scadd(A,B) result (C)
  type (scomplex), intent (in) :: A,B
  type (scomplex) :: C
function scmul(A) result (C)
  type (scomplex), intent (in) :: A
  type (scomplex) :: C
function scsub(A,B) result (C)
  type (scomplex), intent (in) :: A,B
  type (scomplex) :: C
function sceneg(A) result (C)
  type (scomplex), intent (in) :: A
  type (scomplex) :: C
function scmult(A,B) result (C)
  type (scomplex), intent (in) :: A,B
  type (scomplex) :: C
function scdiv(A,B) result (C)
  type (scomplex), intent (in) :: A,B
  type (scomplex) :: C
function scinv(A) result (C)
  type (scomplex), intent (in) :: A
  type (scomplex) :: C
function scequ(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function seless(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function segre(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function sceq(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function reins(A,B) result (C)
  complex, intent (in) :: A
  type (scomplex), intent (in) :: B
  logical :: C
function seleq(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function scxiny(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function scxcy(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function scxmy(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function sciv(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function sceneg(A,B) result (C)
  type (scomplex), intent (in) :: A,B
function scmul(A,B) result (C)
  type (scomplex), intent (in) :: A,B
  type (scomplex) :: C
function scmulid(A,B) result (C)
  type (scomplex), intent (in) :: A,B
  type (scomplex) :: C

```

```

function scinter(A,B) result (C)
    type (scomplex), intent (in) :: A,B
    type (scomplex) :: C
function scblow(A,B) result (C)
    type (scomplex), intent (in) :: A
    real, intent (in) :: B
    type (scomplex) :: C
function scang(A) result (C)
    type (scomplex), intent (in) :: A
    type (interval) :: C
    real :: AUX1, AUX2
    real, intent (in) :: A,B
    real :: C, pi
function scsqr(A) result (C)
    type (scomplex), intent (in) :: A
    type (scomplex) :: C
function scexp(A) result (C)
    type (scomplex), intent (in) :: A
    type (scomplex) :: C
function l_times(A) result (C)
    type (scomplex), intent(in) :: A
    type (scomplex) :: C
function scos(A) result (C)
    type (scomplex), intent (in) :: A
    type (scomplex) :: C,AUX1
function scsin(A) result (C)
    type (scomplex), intent (in) :: A
    type (scomplex) :: C, AUX1
function sctan(A) result (C)
    type (scomplex), intent (in) :: A
    type (scomplex) :: C
subroutine scread(A,B)
    type (scomplex), intent (out) :: B
    integer , intent (in) :: A
subroutine scwrite(A,B)
    type (scomplex), intent (in) :: B
    integer , intent(in) :: A
function sv saxpy(R,X,Y) result (C)
    real, intent (in) :: R
    type (interval),dimension(:), intent (in) :: X,Y
    type (interval),dimension(size(X)) :: C
function svasum(A) result (C)
    type (interval), dimension(:), intent (in) :: A
    type (interval) :: C
function svddot(A,B) result (C)
    type (interval), dimension(:) :: A,B
    type (interval) :: C
    double precision:: AUXinf, AUXsup
    double precision, dimension(size(A)) :: Dinf, Dsup,Einf,Esup
function svdot(A,B) result (C)
    type (interval), dimension(:), intent (in) :: A,B
    type (interval) :: C
    real, dimension(size(A)) :: X,Y,W,Z
    real :: S,T,U,V

```



```

integer :: i
function svsum(A) result (C)
  type (interval), dimension (:), intent (in) :: A
  type (interval) :: C
function sgemv(R,S,A,X,Y) result (C)
  real, intent (in) :: R,S
  real, dimension (:,:), intent (in) :: A
  type (interval), dimension (:), intent (in) :: X,Y
  type (interval), dimension (size(Y)) :: C
function sgemm(R,S,A,B,C) result (D)
  real, intent (in) :: R,S
  type (interval), dimension (:,:), intent (in) :: A,B,C
  type (interval), dimension (size(A,1), size(A,2)) :: D
function svnrn2(A) result (C)
  type (interval), dimension (:), intent (in) :: A
  type (interval) :: AUX1
  real :: C
function rrrml(A) result (C)
  real, dimension (:,:), intent (in) :: A
  real, dimension (size(A,1)) :: AUX1
  real :: C
function rrrnml(A) result (C)
  type (interval), dimension (:,:), intent(in) :: A
  type (interval), dimension (size(A,1)) :: AUX1
  real :: C
function smrrml(A) result (C)
  type (interval), dimension (:,:), intent(in) :: A
  type (interval), dimension (size(A,1)) :: AUX1
  type (interval) :: C
function rrrmc(A) result (C)
  real, dimension (:,:), intent (in) :: A
  real, dimension (size(A,2)) :: AUX1
  real :: C
function rrrnrc(A) result (C)
  type (interval), dimension (:,:), intent(in) :: A
  type (interval), dimension (size(A,2)) :: AUX1
  real :: C
function smrrnc(A) result (C)
  type (interval), dimension (:,:), intent(in) :: A
  type (interval), dimension (size(A,2)) :: AUX1
  type (interval) :: C
function rrrm2(A) result (R)
  real, dimension (:,:), intent (in) :: A
  real :: R, AUX1
function rrrnrm2(A) result (R)
  type (interval), dimension (:,:), intent (in) :: A
  real :: R
  type (interval) :: AUX1
function smrrm2(A) result (R)
  type (interval), dimension (:,:), intent (in) :: A
  type (interval) :: R, AUX1
function rvasum(A) result (C)
  real, dimension (:), intent (in) :: A
  real :: C

```

```

function rvsum(A) result (C)
    real, dimension (:), intent (in) :: A
    real :: C
function rdet(A) result (C)
    real, dimension (:,:), intent (in) :: A
    real, dimension (size(A,1),size(A,2)) :: B
    real :: C
function prodiag(A) result (C)
    real, dimension (:,:), intent (in) :: A
    real :: C
subroutine triang(A)
    real, dimension (:,:), intent (inout) :: A
function retro(M,Y) result (X)
    real, dimension (:,:), intent (in) :: M
    real, dimension (:), intent (in) :: Y
    real, dimension (size(M,1), size(M,1)+1) :: AUX1
    real, dimension (size(Y,1)) :: X
    real :: pivo
function smhilbn(N) result (H)
    integer, intent (in) :: N
    type (interval), dimension (N,N) :: H
function rminv(A) result (C)
    real, dimension (:), intent (in) :: A
    real, dimension (:) :: C
function sminv(A) result (C)
    type (interval), dimension (:,:), intent (in) :: A
    type (interval), dimension (size(A,1),size(A,2)) :: E,S,P,C
    real, dimension (size(A,1),size(A,2)) :: B, Ident,Z, AC
    real :: r,BI
    integer, dimension (size(A,1)) :: ipiv
function sdgemm(r,s,A,B,C) result (D)
    real, intent (in) :: r,s
    type (interval), dimension (:,:), intent (in) :: A,B,C
    type (interval), dimension (size(A,1), size(A,2)) :: D
function sdgemv(r,s,A,x,y) result (w)
    real, intent (in) :: real
    real, dimension (:,:), intent (in) :: A
    type (interval), dimension (:), intent (in) :: x,y
    type (interval), dimension (size(Y)) :: w
function smdet(M) result (r)
    type (interval), dimension (:,:), intent (in) :: M
    real :: r
function rconda(M) result (r)
    real, dimension (:,:), intent (in) :: M
    real :: r
function smdeth(n) result (r)
    integer, intent (in) :: n
    real :: r

```



*Uso Efetivo da Matemática Intervalar em Supercomputadores Vetoriais.*

por

Tiarajú Asmuz Diverio

Defesa de Tese apresentada aos Senhores:

Prof. Dr. Jairo Panetta (IEAV-CTA)

Profa. Dra. Marcia de Barros Correia (UFPe)

Prof. Dr. Raul Fernando Weber

Prof. Dr. Rudnei Dias da Cunha (DMPA/UFRGS)

Vista e permitida a impressão.

Porto Alegre, 28/06/95.

Prof. Dr. Dalcídio Moraes Claudio,  
Orientador.

Prof. Dr. Philippe Olivier Alexandre Navaux,  
Co-orientador.

**Prof. José Palazzo Moreira de Oliveira**  
Coordenador do Curso de Pós-Graduação  
em Ciência da Computação - CPGCC  
Instituto de Informática - UFRGS