UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

Algoritmos Paralelos Iterativos do tipo quasi-Newton para a Minimização de Funções Multivariadas

por

G. Amado Méndez Cruz
- Gilberto

Dissertação submetida como requisito parcial
para a obtenção do grau de
Mestre em Matemática Aplicada

Prof. Dr. Rudnei Dias da Cunha

Orientador

Porto Alegre, Fevereiro de 1997.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Méndez Cruz, G. Amado

Algoritmos Paralelos Iterativos do tipo quasi-Newton para a Minimização de Funções Multivariadas / G. Amado Méndez Cruz. – Porto Alegre: CPGMAp da UFRGS, 1997.

86 p.: il.

Dissertação (Mestrado)—Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Matemática Aplicada, Porto Alegre, 1997. da Cunha, Dias Rudnei, Orient.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Profa. Wrana Panizzi
Pró-Reitor de Pesquisa : Prof. Pedro Cézar Dutra Fonseca
Pró-Reitor de Pós-Graduação : Prof. José Carlos Ferraz Hennemann
Diretor do Instituto de Matemática: Prof. Aron Teitelbaum
Coordenadora do CPGMAp: Profa. Maria Cristina Varriale
Bibliotecário-Chefe do Instituto de Matemática: Carlos Brandão Schwab

" Se nos encantamos com um sonho
nossa esperança cresce
e nunca mais seremos fazedores
de tarefas "

A meus queridos filhos Paola e Ricardo,
eles foram sempre a minha motivação.

AGRADECIMENTOS

- Ao longo do processo de levantamento de informação, pesquisa e execução das idéias que nortearam este trabalho muitas amizades foram criadas e muitos contatos foram feitos. A estas pessoas meu muito obrigado pela colaboração.

- Ao corpo docente de Pós-Graduação em Matemática Aplicada da UFRGS, em especial ao meu orientador Dr. Rudnei Dias da Cunha, pelo apoio e pelas sugestões que muito me auxiliaram na elaboração deste trabalho.

- Ao Conselho Nacional de Pesquisa e Desenvolvimento Tecnológico da República Federativa do Brasil, pela concessão da bolsa de estudos.

- A meu centro de estudos e trabalho, a Universidade Nacional da Liberdade - Trujillo, Perú pela oportunidade para fazer estes estudos de Pós-Graduação na Universidade Federal do Rio Grande do Sul, Brasil.

- Finalmente, quero agradeçer a minha esposa *Marleny*, quem sempre me apoiou nesta experiência e suportou minha ausência do lar durante todo este período.

SUMÁRIO

LISTA DE FIGURAS

GLOSSÁRIO

$\{x_k\}_{k\geq 0}$	uma sequência de iteracões.
$(x_k)_i$	i-componente de um vetor x_k.
$\|\cdot\|$	uma norma sobre R^n.
$\|\cdot\|_p$	l^p-norma sobre R^n, $\|\cdot\|_p = (\sum_{j=1}^{n} \mid (x)_i \mid^p)^{1/p}$.
$\kappa_p(A)$	número de Condição de A relativo à norma l^p.
$\sigma(A)$	conjunto de autovalores de A.
$\|u\|_A$	A-norma do vetor u, $\|u\|_A = \sqrt{x^T Ax}$.
$\mathcal{K}_k$	subespaço de Krylov.
$F'(x)$	derivada de primeira ordem da função F, $F'(x) = \partial F(x)/\partial x$.
$J(x)$	matriz Jacobiana da função F.
$\mathcal{B}(\delta)$	uma bola ao redor de x^* de raio δ.
s	diferença entre dois pontos de iteração, $s = x^{k+1} - x^k$.
e_k	erro da aproximação na iteração k, $e_k = x_k - x^*$.
$\frac{\|F(x)\|}{\|F(x_0)\|}$	resíduo relativo não-linear.
$\|\cdot\|_*$	norma ponderada de $\cdot$, $\|\cdot\|_* = \|F'(x^*)\cdot\|$.
η_k	k-termo força.
$\mathcal{O}_{QR}$	número das oper. da fatorização de uma matriz superior de Hessenberg.
$\mathcal{O}_{RS}$	número de operações para resolver um sist. triangular superior de ordem k usando subst. por retrocesso, $\mathcal{O}_{RS} = k\mathcal{O}_{\div} + \frac{1}{2}(k^2 - k)\mathcal{O}_*$.

$\mathcal{O}_{u^T v}(n)$ número de operações em um produto interno sobre os vetores de tamanho n, $\mathcal{O}_{u^T v}(n) = n\mathcal{O}_* + (n-1)\mathcal{O}_+$.

$\mathcal{O}_{\|u\|_2}$ número de operações para calcular a norma do vetor u de dimensão n, $\mathcal{O}_{\|u\|_2} = n\mathcal{O}_* + (n-1)\mathcal{O}_+ + \mathcal{O}_{\sqrt{\cdot}}$.

$\mathcal{O}_{u+v}$ número de operações para calcular a suma de vetores de dimensão n, $\mathcal{O}_{u+v} = n\mathcal{O}_+$.

$\mathcal{O}_{u+\alpha v}$ número de operações para calcular $w = u + \alpha v$, $\mathcal{O}_{u+\alpha v} = n\mathcal{O}_+ + \mathcal{O}_*$.

$\mathcal{O}_{Au}$ número de operações para calcular o produto matriz-vetor Au, $\mathcal{O}_{Au} = n^2\mathcal{O}_* + (n^2 - n)$.

T_1 tempo para realizar uma tarefa em um único processador.

T_p tempo para realizar uma tarefa sobre p processadores.

S_p "*speed-up*", $S_p = T_1/T_p$

K número de iterações do algoritmo.

$K1$ número de iterações do GMRES.

$\mathcal{O}_F$ número de operações para uma avaliação da função.

$\mathcal{O}_{parab}$ número de operações na rotina *parab3p*.

$\mathcal{O}_{gold}$ número de operações na rotina *golden*.

$\mathcal{O}_{jac}$ número de operações na rotina *jacpa*.

$\mathcal{O}_{mv}$ número de operações na rotina matriz-vetor.

c base do algoritmo.

LISTA DE TABELAS

RESUMO

O objetivo deste trabalho é apresentar e descrever a teoria e implementação paralela em *PVM*, de dois algoritmos iterativos do tipo quasi-Newton - *Newton-GMRES* e *Broyden* - para a solução de equações não lineares $F = 0$, onde a função $F : R^n \rightarrow R^n$ é de classe C^1 e seu Jacobiano $J(x)$ é esparso. Uma ilustração e comparação destes métodos com suas versões sequenciais é obtida ao aplicá-los a dois problemas específicos.

ABSTRACT

The objective of this work is to introduce and describe the theory and implementation on *PVM*, of two quase-Newton iterative algorithms - *Newton-GMRES e Broyden* - for the resolution of nonlinear equations $F = 0$, where a function $F : R^n \to R^n$ is of class C^1 and its Jacobian $J(x)$ is sparse. An ilustration and comparison of these methods with their serial versions is obtained as they apply to two especific problems.

1 INTRODUÇÃO

Muitos fenômenos de interesse para os cientistas e engenheiros são caracterizados por modelos matemáticos na forma de equações diferenciais parciais simultâneas. Para resolver estes equações por técnicas de computação digital, o contínuo espaço/tempo é representado por pontos espaçados discretamente por diferentes aproximações, tais como diferenças finitas, elementos finitos ou volumes finitos, entre outras. Para alcançar precisões desejadas em muitas situações, os modelos de diferenças-finitas podem conter milhares ou milhões de pontos no domínio, onde uma equação algébrica deve ser resolvida em cada ponto para cada instante de tempo. Particularmente onde as equações diferenciais parciais não são lineares e onde os algoritmos implícitos devem ser usados para evitar instabilidade computacional, a solução destes sistemas de equações algébricas apresenta problemas verdadeiramente formidáveis. Ainda com os melhores supercomputadores disponíveis, o processo de computação pode necessitar horas ou dias de execução.

Nestes últimos anos tem sido costume falar dos "grandes desafios" da ciência - um comum denominador para a solução destes problemas é a necessidade de enorme poder de computação. Por exemplo, a figura (1.1) mostra os recursos de computação (em operações de ponto flutuante por segundo (*FLOPS*)) e os requerimentos de memória (em palavras de 64 bits) para algumas aplicações.

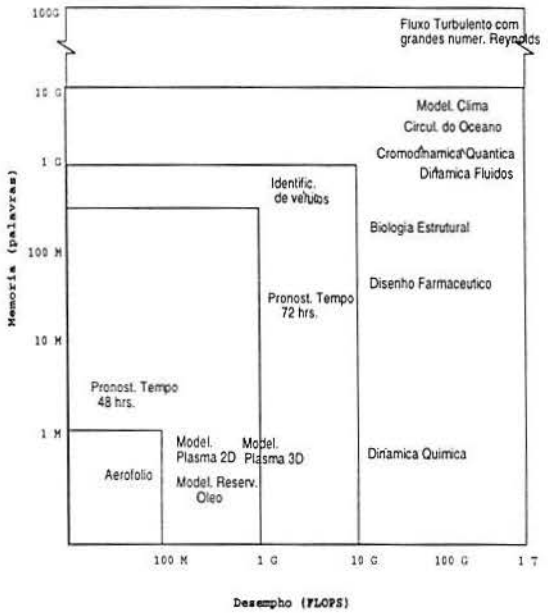


Figura 1.1 Os "Grandes desafios"

Uma grande desvantagem dos supercomputadores é que são sistemas computacionais de custo muito alto e centralizado, atendendo a muitos grupos de pesquisa. A crescente disponibilidade de alternativas relativamente de baixo custo, se comparadas aos supercomputadores, promete nos próximos anos mudanças significativas na maneira através da qual as simulações serão conduzidas. Por exemplo, o advento de mini-supercomputadores nos anos 80, permitiu ter um sistema descentralizado e com um custo muito menor. Também a idéia de aproveitar os recursos que geralmente se têm, originou a aparição de meios fisicos e lógicos para acoplar estes recursos, com a finalidade de obter altos desempenhos computacionais.

1.0.1 Apresentação do problema e alternativas de solução

A maioria das aplicações referidas anteriormente envolvem a solução de grandes sistemas de equações não lineares da forma:

$$F(x) = 0, \qquad F = (f_1, ..., f_n) \tag{1.1}$$

onde $F : R^n \rightarrow R^n$ é uma função não - linear de classe C^1, e sua matriz Jacobiana $J(x)$ é esparsa [Dennis 83] [Ortega 70]. Ditos sistemas de equações são usualmente grandes (milhares ou milhoẽs de equações) e esparsas, dependendo do modelo matemático do qual eles são derivados.

Os métodos iterativos são usualmente preferidos para resolver ditos sistemas dado que teoricamente pode-se mostrar que o número de *FLOPS* requerido para alcançar uma tolerância ϵ dada é substancialmente menor do que um método direto. Além disso, os métodos diretos são usualmente difíceis de paralelizar, uma vez que eles impõem precedência de operações e comunicação global entre os processadores, ver [Geist 87], p. 241; isto é particularmente o caso quando se utiliza máquinas de memória distribuída e ainda o uso de algoritmos mais refinados e estruturas de dados que não tem capacidade para brindar tantas implementações eficientes como aqueles obtidas para métodos iterativos. Outra dificuldade é o fato que os métodos diretos conduzem a "fill-in", destruindo a esparsidade que pode ser explorada em alguns problemas, de forma a aumentar o paralelismo.

Os métodos iterativos podem ser expressos em termos de um conjunto de operações de álgebra linear como acumulação de vetores, produtos-internos e produtos matriz-vetor. A paralelização de métodos iterativos pode ser assim obtida, pelas paralelizações eficientes destas operações de álgebra linear [da Cunha 92].

O melhor método para resolver este tipo de problema é o método de Newton. Este é um método iterativo, onde as aproximações sucessivas à solução de (1.1) são calculados segundo a seguinte fórmula:

$$x_{k+1} = x_k - J^{-1}(x_k)F(x_k) \tag{1.2}$$

Portanto, em cada iteração do método de Newton, as derivadas $\partial f_i/\partial x_j$ devem ser calculadas, e o sistema linear de ordem n

$$J(x_k)s = -F(x_k) \tag{1.3}$$

deve ser resolvido, para obter x_{k+1}.

Os métodos quasi-Newton foram introduzidos para tratar com situações onde as derivadas analíticas não estão disponíveis, ou são muito custosas para se calcular. Elas obedecem à formula

$$B_k s = -F(x_k), \qquad x_{k+1} = x_k + s. \tag{1.4}$$

Em cada iteração de um método quasi-Newton, somente os valores da função são calculados e o sistema linear dado acima é resolvido. A nova matriz B_{k+1} é obtida a partir de B_k usando relações de recorrência, as quais incluem x_k, x_{k+1}, $F(x_k)$ e $F(x_{k+1})$.

O melhor método quasi-Newton conhecido para pequenos problemas densos é o método de Broyden [Broyden 65], [Dennis 83]. A situação no caso de matrizes esparsas grandes é diferente. Com efeito, se B_k é uma matriz esparsa, geralmente B_{k+1} se torna uma matriz densa, a qual pode ter pouca relação com as matrizes Jacobianas verdadeiras. Esta situação motivou o desenvolvimento dos métodos quasi-Newton.

Os algoritmos desenvolvidos nesta dissertação são voltados para resolver sistemas de equações lineares e não lineares derivados de aproximações por diferenças finitas de equações diferenciais parciais específicas. Os algoritmos foram implementados usando *PVM* (Parallel Virtual Machine) sobre estações de trabalho SUN.

Um breve panorama dos conteúdos da dissertação segue abaixo. O capítulo 1 descreve a introdução da dissertação. O capítulo 2 discute o panorama do ambiente paralelo ressaltando aspectos do *PVM*. O capítulo 3 descreve resultados básicos e a notação correspondente. O capítulo 4 apresenta métodos iterativos para sistemas de equações nao simétricos, descrevendo o método dos resíduos generalizado (*GMRES*). O capítulo 5 apresenta métodos iterativos para a solução de equações não lineares. No capítulo 6 descrevemos a implementação paralela dos algoritmos sobre *PVM*. Finalmente, o capítulo 7 apresenta as conclusões e discussões sobre possíveis desenvolvimentos futuros.

2 UM PANORAMA DO AMBIENTE PARALELO

Neste capítulo, daremos um panorama do ambiente paralelo o qual usamos para desenvolver as implementações dos algoritmos.

2.1 PVM

O PVM (Parallel Virtual Machine)[PVM97] é um software que permite que uma rede de computadores Unix heterogêneos seja utilizada como um único computador; assim, grandes problemas computacionais podem ser resolvidos usando a potência combinada de muitos computadores. Sob PVM, uma coleção definida de computadores vetoriais, paralelos, seqüenciais aparecem como um grande computador com memória distribuída.

Neste trabalho, o termo *máquina virtual* será usado para designar este computador lógico com memória distribuída, e *"host"* será usado para designar cada um dos computadores membros.

O PVM fornece a função para automaticamente começar as *tarefas* sob a máquina virtual e permite que as tarefas se comuniquem e sincronizem com cada outra. Uma *tarefa* é definida como uma unidade de computação em PVM, análoga a um processo Unix (é frequentemente um processo Unix, mas não necessariamente). As aplicações, as quais podem ser escritas em Fortran ou C (ou ambas em uma mesma aplicação), podem ser paralelizadas usando *troca de mensagens*, mecanismo comum à maioria de computadores de memória distribuída. Enviando e recebendo mensagens, múltiplas tarefas de uma aplicação podem cooperar para resolver um problema em paralelo.

PVM suporta heterogeneidade a nível de aplicação, máquina e rede. Em outras palavras, PVM permite aplicar tarefas para explorar a arquitetura mais conveniente para sua solução . PVM trata todas as conversões dos dados que podem ser requeridas se dois computadores usam diferentes representações de números inteiros ou em ponto flutuante. Permite, ainda, que a máquina virtual seja interconectada por uma variedade de redes diferentes (incluindo Ethernet e FDDI).

O sistema PVM é composto de duas partes. A primeira parte é um *daemon*, chamado *pvmd3* e algumas vezes abreviado *pvmd*, que reside nos computadores que formam a máquina virtual. Multiplos usuários podem configurar inúmeras máquinas virtuais, e cada usuário pode executar várias aplicações PVM simultaneamente.

A segunda parte do sistema é uma biblioteca de rotinas de interface PVM (*libpvm3.a*). Esta biblioteca contém rotinas executáveis por um programa para efetuar troca de mensagens, criar processos, coordenar tarefas, e modificar a máquina virtual. Os programas de aplicações devem ser vinculados ("linked") com esta biblioteca para usar PVM.

Os programas de aplicação vêem PVM como um recurso de computação paralelo flexível e geral que suporta um modelo de computação de troca de mensagens. Este recurso poder ser acessado em três níveis diferentes: o modo *transparente* no qual as tarefas são automaticamente executadas sobre os host mais apropriados (geralmente os menos carregados), o modo *arquitetura dependente* no qual o usuário pode indicar arquiteturas específicas sobre as quais tarefas particulares podem ser executadas, e o modo *baixo-nível* no qual um host particular pode ser especificado. Ditas camadas permitem maior flexibilidade; entretanto, retêm a habilidade para explorar potências particulares de máquinas individuais sob a rede.

Os programas de aplicação sob PVM podem possuir controle arbitrário e dependência de estruturas. Em outras palavras, em qualquer ponto da execução de

uma aplicação concorrente, os processos em execução podem ter relações arbitrárias entre elas e, além disso, qualquer processo pode comunicar-se e/ou sincronizar-se com qualquer outro. Isto leva em conta a forma mais geral da computação paralela MIMD (*Multiple-Instructions Multiple-Data*), mas na prática as aplicações mais concorrentes são mais estruturadas. Duas estruturas típicas são o modelo *spmd* ("single-process multiple-data") no qual todos os processos são idênticos, porém cada um opera sobre dados diferentes e o modelo *mestre-escravo* no qual um conjunto de processos computacionais escravos desempenham trabalho para um ou mais processos mestres.

3 PRELIMINARES

Apresentamos algumas definições e resultados que serão usados para desenvolver os métodos e implementar os algoritmos.

3.1 Definições

Escreveremos as equações lineares como

$$Ax = b, \tag{3.1}$$

onde A é uma matriz não singular de ordem $n \times n$, b é dado, e

$$x^* = A^{-1}b \ \epsilon \ R^n$$

é a solução de (3.1).

Definição 3.1.1 *A norma matriz induzida de uma matriz A de ordem $n \times n$ é definida por*

$$\|A\| = \max_{\|x\|=1} \|Ax\|.$$

Lembra-se que a *número de condição* de A relativo à norma $\|\cdot\|$ é

$$\kappa(A) = \|A\|\|A^{-1}\|,$$

onde $\kappa(A)$ é infinito se A é singular.

A maioria dos métodos iterativos terminam quando o resíduo

$$r = b - Ax$$

é suficientemente pequeno. Um critério de terminação é

$$\frac{\|r_k\|}{\|r_0\|} < \tau, \tag{3.2}$$

o qual pode ser relacionado ao erro

$$e = x - x^*$$

em termos do número de condição .

Lema 3.1.1 *Seja $b, x, x_0 \in R^n$. Seja A não singular e seja $x^* = A^{-1}b$. Então ,*

$$\frac{\|e\|}{\|e_0\|} \leq \kappa(A)\frac{\|r\|}{\|r_0\|} \tag{3.3}$$

Ver a demostração em [Kelley 95].

O criterio de terminação (3.2) depende da iteração inicial e pode resultar em trabalho desnecessário quando a iteração inicial é boa e num pobre resultado quando a iteração inicial está longe da solução . Por esta razão, preferimos terminar a iteração quando

$$\frac{\|r_k\|}{\|b\|} < \tau \tag{3.4}$$

As duas condições (3.2) e (3.4) são iguais quando $x_0 = 0$, o qual é usualmente utilizado como estimativa inicial, particularmente quando a iteração linear esta sendo usada como parte de um resolvedor não linear.

A matriz A é dita *definida simétrica positiva(SPD)* se, além de ser simétrica, os autovalores de A são reais e positivos. Se A é *SPD* podemos definir a norma de um vetor u por

$$\|u\|_A = \sqrt{u^T A u}. \tag{3.5}$$

Dada uma matriz $A \in R^{n\times n}$ e um vetor z, o subespaço gerado pelos k vetores

$$z, Az, A^2z, ..., A^{k-1}z \tag{3.6}$$

para $k \geq 1$, é chamado um *Subespaço de Krylov* e é denotado por $\mathcal{K}(z, A, k)$ ou $\mathcal{K}_k$.

3.2 Convergência

Os métodos iterativos podem ser classificados pela *razão de convergência*

Definição 3.2.1 *Seja* $\{x_n\} \subset R^n$ *e* $x^* \in R^n$. *Então*

- $x_n \to x^*$ *q-quadraticamente se* $x_n \to x^*$ *e existe* $K > 0$ *tal que*

$$\|x_{n+1} - x^*\| \leq K\|x_n - x^*\|^2.$$

- $x_n \to x^*$ *q-superlinearmente com q-ordem* $\alpha > 1$ *se* $x_n \to x^*$ *e existe* $K > 0$ *tal que*

$$\|x_{n+1} - x^*\| \leq K\|x_n - x^*\|^\alpha.$$

- $x_n \to x^*$ *q-superlinearmente se*

$$\lim_{n\to\infty} \frac{\|x_{n+1} - x^*\|}{\|x_n - x^*\|} = 0.$$

- $x_n \to x^*$ *q-linearmente com q-fator* $\sigma \epsilon (0,1)$ *se*

$$\|x_{n+1} - x^*\| \leq \sigma \|x_n - x^*\|.$$

para n suficientemente grande.

- $x_n \to x^*$ *r-(quadraticamente, superlinearmente, linearmente) se existe uma sequência* $\{\xi_n\} \subset R$ *convergindo q-(quadraticamente, superlinearmente, linearmente) a zero tal que* $\|x_n - x^*\| \leq \xi_n$.
Dizemos que $\{x_n\}$ *converge r-superlinearmente com r-ordem* $\alpha > 1$ *se* $\xi_n \to 0$ *q-superlinearmente com q-ordem* α

Definição 3.2.2 *Um método iterativo para calcular* x^* *diz-se ser localmente (q-quadraticamente , q-superlinearmente, etc.) convergente se as iterações convergem a* x^* *(q-quadraticamente , q-superlinearmente, etc.) considerando que o dado inicial para a iteração é suficientemente bom.*

3.3 Hipóteses padrão

Seja

$$F(x) = 0, \qquad F = (f_1, ...f_n) \tag{3.7}$$

Se as componentes de F são diferenciáveis em $x \epsilon R^n$, definimos a *matriz Jacobiana* $J(x)$ por

$$J(x)_{ij} = \frac{\partial f_i}{\partial x_j}(x).$$

Estabelecemos as hipóteses padrão sobre F :

- A equação (3.7) tem uma solução x^*.
- $J : \Omega \to R^n$, $\Omega \epsilon R^n$ é Lipschiz contínua com constante de Lipschitz γ.
- $J(x^*)$ é não singular.

Estas hipóteses podem ser enfraquecidas [Keller 70] sem sacrificar as condições de convergência dos métodos que consideramos aqui; porém, o resultado clássico de convergência quadrática do método de Newton as requer.

3.4 Problemas

Nesta seção apresentamos os problemas que serão usados nos capítulos seguintes, como "objetos testes" para os algorítmos estudados.

3.4.1 A equação H de Chandrasekhar

A equação H de Chandrasekhar [Busbrigde 60],

$$F(H)(\mu) = H(\mu) - \left(1 - \frac{C}{2}\int_0^1 \frac{\mu H(\nu)d\mu}{\mu + \nu}\right)^{-1} = 0, \tag{3.8}$$

é usada para resolver problemas de distribuição de saída em transferência radioativa. Ela é uma equação não linear quadrática onde a incógnita é a função $H \epsilon C[0,1]$, e

C é um parâmetro. Fisicamente, valores razoáveis dos parâmetros são $0 < C \leq 1$. Discretizaremos a equação com a regra do ponto médio composto ("composite midpoint rule"). Aproximamos a integral presente em (3.8) por

$$\int_0^1 f(\mu)d\mu \approx \frac{1}{n}\sum_{j=1}^{n} f(\mu_j),$$

onde $\mu_i = (i - 1/2)/n$ para $1 \leq i \leq n$. O problema discreto resultante é

$$F(x)_i = x_i - \left(1 - \frac{C}{2n}\sum_{j=1}^{n}\frac{\mu_i x_j}{\mu_i + \mu_j}\right)^{-1}. \tag{3.9}$$

Devido a singularidade apresentada por H em $\mu = 0$, a solução do problema discreto não é ainda uma aproximação com precisão de primeira ordem do problema contínuo.

3.4.2 Equação de convecção - difusão

Consideramos a equação diferencial parcial

$$-\nabla^2 u + Cu(u_x + u_y) = f \tag{3.10}$$

com condições de contorno de Dirichlet homogêneas sobre o quadrado unitário $(0,1) \times (0,1)$. A função f é construída considerando a discretização da solução exata

$$10xy(1-x)(1-y)\exp^{x^{4.5}}.$$

A equação é discretizada usando uma malha de $l \times l$ pontos internos e o sistema de equações de ordem $n = l^2$ é derivado da discretização da equação por diferenças finitas de cinco pontos, onde $h = 1/(l+1)$. Os termos difusivos são dis-

cretizados usando diferenças centrais originando uma matriz simétrica e tridiagonal em blocos da forma

$$A = \begin{bmatrix} M & -I & & & \\ -I & M & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & M & -I \\ & & & -I & M \end{bmatrix}_{nxn}, \quad M = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix}_{lxl} \tag{3.11}$$

onde I é a matriz identidade de ordem l.

Os termos convectivos são discretizados usando diferenças frontais originando uma matriz banda superior não simétrica da forma

$$B = \begin{bmatrix} N & I & & & \\ & N & I & & \\ & & \ddots & \ddots & \\ & & & N & I \\ & & & & N \end{bmatrix}_{nxn}, \quad N = \begin{bmatrix} -2 & 1 & & & \\ & -2 & 1 & & \\ & & \ddots & \ddots & \\ & & & -2 & 1 \\ & & & & -2 \end{bmatrix}_{lxl} \tag{3.12}$$

onde I é a matriz identidade de ordem l.

4 MÉTODOS ITERAT. PARA SISTEMAS NÃO SIMÉTRICOS

Existem diferentes métodos para resolver sistemas não simétricos. Aqueles métodos os quais estão relacionados ao método dos Gradientes-Conjugados (*CG*) pertencem em geral a três classes, a saber *resolvedores de equações normais* (ex: CGNR [Hestenes 52] e CGNE [Craig 55]), *minimizadores de normas sobre um subespaço de Krylov* (ex: ORTHOMIN [Young 80], ORTHODIR [Young 80], GCR [Elman 82], GMRES [Saad 86], CGS [Sonneveld 89] e Bi-CGSTAB [Van der Vorst 92]) e *Processos-Galerkin Gradientes-Conjugados generalizados*(ex: GCG [Concus 76], Full Orthogonalization [Saad 86], ORTHORES [Young 80]). Para uma taxonomia destes métodos, remetemos o leitor aos trabalhos de Ashby [Ashby 90].

Desafortunadamente, nenhum destes métodos (ou suas variantes) resolvem com êxito todos, ou ainda uma grande classe de sistemas não simétricos. Métodos que trabalham bem sobre um sistema em particular podem falhar completamente sobre outro. Neste trabalho usaremos para nossas implementações o método Mínimo Residual Generalizado (GMRES), cujas características descreveremos a seguir.

4.1 Mínimo Residual Generalizado

O Mínimo Residual Generalizado ou GMRES é um método iterativo para resolver sistemas lineares não simétricos, o qual tem a propriedade de minimizar em cada passo a norma do vetor residual sobre um subespaço de Krylov. O algoritmo é derivado do processo de Arnoldi para a construção de uma base l_2-ortogonal dos subespaços de Krylov. GMRES pode ser considerado como uma generalização do algoritmo MINRES de Paige e Saunders (para matrizes simétricas) e é teoricamente equivalente ao método "Residual Conjugado Generalizado" (GCR) e a ORTHODIR.

No entanto, GMRES apresenta várias vantagens sobre GCR e ORTHODIR. A principal motivação para o desenvolvimento do GMRES foi a necessidade de resolver sistemas onde a matriz de coeficientes não é *positiva real*(*PR*), posto que os outros dois métodos equivalentes ao GMRES, o (*GCR*) e ORTHODIR, não são satisfatórios. Particularmente, o método *GCR* pode parar ("break-down") para uma matriz não *PR* (i.e., algumas variáveis tornasen-se indefinidas e o processo iterativo não pode continuar) e o ORTHODIR é menos estável que o *GCR* (ver [Saad 86]). O GMRES é considerado como um dos métodos mais robustos para resolver sistemas não simétricos.

4.1.1 Teoria

A k-iteracão do GMRES para a solução do sistema linear $Ax = b$, dada uma estimativa inicial para a solução, x_0, é a solução do problema de mínimos quadrados

$$min_{x \epsilon x_0 + \kappa_k} \|b - Ax\|_2. \tag{4.1}$$

O GMRES usa o processo de Arnoldi para calcular uma base ortonormal $v_1, v_2, ..., v_k$ do subespaço de Krylov $\mathcal{K}_k(A, v_1, k)$. A solução x_k é tomada como $x_0 + V_k y_k$, onde V_k é a matriz cujas colunas são os vetores v_k calculados pelo processo de Arnoldi. O processo de Arnoldi determina uma matriz superior de Hessenberg H_k.

No GMRES, y_k é o minimizador do problema de mínimos quadrados

$$\bar{H}_k y_k = \|r_0\|_2 e_1 \tag{4.2}$$

onde $\bar{H}_k$ é uma $(k+1) \times k$ matriz gerada depois de k passos do processo de Arnoldi, e e_1 é o vetor canônico de dimensão $k+1$. O problema de mínimos quadrados pode ser

resolvido usando uma fatorização QR de $\bar{H}_k$ usando rotações de Givens [Golub 89]; o minimizador y_k é obtido como a solução do sistema triangular

$$Ry_k = g$$

onde R é uma matriz triangular $k \times k$, Q é uma matriz $(k+1) \times (k+1)$ e g é um vetor cujos componentes são os primeiros k componentes de $Q\|r_0\|_2 e_1$.

Uma característica importante exibida pelo GMRES é que a norma residual da solução aproximada pode ser obtida sem explicitamente calcular x_k. Devido à forma de como y_k é construído, a norma residual é igual ao valor absoluto da $(k+1)$-componente de $Q\|r_0\|_2 e_1$.

A convergência deste método é boa, pois qualquer curva não crescente de convergência é possível para ele [Greenbaum 96].

4.1.2 Implementação

Uma implementação comum do GMRES é sugerida em [Saad 86] na qual basea-se no uso da ortogonolização modificada de Gram-Schmidt (método de Arnoldi). As transformações de Householder, as quais são relativamente custosas mas estáveis, também tem sido propostas. A aproximação de Householder resulta em um aumento considerável do custo computacional; porém, a convergência pode ser melhor, especialmente para sistemas mal-condicionados [Walker 88]. Do ponto de vista de paralelismo, a ortogonalização de Gram-Schmidt pode ser preferida, melhorando estabilidade e as propriedades de paralelização [Demmel 93]. Aqui adotamos a ortogonalização modificada de Gram-Schmidt.

O seguinte algoritmo descreve os principais passos do algoritmo **GMRES:**

1. Escolher x_0 e calcular $r_0 = b - Ax_0$ e $V_{:,1} = r_0/\|r_0\|$.

2. Para $k = 1, 2, ...$

   Para $j = 1, 2, ..., k$

   $\bar{H}^k_{j,k} = (AV_{:,j})^T V_{:,k},$

   $\hat{V}_{:,k+1} = AV_{:,k} - \sum_{j=1}^{k} \bar{H}_{j,k} V_{:,j},$

   $\bar{H}^k_{k+1,k} = \|\hat{V}_{:,k+1}\|$

   $V_{:,k+1} = \hat{V}_{k+1} / \bar{H}_{k+1,k}$

   Calcular uma fatorização QR de $\bar{H}_k$

   Calcular uma norma residual como $g = \|(Q\|r_0\|e_1)_{k+1}\|$

3. Resolver o sistema triangular $Ry = g$

   Calcular a solução aproximada $x_k = x_0 + V_{nxk} y_k$

onde a expressão $:, j$ denota a j-ésima coluna.

O número de operações do GMRES pode ser resumido como

$$\mathcal{O}_{GMRES} = \frac{k^2 + 3k}{2}\mathcal{O}_{Au} + \frac{k^2 + k}{2}\mathcal{O}_{u^T v}(n) + n\mathcal{O}_{u^T v}(k) + k\mathcal{O}_{\|u\|_2} + \frac{nk^2 + nk}{2}\mathcal{O}_* + nk\mathcal{O}_\div + k\mathcal{O}_{QR} + \mathcal{O}_{RS} \tag{4.3}$$

onde $\mathcal{O}_{QR}$ é o número das operações da fatorização de uma matriz superior de Hessenberg e $\mathcal{O}_{\mathcal{RS}}$ é o número de operações para resolver um sistema triangular superior de ordem k usando substituição por retrocesso ($\mathcal{O}_{\mathcal{RS}} = k\mathcal{O}_\div + \frac{1}{2}(k^2 - k)\mathcal{O}_*$). As expressões $\mathcal{O}_{u^T v}(n)$ e $\mathcal{O}_{u^T v}(k)$ representam o produto interno sobre os vetores de tamanho n e k. O termo $\frac{1}{2}(k^2 + k)\mathcal{O}_*$ é o número de multiplicações envolvidos na expressão $\sum_{j=1}^{k} \bar{H}^k_{j,k} V_{:,j}$, devido à estrutura de $\bar{H}_k$.

O exame do algoritmo GMRES acima mostra que a área de armazenagem cresce com o número de iterações requerido para a convergência. Considerando a estrutura das matrizes $\bar{H}_k$ e R e com algum código conveniente para minimizar o número de vetores necessários (i.e., um único vetor pode conter x_0 e x_k e $V(:, 1)$

pode armazenar r_0 e logo ser normalizado), a área de armazenagem requerido pelo GMRES pode ser dada como

$$S_{GMRES} = (2+k)n + 2k^2 + 5k + 3 \tag{4.4}$$

O fato que $\mathcal{O}_{GMRES}$ e S_{GMRES} cresce quadraticamente com o número de iterações (para k grande) é a razão para o uso de um algoritmo *reinicialisado* GMRES(m), onde a base ortonormal é calculada sobre um espaço de m dimensões fixas, e m é usualmente pequeno ($10 \leq m \leq 50$) se comparado a n [Saad 86].

4.1.3 Descrição do GMRES(m)

Em vez de gerar uma base ortonormal de dimensão k, como em GMRES, escolhe-se um valor $m < k$, usualmente pequeno ($10 \leq m \leq 50$), e gera-se uma aproximação da solução usando uma base ortonormal de dimensão m. Claramente, então, a quantidade de armazenagem necessária pode ser estimada com antecedência, e mantendo-o dentro dos limites disponíveis.

O GMRES(m) não para ("break-down") ([Saad 86],pp. 865). Porém, dependendo do sistema e do valor de m, pode produzir uma sequência estacionária de resíduos, e assim não alcançar convergência. Um resultado dado por [Saad 86], pp. 867, mostra que se A é real positiva, o GMRES(m) converge para qualquer x_0 se m é mais grande que um certo valor mínimo. Este limite sobre m não é definido, porém, o GMRES(m) pode convergir ainda se m é menor que esse mínimo.
As duas principais operações em GMRES(m) são o processo de Arnoldi e a solução do problema de mínimos quadrados

$$\bar{H}_m y_m = \beta_k e_1, \qquad \beta_k = \|r_k\|_2 \tag{4.5}$$

O primeiro é rico em operações vetor-vetor e as implicações desta propriedade para a paralelização do método será discutida no capítulo 6. A solução de (4.5) é obtida usando a fatorização QR de $\bar{H}_m$. No final de cada j-passo do processo de Arnoldi uma nova coluna de $\bar{H}_m$ tem sido produzida (note que $\bar{H}_c$ é armazenado na forma fatorada R) e é adicionado a R. Então R é atualizado aplicando a rotação anterior de Givens à primeiras $(j-1)$-linhas da nova coluna e uma nova rotação de Givens é aplicado para fazer zero o $(j+1,j)$-elemento. Assim, depois de m passos do processo de Arnoldi temos $Q\bar{H}_m = R$, onde Q é o produto das matrizes de rotação de Givens.

O minimizador y_m em (4.5) é a solução do sistema triangular

$$Ry_m = g, \qquad g = Q\beta_k e_1 \tag{4.6}$$

Note que g é o vetor obtido multiplicando Q pelo vetor canônico $e_1 = (1,0,...,0)^T$ de dimensão m. Esta formulação nos permite evitar calcular Q explicitamente, com a correspondente poupança em operações e armazenagem. As matrizes rotações usadas são da forma

$$P_j = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \eta_j & -\xi_j & & \\ & & & \xi_j & \eta_j & & \\ & & & & & 1 & \\ & & & & & & \ddots \\ & & & & & & & 1 \end{bmatrix}$$

onde η_j e ξ_j são os parâmetros rotação e $Q = P_j(P_{j-1}(P_{j-2}\cdots P_3(P_2P_1)))$ é uma matriz triangular inferior. Posto que Q é multiplicado por e_1 estamos interessados

somente nos elementos da primeira coluna de Q. O vetor g é assim da forma

$$g = \begin{bmatrix} \eta_1 & & & & \\ \xi_1 & \eta_2 & & & \\ \xi_1 & \xi_2 & \eta_3 & & \\ \vdots & & & & \\ \xi_1 & \xi_2 & \cdots & \xi_{m-1} & \eta_m \end{bmatrix}$$

Os elementos de g podem ser expressos por

$$g_j = \beta(\prod_{i=1}^{j-1} \xi_i)\eta_j, \quad j = 1, 2, .., m$$

Porém dado que η_j e ξ_j são gerados durante as iterações do processo de Arnoldi, podemos usar o seguinte procedimento para atualizar g. Antes que o processo de Arnoldi inicie, g é o vetor $(\beta_k, \beta_k, 0, .., 0)^T$ de dimensão $m+1$. Em cada j-passo, g é atualizado por

$$\begin{aligned} g_j &= \eta_j g_j \\ g_{j+1} &= \xi_j g_{j+1} \\ g_{j+2} &= g_{j+1} \end{aligned}$$

Apresentamos abaixo uma formulação algébrica do GMRES(m).

GMRES(m):

1. Escolher x_0 e calcular $r_0 = b - Ax_0$ e $V_{:,1} = r_0/\|r_0\|$.
2. Para $k = 1, 2, ...$

   Para $j = 1, 2, ..., m$

   $\bar{H}^k_{j,k} = (AV_{:,j})^T V_{:,k}$,

   $\hat{V}_{:,k+1} = AV_{:,k} - \sum_{j=1}^{k} \bar{H}_{j,k} V_{:,j}$,

$\bar{H}^k_{k+1,k} = \|\hat{V}_{:,k+1}\|$

$V_{:,k+1} = \hat{V}_{k+1}/\bar{H}_{k+1,k}$

Calcular uma fatorização QR de $\bar{H}_k$

Calcular uma norma residual como $\|(Q\|r_0\|e_1)_{k+1}\|$

3. Resolver o sistema triangular $Ry = g$

   Calcular a solução aproximada $x_m = x_0 + V_{nxm}y_m$

4. Reinicializar:

   Calcular $r_m = b - Ax_m$;

   Se é satisfeito PARE

   en caso contrario

   $x_o = x_m$,

   $v_1 = r_m/\|r_m\|$,

   ir ao passo 2.

onde a expressão $:, j$ denota a j-coluna.

5 MÉTODOS ITERATIVOS PARA A SOLUÇÃO DE SISTEMAS DE EQUAÇÕES NÃO-LINEARES

A solução de problemas lineares tem sido substancialmente estudada nestes tempos. Estes métodos têm sido amplamente aplicados, porém se tem sempre procurado melhores modelos do mundo real; tais modelos, mais refinados, passam a incluir termos não-lineares. Como conseqüência, tem se desenvolvido muitos métodos para resolver problemas não lineares.

Neste trabalho estamos interessados na computação de uma solução x^* do sistema de equações

$$F(x) = 0, \qquad F = (f_1, f_2, ..., f_n) \tag{5.1}$$

onde $F : \Omega \subset R^n \to R^n$.

Exemplos do problema (5.1) originam-se em uma variedade de contextos e algumas vezes como resultados de métodos matemáticos ou formulações de problemas, tais como em problemas com restrições, no qual (5.1) determina uma superficie restrita, e na solução de equações diferenciais não lineares por métodos implícitos [Walker 90].

Nas seguintes secções, desenvolveremos alguns métodos iterativos para resolver o problema (5.1), os quais foram objeto de estudo nesta dissertação.

5.1 Método de Newton

5.1.1 Teoria

Um método muito importante para a solução de (5.1) é o método de Newton [Ortega 70]. Este é um método iterativo para equações não lineares em termos da transição de uma iteração atual x_k a uma nova iteração x_{k+1}. Nestes termos, o método de Newton pode ser expresso como: para uma solução estimativa inicial x_0, geramos uma sequência de iterações $x_1, x_2, ...$ por

$$x_{k+1} = x_k + s, \tag{5.2}$$

onde s é a solução do sistema

$$J(x)s = -F(x_k). \tag{5.3}$$

Considerando as hipóteses padrão estabelecidas no capítulo (3), a sequência é bem definida e converge quadraticamente a x^*, ver [Kelley 95] ou [Dennis 83], i.e.

$$\exists\ C < 1\ \ tal\ que\ \ \|x_{k+1} - x^*\| \leq C\|x_k - x^*\|^2.$$

A hipotése que a iteração inicial seja "suficientemente próxima" à solução ($x_0 \epsilon \mathcal{B}(\delta)$) pode parecer artificial, porém existem muitas situações na qual a iteração inicial é muito próxima da raiz. Dois exemplos são a integração implícita de equações diferenciais ordinárias e equações diferenciais algébricas ([Brenan 89] e [Gear 71]), onde a iteração inicial é derivada da solução no passo anterior de tempo e, a solução de discretização de equações parciais onde a iteração inicial é uma interpolação de uma solução da malha computacional mais grossa [McCormick 89]. Além disso, quando a iteração inicial está longe da raiz é possível, mediante alguns métodos, transformá-la em uma iteração com convergência local, como veremos mais adiante.

5.1.2 Implementação

Nesta seção assumimos que os métodos diretos serão usados para resolver a equação linear (5.3). Nossos exemplos e contagens de operações assumem implicitamente que o Jacobiano é denso, porém, as considerações para Jacobianos esparsos quando métodos diretos são usados para resolver sistemas lineares são similares. Na próxima seção consideraremos algoritmos nos quais os métodos iterativos são usados para resolver a equação (5.3).

Para calcular a iteração x_{k+1} de um ponto atual x_k, deve-se avaliar primeiro $F(x_k)$ e decidir se é necessário continuar a iteração. Decidindo-se continuar, o Jacobiano $J(x_k)$ deve ser calculado e fatorizado. Logo a seguir, o passo é calculado com a solução de $J(x_k)s = -F(x_k)$ e a iteração é atualizada por $x_{k+1} = x_k + s$. Destes passos, a avaliação e a fatorização de J são os mais custosos. A fatorização de J no caso denso custa $\mathcal{O}(n^3)$ operações de ponto-flutuante. A avaliação de $J(x)$ por diferenças finitas deve ser esperada custar n vezes o custo de uma avaliação de F, pois cada coluna em J requer uma avaliação de F para formar a aproximação por diferenças. Portanto, o custo de um passo de Newton pode ser rapidamente estimado como $n+1$ avaliações de F e $\mathcal{O}(n^3)$ operações de ponto-flutuante. Em muitos casos, J pode ser calculado mais eficientemente, com mais precisão e diretamente, com diferenças tais que a análise acima para o custo de uma iteração de Newton seria muito pessimista.

O algoritmo de Newton, descrito a seguir, considera o uso de uma fatorização LU de J, mas qualquer outra fatorização apropriada como QR ou Cholesky pode ser utilizada.

Algoritmo de Newton:

1. Escolher x_0 e calcular $F(x_0)$ e $r_0 = \|F(x_0)\|$.
2. Enquanto $\|F(x)\| > \tau$ fazer
   (a) Calcular $J(x)$

(b) Factorar $J(x) = LU$

(c) Resolver $LUs = -F(x)$

(d) $x = x + s$

(e) Avaliar $F(x)$

Uma aproximação para reduzir o custo de (2.a) e (2.b) na iteração de Newton é movêr-los para fora do laço principal. Isto significa que a aproximação linear de $F(x) = 0$, resolvida em cada iteração fica determinada pela iteração inicial $x_{k+1} = x_k - J(x_0)^{-1}F(x_k)$. Este método é chamado o *método da corda*:

1. Escolher x_0 e calcular $F(x_0)$ e $r_0 = \|F(x_0)\|$.
2. Calcular $J(x)$
3. Factorar $J(x) = LU$
4. Enquanto $\|F(x)\| > \tau$ (vetor de tolerância)
   (c) Resolver $LUs = -F(x)$
   (d) Calcular $x = x + s$
   (e) Avaliar $F(x)$

A única diferença na implementação do método de Newton é que o cálculo e a fatorização do Jacobiano são feitas antes do início da iteração. A diferença na iteração é que uma aproximação a $J(x_k)$ é usada. Diferenças similares surgem se $J(x_k)$ é numericamente aproximado por diferenças.

5.2 Aproximação de J por diferenças finitas

O elemento na linha i e coluna j de $J(x)$ é $\partial f_i(x)/\partial x_j$, e é natural aproximá-lo por diferenças finitas, isto é,

$$\frac{\partial f_i(x)}{\partial x_j} \approx h_j^{-1}[f_i(x + h_j e_j) - f_i(x)] \tag{5.4}$$

onde $h_j \in R$ e $e_j \in R^n$ é a coluna j da matriz identidade $n \times n$. O tamanho de h_j determina a precisão da aproximação; cujas diretrizes para a sua seleção podem ser obtidas em [Dennis 83]. Aqui notamos imediatamente que toda a coluna j de $J(x)$ pode ser aproximada pela diferença

$$h_j^{-1}[F(x + h_j e_j) - F(x)] \tag{5.5}$$

se utilizamos o mesmo h_j para cada função componente f_i (em caso contrário, consideramos $h_{i,j}$). A forma (5.5) pode ser mais conveniente que (5.4), em situações nas quais diferentes funções componentes têm subexpressões comuns, ou quando as funções componentes não são rapidamente acessáveis de forma individual. Usando (5.5), vemos que $n + 1$ avaliações de F são suficientes para aproximar toda a matriz $J(x)$.

Para uma iteração quasi-Newton uma aproximação da matriz $J(x)$ é necessária, e nesse caso o valor de $F(x_k)$ requerido por (5.5) já é conhecido. Isto mostra que, examinando aproximações por diferenças finitas, dentro do contexto da iteração quasi-Newton, é possível reduzir o número de avaliações de F requeridas para calcular uma aproximação B_k de $J(x_k)$. Em particular, é freqüentemente possível explorar valores conhecidos de F das iterações anteriores para obter uma aproximação com menor custo. Outras diferenças, que não são da forma (5.4), são freqüentemente úteis para ditos propósitos. Numerosos destes esquemas no qual o número de avaliações de F por iteração pode ser consideravelmente menor que n, mas que não tomam em conta alguma esparsidade que possa existir, tem sido propostos em [Ortega 70], onde são dadas razões por quê ditos esquemas não são úteis.

Além disso, assume-se que ao calcular $F(x)$, temos um erro numérico - i.e., calculamos na verdade $F(x)+\epsilon(x)$ - e tentamos aproximar a ação de $F'(x)$ sobre um vetor por diferenças frontais, por exemplo, quando construímos um Jacobiano

aproximado. Uma aproximação por diferenças frontais para $J(x)w$ seria

$$\frac{F(x+hw)+\epsilon(x+h)-F(x)-\epsilon(x)}{h}$$

Assuma que $\|\epsilon(x)\| \leq \bar{\epsilon}$. Então

$$J(x)w - \frac{F(x+hw)+\epsilon(x+H)-F(x)-\epsilon(x)}{h} = \mathcal{O}(h+\bar{\epsilon}/h)$$

A quantidade dentro do $\mathcal{O}$-termo é minimizada quando $h = \sqrt{\bar{\epsilon}}$. Isto significa, por exemplo, que passos h muito pequenos podem conduzir a resultados imprecisos. No caso especial onde $\epsilon(x)$ é um resultado do arredondamento de ponto-flutuante com precisão total, por ex. ($\bar{\epsilon} \approx 10^{-15}$), esta análise indica que $h \approx 10^{-7}$ é uma escolha razoável; a escolha $h \approx 10^{-15}$ pode conduzir a um desastre numérico. Aqui temos feito a asserção implícita que x e w são aproximadamente do mesmo tamanho. Em caso contrário, h deve ser tratado para refletir isto. A escolha

$$h = \bar{\epsilon}^{1/2}\|x\|/\|w\|.$$

reflete a discusão acima.

Uma conseqüência importante desta análise é que a escolha do tamanho do passo, em uma aproximação por diferenças frontais para a ação do Jacobiano sobre um vetor, deve levar em conta o erro na avaliação de F. Isto podese tornar muito importante se parte de F é calculada com base em dados estimados.

Se $F(x)$ já foi calculado, o custo da aproximação frontal de $J(x)w$ é uma avaliação adicional de $F(x+hw)$. Portanto, a avaliação do Jacobiano completo por diferenças finitas custaria n avaliações da função, uma por cada coluna do Jacobiano. Explorando a estrutura especial que o Jacobiano porventura ofereça, se pode reduzir o custo.

A aproximação por diferenças frontais para a derivada não é um operador linear em w. A razão para isto é que a derivada tem sido aproximada, e assim a linearidade é perdida. Por isto devemos cuidadosamente especificar o que tomamos por aproximação do Jacobiano.

Definição 5.2.1 *Seja F definida em uma vizinhança de $x \in R^n$. $(\nabla_h)(x)$ é a matriz $n \times n$ cuja coluna j é dada por*

$$(\nabla_h)(x)_j = \begin{cases} \frac{F(x+h\|x\|e_j)-F(x)}{h\|x\|} & x \neq 0 \\ \frac{F(he_j)-F(x)}{h} & x = 0 \end{cases}$$

Fazemos uma definição similar da aproximação por diferenças da derivada direcional.

Definição 5.2.2 *Seja F definida em uma vizinhança de $x \in R^n$ e $w \in R^n$. Temos*

$$D_h F(x : w) = \begin{cases} 0, & w = 0, \\ \|w\| \frac{F(x+h\|x\|w/\|w\|)-F(x)}{h\|x\|} & w, x \neq 0, \\ \|w\| \frac{F(hw/\|w\|)-F(x)}{h} & x = 0, w \neq 0. \end{cases}$$

Para problemas nos quais a constante de Lipschitz de $J(x)$ é grande, o erro na aproximação por diferenças será também grande. Além disso, $D_h F(x : w)$ não é, em geral, uma função linear de w e é usualmente o caso em que

$$(\nabla_h F(x))_w \neq D_h F(x : w).$$

Se $J(x^*)$ é mal-condicionado ou a constante de Lipschitz de $J(x)$ é grande, a diferença entre eles pode ser significante e uma aproximação por diferenças para uma derivada direcional, o qual usa o tamanho de w, provavelmente seja mais

precisa do que $\nabla_h F(x)w$. Resumindo, podemos dizer que as aproximações numéricas a Jacobianos devem ser utilizados com cautela.

5.3 Métodos quasi-Newton

Um número de modificações têm sido propostas para reduzir o custo do método de Newton, tais como o método da corda ou o método simplificado de Newton e ainda uma variedade de métodos iterativos quasi-Newton [Ortega 70]. Para certos problemas estes métodos requerem menos armazenagem ou trabalho que o método de Newton. Porém, dificuldades computacionais são típicas na solução de sistemas não-lineares, e cada um destes métodos tem suas desvantagens. A escolha do método para um sistema não linear específico (5.1) depende enormemente das propriedades de F, da armazenagem e das restrições de tempo impostas pelo ambiente computacional.

O método simplificado de Newton substitui $J(x_k)$ com $J(x_0)$ em (5.3). Isto evita o cálculo de $J(x_k)$ em cada passo e reduz o custo resolvendo (5.3) para $k > 1$, dado que, se $J(x_0)$ é fatorado no produto $U^T DU$ no primeiro passo, cada passo seguinte requer a solução de um sistema diagonal e dois sistemas triangulares. Porém, este método somente converge q-linearmente e requer a mesma quantidade de armazenagem que o método de Newton.

Os métodos quasi-Newton mantem aproximações da solução e o Jacobiano quando a iteração progride. Se x_k e B_k são as aproximações atuais da solução e o Jacobiano respectivamente, então

$$x_{k+1} = x_k - B_k^{-1} F(x_k). \tag{5.6}$$

Após do cálculo de x_k, B_k, a solução aproximada é atualizada para formar B_{k+1}. A construção de B_{k+1} determina o método quasi-Newton.

As vantagens destes métodos é que as soluções das equações com uma aproximação quasi-Newton do Jacobiano são freqüentemente menos custosas do que usar $J(x_k)$ como matriz de coeficientes [Kelley 95]. O método que apresentamos nesta seção tem convergência superlinear localmente e, por tanto, é uma alternativa poderosa para o método de Newton.

Em comparação com os métodos Newton-iterativos é que os métodos quasi-Newton requerem somente uma avaliação da função para cada iteração não linear; não existe custo na avaliação da funções associadas com uma iteração interna. Portanto, se um bom precondicionador (aproximação inicial para $J(x^*)$) pode ser obtido, estes métodos poderiam ter uma vantagem em termos do custo da avaliação da funcão sobre os métodos Newton-iterativos.

O problema com os métodos iterativos quasi-Newton é que a razão total da convergência é essencialmente determinada pela razão de convergência do método iterativo linear aplicado no problema linearizado (5.3). Assim, eles encontram todas as dificuldades usuais dos métodos iterativos lineares relativos a estimação dos parâmetros, estimativas iniciais das soluções e critérios de parada.

Os erros no cálculo da função e de sua derivada afetam também o progresso na iteração de Newton [Kelley 95]. Outra forma de vermos isto é perguntar como uma solução aproximada de (5.3) afeta a iteração. Isto foi visto em [Dembo 82] onde os *métodos quasi-Newton* que satisfazem

$$\|J(x_k)s + F(x_k)\| \leq \eta_k \|F(x_k)\| \tag{5.7}$$

são considerados. Qualquer aproximação é aceita contanto que o resíduo relativo da equação linear (5.3) seja pequeno. Isto é bastante útil, pois condições como (5.7) são precisamente condições de terminação de resíduos lineares pequenos na solução iterativa do sistema linear para o passo de Newton.

O termo η_k do lado direito de (5.7) é chamado *termo força.* Para mostrar como um passo que satisfaz (5.7) afeta a convergência destes métodos, seguimos [Kelley 95] para apresentar dois teoremas a respeito

Teorema 5.3.1 *Sejam as hipóteses padrão válidas. Então existem δ e K_I tal que se $x_k \in \mathcal{B}(\delta)$, s satisfaz (5.7), e $x_{k+1} = x_k + s$ então*

$$\|e_{k+1}\| \leq K_I(\|e_k\| + \eta_k)\|e_k\|.$$

As implicações deste resultado para métodos iterativos são resumidos no seguinte teorema:

Teorema 5.3.2 *Sejam as hipóteses padrão válidas. Então existem δ e $\bar{\eta}$ tal que se $x_0 \in \mathcal{B}(\delta)$, $\{\eta_n\} \subset [0, \bar{\eta}]$, então a iteração de Newton inexata*

$$x_{n+1} = x_n + s_n,$$

onde

$$\|J(x_n)s_n + F(x_n)\| \leq \eta_n\|F(x_n)\|$$

converge q-linearmente a x^*. Além disso,

- *Se $\eta_n \to 0$ a convergência é q-superlinear, e*
- *Se $\eta_n \leq K_\eta\|F(x_n)\|^p$ para algum $K_\eta > 0$ a convergência é q-superlinear com q-ordem $1 + p$.*

Os requerimentos dos resultados anteriores podem ser mudados para admitir uma escolha mais agressiva do parâmetro η [Kelley 95]; assím, não exigimos que $\{\eta\}$ seja menor de 1 por uma quantidade suficiente grande, senão que 1 não seja

um ponto de acumulação. A importância deste resultado para aimplementação é que a escolha da sequência dos termos força $\{\eta_n\}$ (tal como $\eta_n = 0.5$ para qualquer n), os quais tentam minimizar o número de iterações internas, são completamente justificadas por este resultado, se a iteração inicial for suficientemente próxima da solução. Também se obtém convergência q-linear com a norma ponderada se a iteração é suficientemente próxima de x^*. Esta convergência é equivalente a convergência q-linear da sequência dos resíduos não-lineares $\{\|F(x_n)\|\}$ [Kelley 95].

5.3.1 Métodos Newton-iterativos

Um método *Newton-iterativo* realiza (5.7) com um método iterativo para o sistema linear (5.3) do passo de Newton, terminando quando o resíduo linear relativo é menor do que η_k (i.e. quando (5.7) é satisfeita). O nome do método indica qual método iterativo linear é usado, por exemplo, Newton-SOR (aproximação usando método semi iterativo), Newton-CG (se J é SPD), Newton-GMRES (sistemas não simétricos), Newton-Multigrid (uma aproximação usando um método iterativo estacionário). Tipicamente, a iteração não-linear que gera a sequência $\{x_n\}$ é chamada de *iteração externa* e a iteração linear que gera as aproximações aos passos é chamada de *iteração interna.*

5.3.1.1 Newton-GMRES

Proporcionamos uma análise detalhada da iteração Newton-GMRES, por ser uma parte importante em nossas implementações . Começamos discutindo os efeitos de uma aproximação por diferenças frontais à ação do Jacobiano sobre um vetor.

Se o método iterativo linear é algum dos métodos de subespaço de Krylov (ex. GMRES) então cada iteração interna requer ao menos uma avaliação

da ação de $F'(x_k)$ sobre um vetor. Em muitas implementações [Brown 89], a ação sobre um vetor w é aproximada por uma diferença frontal (5.2.2), $D_hF(x : w)$ para algum h. É importante notar que isto é inteiramente diferente de formar o Jacobiano ($\nabla_hF(x)$) por diferenças finitas e aplicar essa matriz a w. Com efeito, como é indicado em [Brown 87], a aplicação de Newton-GMRES à equação (5.3) com produtos matriz-vetor aproximados por diferenças-finitas é a mesma aplicação de GMRES à matriz $G_hF(x)$ cujas primeiras $(k-1)$-colunas são os vetores

$$v_k = D_hF(x : v_{k-1})$$

A sequência $\{v_k\}$ é formada no curso da iteração GMRES diferenças frontais. Para ilustrar isto, mostramos abaixo o algoritmo GMRES diferenças-frontais para o cálculo do passo de Newton, $s = -J(x)^{-1}F(x)$. Note que os produtos matriz-vetor são substituídos por aproximações em diferenças-frontais a $J(x)$. A sequência de passos aproximados$\{s_k\}$ é produzida, $b = -F(x)$, e a iteração inicial para o problema linear é o vetor zero.

Algoritmo 5.3.1 `n-gmres`(s,x,F,h, η ,kmax, ρ)

1. $s = 0, r = -F(x), v_1 = r/\|r\|_2, \rho = \|r\|_2, \beta = \rho, k = 0$
2. Enquanto $\rho > \eta\|F(x)\|_2$ e $k < kmax$ fazer
   a) $k = k + 1$
   b) $v_{k+1} = D_hF(x : v_k)$
      Para $j = 1, ...k$
         i) $h_{jk} = v_{k+1}^T v_j$
         ii) $v_{k+1} = v_{k+1} - h_{jk}v_j$
   c) $h_{k+1} = \|v_{k+1}\|_2$
   d) $v_{k+1} = V_{k+1}/\|v_{k+1}\|_2$
   e) $e_1 = (1, 0, .., 0)^T \epsilon R^{k+1}$
      Minimizar $\|\beta e_1 - H_k y_k\|_R^{k+1}$ para obter $y_k \epsilon R^k$

f) $\rho = \|\beta e_1 - H_k y_k\|_R^{k+1}$.

3. $x_k = x_0 + V_k y_k$

És claro do algoritmo anterior, que a diferença entre $\nabla_h F$ e $G_h F$ é que as colunas são calculadas tomando derivadas direcionais baseadas sob duas bases diferentes ortonormais. $\nabla_h F$ usa a base de vetores unitários nas direções coordenadas, e $G_h F$ a base para o espaço de Krylov $\mathcal{K}_k$ construído pelo algoritmo *newtmres*.

Assumindo que não existe erro na avaliação de F, podemos dizer que se $\eta \epsilon (0,1)$ então existe $\bar{h}$ e δ tal que se $x \epsilon \mathcal{B}(\delta)$, $h \leq \bar{h}$, e o algoritmo *newtmres* termina com $k < kmax$ se existe um s que satisfaz $\|J(x)s + F(x)\|_2 < (\eta + 4\gamma h)\|F(x)\|_2$. Isto implica que a implementação por diferenças-finitas não afetará o desempenho de Newton-GMRES se os passos na aproximação por diferenças finitas das derivadas são suficientemente pequenos [Kelley 95]. Por conseguinte, em uma implementação, devemos especificar não somente a sequência $\{\eta_n\}$ como também os passos $\{h_n\}$ usados para calcular as diferenças frontais.

O método para formar a sequência dos *termos força* é como segue. Ajustamos η quando a iteração progride; para isto seguimos o critério dado em [Eisenstat 94]. Esta escolha é motivada pelo desejo de evitar "*over solving*" (i.e. a equação linear para o passo de Newton é resolvida com precisão muito maior do que é preciso para corrigir a iteração não-linear); assim, consideramos

$$\eta_n^C = \begin{cases} \eta_{max}, & n = 0, \\ min(\eta_{max}, \eta_n^A), & \eta > 0, \gamma\eta_{n-1}^2 < 0.1, \\ min(\eta_{max}, max(\eta_n^A, \gamma\eta_{n-1}^2)), & n > 0, \gamma\eta_{n-1}^2 > 0.1 \end{cases}$$

onde a constante 0.1 é arbitrária e $\eta_n^A = \gamma\|F(x_n)\|^2/\|F(x_{n-1})\|^2$, onde $\gamma \epsilon (0,1]$. O parâmetro η_{max} é um limite superior da sequência $\{\eta_n\}$. Em nossa implementação são usados os valores $\gamma = 0.9$ e $\eta_{max} = 0.9999$, como é recomendado em [Eisenstat 94].

Existe uma possibilidade que a iteração final reduza $\|F\|$ muito longe do nível de precisão desejado e a solução da equação linear para o último passo seja mais precisa do que realmente necessitamos. Este "over-solving" sobre o passo final pode ser controlado comparando a norma do resíduo não-linear atual $\|F(x_n)\|$ com a norma do resíduo não linear $\tau_t = \tau_a + \tau_r\|F(x_0)\|$ e limitando η_n anterior por uma constante múltipla de $\tau_t/\|F(x_n)\|$, com o qual a iteração terminaria. Usamos então,

$$\eta_n = min\left(\eta_{max}, max(\eta_n^C, 0.5\tau_t/\|F(x_n)\|)\right). \tag{5.8}$$

Além disso, como o GMRES forma as iterações internas e faz decisões de terminação baseadas sobre produtos escalares e l^2-normas, também terminamos a iteração exterior sobre pequenas l^2-normas dos resíduos não lineares. Porém, devido a nossas aplicações em equações diferenciais, multiplicamos a l^2-norma do residuo não linear por um fator de $1/\sqrt{n}$, tal que as funções constantes tenham normas que sejam independentes da malha computacional.

Com tudo isto, podemos descrever o seguinte algoritmo:

Algoritmo 5.3.2 `newgm`(x, F, τ, η)

1. $r_c = r_0 = \|F(x)\|_2/\sqrt{n}$
2. *Enquanto* $\|F(x)\|_2/\sqrt{n} > \tau_r r_0 + \tau_a$
   - *a) Selecionar* η
   - *b)* n-*gmres*(s, x, F, η)
   - *c)* $x = x + s$
   - *d) Avaliar* $F(x)$
   - *e)* $r_+ = \|F(x)\|_2/\sqrt{n}$, $\sigma = r_+/r_c$, $r_c = r_+$
   - *f) Se* $\|F(x)\|_2 \leq \tau_r r_0 + \tau_a$ *terminar*

5.4 O Método de Broyden

O método de Broyden [Broyden 65] calcula B_{k+1} por

$$B_{k+1} = B_k + \frac{(y - B_k s)s^T}{s^T s} = B_k + \frac{F(x_{k+1})s^T}{s^T s}. \tag{5.9}$$

onde $y = F(x_{k+1}) - F(x_k)$ e $s = x_{k+1} - x_k$.

O método de Broyden é um exemplo de uma *secante atualizada*, isto é, a aproximação a $F'(x^*)$ satisfaz a *equação secante* $B_{k+1}s = y$. Os métodos secantes que localmente têm convergência superlinear podem ser escritos de tal maneira a manter o modelo esparso, simétrico, ou definido positivo do Jacobiano aproximado (ver [Dennis 83] para maiores detalhes).

O método de Broyden é também aplicado a equações lineares, tal como $Ax = b$, onde $B \approx A$ é atualizado. Um resultado importante para problemas lineares é a convergência em $2n$ iterações [Gerber 81].

Se os dados x_0 e B_0 são suficientemente bons, a iteração convergirá superlinearmente à raiz [Kelley 95].

5.4.1 Problemas não lineares

Para problemas não lineares a análise de convergência é local. Se as hipóteses padrão são válidas e x_0 e B_0 são suficientemente boas aproximações a x^* e $J(x^*)$, então a sequência de Broyden (x_n, B_n) existe para os dados (F, x_0, B_0) e as iterações de Broyden convergem q-superlinearmente a x^* [Kelley 95]. No caso não linear, os erros na aproximação do Jacobiano podem crescer quando a iteração progride. Mas, se os dados iniciais são suficientemente bons, este crescimento pode ser limitado e, por conseguinte, a seqüência de Broyden existe e a iteração converge para x^* pelo menos q-linearmente [Kelley 95]

5.4.2 Implementação

Nossa implementação do método de Broyden é relativa a um problema usado em mecânica de fluidos computacional [Engelman 81]. Assim, depois que a armazenagem disponível para a iteração é esgotada, deve-se reinicializar as iterações. Isto é similar à iteração GMRES reinicializada. A base para isto é a fórmula de *Sherman-Morrinson* [Sherman 49]

$$(B + uv^T)^{-1} = \left(I - \frac{(B^{-1}u)v^T}{1 + v^T B^{-1}u}\right) B^{-1} \tag{5.10}$$

onde B é não-singular, $u, v \in R^n$ e $1 + v^T B^{-1}u \neq 0$.

No contexto de uma sequência de atualizações de Broyden $\{B_n\}$ temos, para $n \geq 0$,

$$B_{n+1} = B_n + u_n v_n^T,$$

onde

$$u_n = F(x_{n+1})/\|s_n\|, \quad e \quad v_n = s_n/\|s_n\|.$$

Estabelecendo

$$w_n = (B_n^{-1}u_n)/(1 + v_n^T B_n^{-1}u_n)$$

vemos que, se $B_0 = I$,

$$B_n^{-1} = (I - w_{n-1}v_{n-1}^T)(I - w_{n-2}v_{n-2}^T)...(I - w_0 v_0^T) = \prod_{j=0}^{n-1}(I - w_j v_j^T) \tag{5.11}$$

Dado que o produto da matriz vazia é a matriz identidade, (5.11) é válido para $n \geq 0$. Portanto, a ação de B_n^{-1} sobre $F(x_n)$ (i.e. o cálculo do passo de Broyden) pode ser calculado a partir de $2n$ vetores $\{w_j, v_j\}_{j=0}^{n-1}$ com um custo de $\mathcal{O}(k_* n)$ (k_* indica o número de iterações) operações de ponto-flutuante. Além disso, o passo de Broyden para a operação sequinte é

$$s_n = -B_n^{-1} F(x_n) = -\prod_{j=0}^{n-1} (I - w_j v_j^T) F(x_n) \tag{5.12}$$

Dado que o produto

$$\prod_{j=0}^{n-2} (I - w_j v_j^T) F(x_n)$$

deve ser calculado como parte do cálculo de w_{n-1}, podemos combinar o cálculo de w_{n-1} e s_n como segue

$$\begin{aligned} w &= \prod_{j=0}^{n-2} (I - w_j v_j^T) F(x_n) \\ w_{n-1} &= C_w w \quad \textit{onde} \;\; C_w = (\|s_{n-1}\|_2 + v_{n-1}^T w)^{-1} \\ s_n &= -(I - w_{n-1} v_{n-1}^T) w \end{aligned} \tag{5.13}$$

Podemos, ainda, eliminar a necessidade de armazenar o vetor w_{n-1}. Note que (5.12) implica que, para $n \geq 1$

$$\begin{aligned} s_n &= w - C_w w (v_{n-1}^T w) = w(1 - C_w (C_w^{-1} - \|s_{n-1}\|_2)) \\ &= -C_w w \|s_{n-1}\|_2 = -\|s_{n-1}\|_2 w_{n-1} \end{aligned} \tag{5.14}$$

assim, para $n \geq 0$,

$$w_n = -s_{n+1}/\|s_n\|_2. \tag{5.15}$$

Por conseguinte, necessita-se armazenar somente os passos $\{s_n\}$ e suas normas para construir a sequência $\{w_n\}$. Realmente, podemos escrever (5.11) como

$$B_n^{-1} = \prod_{j=0}^{n-1} \left(I + \frac{s_{j+1} s_j^T}{\|s_j\|_2^2} \right) \tag{5.16}$$

Podemos usar (5.16) e (5.12) diretamente para calcular s_{n+1}, pois s_{n+1} aparece em ambos membros da equação

$$\begin{aligned} s_{n+1} &= -(I + \frac{s_{n+1} s_n^T}{\|s_n\|_2^2}) \prod_{j=0}^{n-1} \left(I + \frac{s_{j+1} s_j^T}{\|s_j\|_2^2} \right) F(x_{n+1}) \\ &= -\left(I + \frac{s_{n+1} s_n^T}{\|s_n\|_2^2} \right) B_n^{-1} F(x_{n+1}) \end{aligned} \tag{5.17}$$

Em vez disso, resolvemos (5.17) para s_{n+1} obtendo

$$s_{n+1} = -\frac{B_n^{-1} F(x_{n+1})}{1 + s_n^T B_n^{-1} F(x_{n+1})/\|s_n\|_2^2} \tag{5.18}$$

O requerimento de armazenagem é da mesma ordem como o GMRES, fazendo o método de Broyden competitivo em armazenagem com GMRES como resolvedor linear [Deuflhard 90].

Algoritmo 5.4.1 Broyden:

1. $r_o = \|F(x)\|_2, \quad n = -1$

$s_0 = -F(x), \quad itc = 0.$

2. Enquanto $itc < maxit$ *fazer*

a) $n = n + 1, \quad itc = itc + 1$

b) $x = x + s,$

c) Avaliar $F(x)$

d) Se $\|F(x)\|_2 \leq \tau_r r_0 + \tau_a$ *sair*

e) Se $n < nmax$ *então*

i) $z = -F(x)$

ii) Para $j = 0, n-1$

$z = z + s_{j+1} s_j^T z \ / \|s_{j-1}\|_2^2$

iii) $s_{n+1} = z/(1 + s_n^T z / \|s_n\|_2^2)$

f) se $n = nmax$ *então*

$n = -1; \quad s_0 = -F(x);$

5.5 Convergência Global

Por um *algoritmo globalmente convergente* entendemos um algoritmo com a seguinte propriedade: para qualquer estimativa inicial, a iteração converge à raiz de F, ou falha, para fazê-lo em um número pequeno de caminhos. Dos muitos de tais algoritmos, enfocaremos a classe dos métodos *"line-search"* e a *regra de Armijo* implementado inexatamente [Brown 94]. Selecionamos o modelo *"line-search"* devido a sua simplicidade, e porque é trivial adicionar um *"line-search"* a uma existente implementação localmente convergente do método de Newton já existente.

O algoritmo para uma função de uma variável consiste em calcular uma direção de busca d, o qual será a *direção de Newton*,

$$d = -f(x_k)/f'(x_k)$$

e, então, testar passos da forma $s = \lambda d$, com $\lambda = 2^{-j}$ para algum $j \geq 0$ até que $f(x + s)$ satisfaça

$$\mid f(x_k + \lambda d) \mid < (1 - \alpha\lambda) \mid f(x_k) \mid$$

Esta desigualdade é chamada *redução suficiente* de $\mid f \mid$. O parâmetro $\alpha \in (0, 1)$ é pequeno porém positivo, tentando fazer a desigualdade anterior tão fácil quanto possível de satifazer. Uma vez que uma redução suficiente tenha sido obtida, aceitamos o passo $s = \lambda d$. Esta estratégia é chamada de *regra de Armijo*.

Estendemos este algoritmo em duas formas. Primeiro, aceitamos qualquer direção que satisfaça (5.7). Escrevemos isto para a sequência de Armijo como

$$\|F'(x_n)d_n + F(x_n)\| \leq \eta_n \|F(n)\| \tag{5.19}$$

Outra extensão é permitir mais flexibilidade na redução de λ. Permitimos qualquer escolha que produza uma redução que satisfaça

$$\sigma_0 \lambda_{velho} \leq \lambda_{novo} \leq \sigma_1 \lambda_{velho}$$

onde $0 < \sigma_0 < \sigma_1 < 1$. O perigo aqui é que o mínimo pode estar perto do zero para ser de muito uso e, realmente, a iteração pode estagnar-se. O parâmetro σ_0 é empregado para se proteger contra isto.

Algoritmo de Armijo:

1. $r_0 = \|F(x)\|$
2. Enquanto $\|F(x)\| > \tau r_0 + \tau_a$ fazer
   a) Achar d tal que $\|F'(x)d + F(x)\| \leq \eta\|F(x)\|$
      Se nenhum d pode ser achado, terminar com falha
   b) $\lambda = 1$
      i) $x_t = x + \lambda d$
      ii) Se $\|F(x_t)\| < (1 - \alpha\lambda)\|F(x)\|$ então $x = x_t$
          else
          Escolher $\sigma\epsilon[\sigma_0, \sigma_1]$
          $\lambda = \sigma\lambda$
          goto (2.b) i

Note que o passo (2a) deve permitir a possibilidade que J seja mal-condicionado e que nehuma direção pode ser achada que satisfaça (5.19).

Seja $\{ x_n\}$ a iteração produzida com iteração inicial x_0. O algoritmo pode falhar em alguns caminhos óbvios:

1. $J(x_n)$ é singular para algum n. A iteração interna poderia terminar com uma falha.
2. $x_n \to \bar{x}$, um mínimo local de $\|F\|$ o qual não é uma raiz.
3. $\|x_n)\| \to \infty$

6 IMPLEMENTAÇÃO PARALELA

A importância dos algoritmos paralelos - i.e., aqueles que permitem execução simultânea, terminando mais rapidamente do que um algoritmo seqüêncial equivalente - é que utilizando computadores com arquitetura paralela a convergência pode ser obtida em um tempo mais curto que sob computadores seqüênciais.

6.1 Aspectos gerais

A parte central desta dissertação é o desenvolvimento e paralelização dos algoritmos que podem ser usados para aproximar a solução da equação não linear (5.1).

Como ressaltamos anteriormente, o principal método para aproximar uma solução de (5.1) é o método de Newton, sendo a parte mais complexa deste algoritmo (pela presença do Jacobiano) o sistema de equações (5.3).

Neste trabalho implementamos dois algoritmos paralelos para resolver (5.1), especificamente o algoritmo Newton-GMRES e o algoritmo de Broyden. Estes algoritmos foram implementados de maneira a permitir ao usuário completa liberdade com respeito ao armazenamento, acesso e particionamento das matrizes.

Começamos este capítulo descrevendo os aspectos gerais de nossas implementações paralelas.

6.1.1 Modelo de programação

Nossos algoritmos foram implementados usando o modelo *"Single Program, Multiple Data"* (SPMD), o qual escreve um único código que roda sobre todos

os processadores cooperando sob uma tarefa. Os dados são particionados entre os processadores os quais conhecem sobre que porções dos dados irão trabalhar.

6.1.2 Particionamento de dados

Em nosso trabalho tratamos com três tipos diferentes de dados; valores escalares, vetores e matrizes. Somente estes dois últimos são possíveis de particionamento de dados; os valores escalares devem estar presentes em todos os processadores. Se um valor escalar é derivado de alguma computação sobre dados distribuídos entre os processadores, o uso do modelo *SPMD* pode resultar em uma alta faixa de comunicação e o resultado deverá ser enviado a todos os processadores. Por exemplo, dita comunicação é necessária quando calcula-se produtos internos e vetores 2-normas. Porém, ainda é possível alcançar bom desempenho [da Cunha 92].

O particionamento de dados que usamos é o *particionamento de dados por contigüidade*, definido como segue. Para particionar os dados (vetores e matrizes) entre os processadores, dividimos o conjunto de variáveis $V = \{i\}_{i=1}^{n}$ em p subconjuntos $\{W_i\}_{i=1}^{p}$ de $m = n/p$ elementos cada um; definindo cada subconjunto por $W_p = \{(p-1)m + j\}_{j=1}^{m}$.

Cada processador p é responsável pela execução das computações sobre as variáveis contidas em W_p. No caso de operações vetoriais, cada processador manterá segmentos de m variáveis.

Para o produto matriz-vetor, $v = Au$, cada processador p manterá um conjunto W_p de m elementos dos vetores u e v. Com relação às matrizes utilizadas, consideramos sua estrutura esparsa (ver, (3.11) e (3.12)); para armazená-las. A matriz A (pentadiagonal), é armazenada em uma matriz com 5 colunas; a matriz B (tridiagonal superior), em uma matriz com três colunas e similarmente a matriz resultante (pentadiagonal) da função que governa a equação diferencial parcial não linear; logo eles são particionados por blocos, onde cada bloco tem n/p linhas. Por

exemplo, se consideramos uma malha com 4 colunas e 2 linhas, teriamos a matriz

$$\left[\begin{array}{cccc:cccc} \bullet & \bullet & \bullet & 0 & 0 & 0 & 0 & 0 \\ \bullet & \bullet & 0 & \bullet & 0 & 0 & 0 & 0 \\ \bullet & 0 & \bullet & \bullet & \bullet & 0 & 0 & 0 \\ 0 & \bullet & \bullet & \bullet & 0 & \bullet & 0 & 0 \\ 0 & 0 & \bullet & 0 & \bullet & \bullet & \bullet & 0 \\ 0 & 0 & 0 & \bullet & \bullet & \bullet & 0 & \bullet \\ 0 & 0 & 0 & 0 & \bullet & 0 & \bullet & \bullet \\ 0 & 0 & 0 & 0 & 0 & \bullet & \bullet & \bullet \end{array}\right]_{2\times 4} \tag{6.1}$$

Esta matriz é particionada por colunas (para dois processadores), tal como indica a linha pontilhada. Seu armazenamento é feito como uma matriz da seguinte forma

$$\left[\begin{array}{ccccc} \bullet & \bullet & 0 & \bullet & 0 \\ \bullet & 0 & \bullet & \bullet & 0 \\ \bullet & \bullet & 0 & \bullet & \bullet \\ \bullet & 0 & \bullet & \bullet & \bullet \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \bullet & \bullet & 0 & \bullet & \bullet \\ \bullet & 0 & \bullet & \bullet & \bullet \\ \bullet & \bullet & 0 & 0 & \bullet \\ \bullet & 0 & \bullet & 0 & \bullet \end{array}\right]_{8\times 5} \tag{6.2}$$

onde as diagonais da matriz (6.1) foram armazenadas como colunas na matriz (6.2); esta matriz é particionada como indica a linha pontilhada.

6.1.3 Balanço de carga

O particionamento de dados usado em nossas implementações designa a mesma quantidade de dados para um sistema denso a cada processador (assumindo que n é um inteiro múltiplo de p), alcançando assim um balanço de carga entre os processadores.

Quando n não é um múltiplo de p, minimizamos os efeitos do inevitável desequilíbrio de carga usando a seguinte estratégia. Seja $r = n \bmod p$ o número de variáveis que ficam de sobra. Posto que $r < p$, designamos uma variável mais para cada um dos r primeiros processadores ($p = 0, 1, 2, ..., r - 1$). Cada um destes processadores então terá a mesma quantidade de dados ($(n-1)/p + 1$ variáveis) designadas a eles; os restantes $p - r$ processadores armazeram $(n - r)/p$ variáveis. Como um exemplo, considere $n = 23$ e $p = 5$. Os três primeiros processadores armazenaram 5 variáveis e os últimos processadores 4.

Os vetores são armazenados localmente começando da posição 1; assim tem-se uma enumeração local das variáveis o qual pode ser deslocado a uma enumeração global se for requerido. Por exemplo, se um vetor de 10 elementos é particionado entre dois processadores, usando blocos de comprimento 1, então o primeiro processador armazena os elementos 1, 2, 3, 4, 5 nas primeras cinco posições de um arreglo; o segundo processador então armazena os elementos 6, 7, 8, 9, 10 em posiçoes 1 a 5 sobre seu arregro.

6.1.4 Subrotinas Básicas de Álgebra Linear

O termo "Basic linear Algebra Subroutines" (BLAS) foi introduzido em [Dongarra 88] e refere-se a um conjunto de subrotinas que implementam várias operações de álgebra linear (OAL) .

O BLAS foi implementado em três níveis; o nível 1 envolve operações vetor-vetor, o nível 2, operações matriz-vetor e o nível 3, operações matriz-matriz. Estes subrotinas são codificadas em Fortran e estão disponíveis em simples e dupla precisão, usando dados reais e complexos.

Como mencionamos anteriormente, a maioria dos métodos iterativos podem ser expressos em termos das OAL operações. Quatro operações ocorrem freqüentemente: produtos internos, vetores 2-normas, acumulação de vetores (adição / substração de vetores ou "saxpys") e produtos matriz-vetor. Estas rotinas pertencem ao nível 1 ou 2 do BLAS. Na continuação definimos as principais rotinas do BLAS que usamos em nossa implementação; cada operação com seu custo associado em termos de operações aritméticas:

Sejam u, v e w são n-vetores; α um escalar real e A é uma $n \times n$-matriz.

`PRODUTO INTERNO:` $\alpha = \|u\|_2$

`PSNRM(loclen,u):`

$$\begin{aligned} \alpha &= \sqrt{u^T v} \\ \mathcal{O}_{\|u\|} &= n\mathcal{O}_* + (n-1)\mathcal{O}_+ + \mathcal{O}_{\sqrt{\cdot}} \end{aligned}$$

`SSCAL:` $v = \alpha u$

`SSCAL(loclen,`α`,u,su):`

$$\begin{aligned} u_i &= \alpha u_i; \quad i = 1, ..., n \\ \mathcal{O}_{\alpha u} &= n\mathcal{O}_* \end{aligned}$$

SCOPY: $v = u$

SCOPY(loclen, u, su, v, sv) :

$$\begin{aligned} v_i &= u_i; \quad i = 1, ..., n \\ \mathcal{O}_{u=v} &= n \text{ transferências de palavras de precisão simples.} \end{aligned}$$

SINIT: $u = \alpha$ Inicialização de um vetor.

SINIT(loclen, α, u, su) :

$$\begin{aligned} u_i &= \alpha; \quad i = 1, ..., n \\ \mathcal{O}_{u=\alpha} &= n \text{ atribuções} \end{aligned}$$

SDOT: $w = v^T * u$

SDOT(loclen, v, sv, u, su) :

$$\begin{aligned} w &= \sum_{i}^{n} v_i^T * u_i; \\ \mathcal{O}_{v^T * u} &= n\mathcal{O}_* + (n-1)\mathcal{O}_+ \end{aligned}$$

SAXPY: $w = u + \alpha v$

SAXPY(loclen, α, v, sv, u, su) :

$$\begin{aligned} w_i &= u_i + \alpha v_i; \quad i = 1, .., n \\ \mathcal{O}_{u+\alpha v} &= n(\mathcal{O}_* + \mathcal{O}_+) \end{aligned}$$

PRODUTO MATRIZ-VETOR: $v = Au$

Subrotinas MATVEC e PSMVPDE:

$$\begin{aligned} v_i &= A_i^T u; \quad i = 1, .., n \\ \mathcal{O}_{Au} &= n^2 \mathcal{O}_* + (n^2 - n)\mathcal{O}_+ \end{aligned}$$

onde A_i é a i-linha de A. Note que o produto matriz-vetor é a única operação cujo custo é de ordem n^2.

Uma rotina especial que usamos em nossa implematação é a rotina $SLAMCH$ do LAPACK (Linear Algebra Package) cujo propósito é determinar parâmetros da máquina de precisão simples. Esta rotina é usada para considerar um critério de parada no algoritmo de Broyden.

6.1.4.1 Saxpy

A operação saxpy tem a caraterística que seu cálculo é disjunto elemento a elemento com respeito dos vetores u, v e w. Isto significa que podemos calcular um saxpy sem comunicação entre processadores; o vetor resultante não precisa ser distribuido entre os processadores. Paralelismo é explorado no saxpy pelo fato que p processadores calcularam a mesma operação com uma quantidade mais pequena de dados. O saxpy é calculado como $w_i = u_i + \alpha v_i, \quad \forall i \in \{W_j\}_{j=1}^{p}$.

6.1.4.2 Produto interno

O produto interno é uma operação que envolve acumulação de dados, implicando um alto nível de comunicação entre todos os processadores. A topologia da rede e a arquitetura dos processos usados permitem uso mais eficiente dos

processadores. O uso do modelo SPMD também implica na difusão global do valor final calculado para todos os processadores.

O produto interno é calculado em três fases. A fase 1 é o cálculo das somas parciais da forma $\alpha = \sum_{\forall i \epsilon \{W_m\}} u_i \times v_i, \quad m = 1, ..., p$. A fase 2 é a acumulação destes somas parciais através da rede dos processadores. Esta operação realiza-se até que a redução total dos valores α é realizada através de comandos especiais proporcionadas pelo PVM, armazenando o resultado final, em nosso caso, no processador 0. A terceira fase é a difusão do valor do produto interno em todos os processadores. Uma vez que o valor do produto interno é recebido pelo processador 0, este calcula a raiz quadrada deste valor obtendo-se, assim, o valor da 2-norma requerida.

6.1.4.3 Produto Matriz-vetor

Em nossas aplicações aparecem produtos matriz-vetor onde a matriz é densa e esparsa. Para o caso denso usamos uma subrotina do BLAS - *sgemv.f* - sobre cada partição da matriz e o vetor e calcula o produto matriz-vetor por linhas.

No caso da matriz esparsa, a matriz é particionada como se indicou em (6.1) e logo cada processador armazena uma porção da matriz, usando uma estrutura de dados, o qual é composto de 5 arreglos em um caso e 3 em outro (ver 6.2). Para realizar o produto matriz-vetor Au, fazemos uma subrotina que considere a posição dos elementos da matriz A como no caso denso e usamos subrotinas do PVM para enviar e receber, entre processadores vizinhos, os elementos correspondentes do vetor u necessários para o produto.

6.2 Newton-GMRES paralelo

Neste caso implementamos um algoritmo paralelo denominado **newtar** cuja lista de argumentos é como segue

```
SUBROUTINE NEWTAR(U,WRK,IPAR,SPAR,MATVEC,FEVAL,JACOBIAN,PRECONL,
+                 PRECONR,PSSUM,PSNRM,PROGRESS)
```

onde os parâmetros são como segue

Parâmetro	*Descrição*
U	Um vetor de comprimento IPAR(4)
	Na entrada, contém o valor estimado inicial
	Na saída, contém o último valor estimado calculado
WRK	Um vetor de trabalho usado internamente, com comprimento
	igual a 9*IPAR(4) + (4+BASE)*IPAR(4)
IPAR	Um vetor inteiro contendo parâmetros de entrada-saída
SPAR	Um vetor real com precisão simples contendo parâmetros de entrada-saída
MATVEC	Subrotina externa para o produto matriz-vetor
FEVAL	Subrotina externa para calcular o valor da função em um ponto
JACOBIAN	Subrotina externa para calcular a matriz jacobiana da função
PRECONL	Subrotina externa par precondicionamento pela ezquerda
PRECONR	Subrotina externa par precondicionamento pela direita
PSSUM	Função externa (redução) soma global
PSNRM	Função externa para calcular o vetor norma
PROGRESS	Rotina de monitorar

Ver o manual de referência para a descrição dos parâmetros e a sinopse das rotinas externas.

Uma das características de nossa implementação é que nem todas as rotinas externas precisam ser fornecidas, pois os parâmetros "dummy" podem ser

sustituidos por aqueles que não são usados. Em nosso caso, acontece com as subrotinas PRECONL, PRECOND e no caso sequencial com PSSUM. As rotinas externas têm uma lista de parâmetros fixos, às quais o usuário deve seguir. Também é bom ressaltar que as matrizes de coeficientes e o cálculo do valor da função não aparecem na lista de parâmetros das rotinas, já que consideramos estes como *operadores* retornando somente o vetor resultante apropriado. Assim nossas rotinas principais não tem conhecimento da forma na qual as matrizes são armazenadas. Isto permite, segundo o caso, aproveitar a estrutura definida das matrizes e funções. As rotinas externas accesam as matrizes, vetores e algumas constantes declaradas no programa principal via blocos COMMON.

Quando as rotinas externas precisam calcular um produto interno, chamam SDOT para calcular os valores parciais dos produtos internos. Logo, a rotina PSSUM é usada para calcular a soma global destas somas parciais. Quando executamos nossas rotinas sobre um computador sequencial, estas rotinas são *vazias* i.e., o conteúdo do vetor U não deve ser alterado de forma alguma posto que seus elementos já são os valores do produto interno.

Temos incluído também na lista de parâmetros, uma subrotina externa (chamada PROGRESS) a qual recebe da rotina o número dos vetores armazenados localmente, o número da iteração atual, a norma do resíduo, o vetor de iteração atual, o vetor resíduo e o vetor resíduo *verdadeiro* $r_k = b - Ax_k$. Note que no caso de problemas de grandes dimensões esta rotina produzirá uma quantidade grande de saída. No caso que não seja necéssario ter esta informação, uma rotina "dummy" tem que ser provista.

A rotina *newtar* permite o uso de precondicionadores, o usuário pode escolher um precondicionamento pela esquerda, direita ou simétrico. Considerando esta lista de argumentos e as subrotinas do BLAS paralelizamos o algoritmo. Selecionamos η segundo o estabelecido em (5.8). Logo determinamos um valor estimado para a solução de (5.1) e usamos a regra de Armijo para convergência global, a

qual através de uma interpolação polinomial permite escolher o λ adequado para a convergência.

6.2.1 *Paralelização de GMRES(m)*

A solução do sistema (5.3) no método Newton-GMRES é obtida através da rotina {*PIMSRGMRES*} presente no pacote Parallel Iterative Methods (PIM) ([da Cunha 95]), detalhado a seguir.

Uma versão paralela do GMRES(m) foi obtida pelo uso das versões paralelas das operações que formam o algoritmo GMRES(m) dado no capítulo 3. Para isto seguimos os delineamentos dados em [da Cunha 94].

Examinando o algoritmo de GMRES(m) no capítulo 3, subseção (4.1.3) notamos que este contém operações básicas como acumulação de vetores (subtração e soma de vetores), produtos escalar de vetores, soma de vetores, produtos internos, vetores 2-normas, produtos matriz-vetor e um resolvedor triangular. As implementações destas operações proporcionam diferentes, mas aceitáveis, níveis de eficiência dependendo da dimensão do problema e o número de processadores. Note que, resolvedores triangulares são conhecidos por oferecer menor eficiência que outras operações de álgebra linear [da Cunha 92]. Optou-se, então, por fazer com que cada processador tenha sua própria cópia de R, y^m e g para resolver o sistema triangular $Ry^m = g$ localmente. As atualizações de x podem ser consideradas como uma sequência de somas e produtos escalares, reescrevendo $x_{k+1} = x_k + Vy_m$ quando $x_{k+1} = x_k + y_j^m V_j, j = 1, 2, .., m$. Estas operações podem ser executadas independentemente em cada processador.

O processo de ortonormalização de Arnoldi faz substancial uso de operações produto vetor-vetor com uma baixa granularidade, requerendo $(m^2+m)/2$ produtos internos e m vetores 2-normas. Existe um número de operações que é executado localmente, sem alguma comunicação.

6.3 Broyden Paralelo

Para o método de Broyden implementamos um algoritmo paralelo denominado `broyden` cuja lista de argumentos é como segue.

```
SUBROUTINE BROYDEN(U,FEVAL,IPAR,SPAR,WRK,PSSUM,PSSUM,PSNRM)
```

onde os parâmetros são como segue

Parâmetro	*Descrição*
U	Um vetor de comprimento IPAR(4)
	Na entrada, contém o valor estimado inicial
	Na saída, contém o último valor estimado calculado
WRK	Um vetor de trabalho usado internamente, com comprimento
	igual a 9*IPAR(4) + (4+BASE)*IPAR(4)
IPAR	Um vetor inteiro contendo parâmetros de entrada-saída
SPAR	Um vetor real com precisão simples contendo parâmetros de entrada-saída
FEVAL	Subrotina externa para calcular o valor da função em um ponto
PSSUM	Função externa (redução) soma global
PSNRM	Função externa para calcular o vetor norma

Ver o manual de referência para a descrição dos parâmetros e a sinopse das rotinas externas.

Considerando esta lista de argumentos e as subrotinas do BLAS, paralelizamos o algoritmo de Broyden (5.4.1). Nesta implementação usamos uma aproximação reinicializada do algoritmo, adicionando aos argumentos de entrada (como no algoritmo Newton-GMRES) um inteiro *nmax* sobre o número de iterações de Broyden antes de reinicializar e um limite *maxit* sobre o número de iterações não lineares. Notemos também que $B_0 = I$ (matriz identidade) está implícito no algoritmo.

Se $n < nmax$, nossa implementação calcula x_n e s_{n+1}, e requer $\mathcal{O}(nN)$ operações de ponto-flutuante e armazenagem de $n+3$ vetores (os passos $\{s_j\}_{j=0}^{n}, x, z, F(x)$ onde z e $F(x)$ podem ocupar a mesma área de armazengem). Se $n = nmax$, a iteração se reinicializará com x_{nmax}, não calcula s_{nmax+1} e, por conseguinte, necessita armazenar $nmax + 2$ vetores.

No passo 2.(e)ii (pag. 43) temos uma operação álgébrica a qual é muito importante para nossa implementação paralela: $s_{j+1}s_j^T z$. Esta operação é implementada assim. O produto $s_j^T z$ é armazenado em um vetor, o qual é multiplicado com o vetor s_{j+1} produzido no passo 2.(e)iii da iteração anterior. Estes resultados são atualizadas em cada iteração.

6.4 Critério de Parada

A seleção do critério de parada tem um efeito substancial sobre o tempo de execução. Evidentemente, existe um "trade-off" entre o tempo esgotado sobre cada iteração e o número de iterações requeridas para a convergência.

O método GMRES(m) usa seu proprio critério de parada o qual é equivalente à 2-norma do resíduo. Em todos os códigos e algoritmos desenvolvidos em nossa implementação paralela, os critérios de terminação são:

- número inteiro *maxit* (máximo número de iterações permitidos); ou
- se $\|F(x)\| \leq \tau_r \|F(x_0\| + \tau_a$, onde τ_r é a tolerância do erro relativo e τ_a é a tolerância do erro absoluto, são satisfeitos.

Todos estes parâmetros são ingressados no algoritmo.

6.5 Resultados numéricos

Para ilustrar e comparar os algoritmos mencionados neste trabalho consideramos os "objetos testes" descritos no capítulo dos preliminares.

6.5.1 Equação da Convecção - Difusão

A equação (3.10), é discretizada usando uma malha de $l \times l$ pontos internos. O sistema de equações que governam (3.10) é derivado da discretização da equação por diferenças-finitas de cinco pontos. Considerando as matrizes A e B obtida em (3.10), a função que governa toda a equação é

$$Fu = Au + C(Bu)u - f$$

Esta função é não linear e é descrita por um sistema de equações também não linear. Para transformar o sistema em um sistema linear calculamos o Jacobiano desta função e, assim, obtemos um sistema de equações linear pentadiagonal, cuja matriz de coeficientes J é particionada e armazenada segundo indicado nos aspectos gerais da implementação paralela.

Sequencialmente se verifica que na aproximação da solução desta equação não se tem convêrgencia para $C \neq 0$ [Kelley 95], sendo crucial o precondicionamento para obter boa perfomance. Neste caso, quando se faz uso do "rápido resolver de Poisson" (*fast Poisson solver*) como precondicionador se obtém convêrgencia para $C = 20$. Considerando tudo isto, fazemos só o análise por iteração.

As tabelas (6.1), (6.2) e as figuras (6.1), (6.2) correspondentes mostram o tempo em segundos por iteração do algoritmo Newton-GMRES e Broyden respectivamente, o número de processadores usados e a dimensão do problema.

Tabela 6.1 *Resultados do algoritmo Newton-GMRES*: Iter.(GMRES) = 10, C = 0.1, Base = 10

		Tempo(s)		
Malha	n	p=1	p=2	p=3
10×20	200	0.53000	2.93000	2.98500
20×40	800	2.05000	3.68400	3.71500
40×80	3200	7.58000	6.57100	5.67880
80×160	12800	30.78300	18.18300	14.56840
160×320	51200	135.24800	65.94500	48.80840

Tabela 6.2 *Resultados do algoritmo Broyden*: C = 0.1, Base = 10

		Tempo(s)		
Malha	n	p=1	p=2	p=3
10×20	200	0.18600	0.221000	0.232500
20×40	800	0.25200	0.26300	0.24000
40×80	3200	0.2730290	0.27760	0.24630
80×160	12800	0.67830	0.52990	0.49040
160×320	51200	1.45800	1.02080	0.93940

Estes resultados mostram para o algoritmo Newton-GMRES, que a partir de um sistema de equações com dimensão 40×80 é menor o tempo por iteração quando são usados mais de um processador. Entretanto para o algoritmo Broyden, é menor o tempo por iteração para um sistema de equações com dimensão 20×40 quando é usado 3 processadores.

6.5.1.1 "Speed-up" como uma função de paralelismo e número de processadores

As figuras (6.3), (6.4) e a tabela (6.3) mostram a aceleração do algoritmo Newton-GMRES paralelo em relação ao algoritmo Newton-GMRES sequencial.

Tabela 6.3 "Speed-up": Newton-GMRES

Malha	n	p = 2	p = 3
10 × 20	200	0.18090	0.17760
20 × 40	800	0.55650	0.55180
40 × 80	3200	1.15360	1.33480
80 × 160	12800	1.69300	2.11300
160 × 320	51200	2.05090	2.77100

A tabela (6.4) e as figuras (6.5),(6.6) mostram a aceleração do algoritmo Broyden-paralelo em relação ao algoritmo Broyden-sequencial.

Tabela 6.4 "Speed-up": Broyden

Malha	n	p = 2	p = 3
10 × 20	200	0.84160	0.80000
20 × 40	800	0.95820	1.05000
40 × 80	3200	1.09123	1.23000
80 × 160	12800	1.21801	1.38310
160 × 320	51200	1.42830	1.5520

6.5.2 Equação H de Chandrasekhar

A equação (3.8), é discretizada sob uma malha de n-pontos. A iteração inicial é a função identicamente um.

As tabelas (6.5), (6.6) e as figuras (6.7), (6.8) mostram o tempo em segundos por iteração do algoritmo Newton-GMRES e Broyden respectivamente, o número de processadores usados e a dimensão do problema.

Tabela 6.5 *Resultados do algoritmo Newton-GMRES*: Iter.(GMRES) = 10, C1 = 0.9, Base = 10.

	Tempo(s)		
n	p=1	p=2	p=3
100	1.62666	2.10300	2.89500
500	14.67400	14.80880	15.12940
1000	20.45270	18.57100	17.28300
2000	34.51600	30.71830	27.87400

Tabela 6.6 *Resultados do algoritmo Broyden*: C1 = 0.9, Base = 10.

	Tempo(s)		
n	p=1	p=2	p=3
100	0.02200	0.09300	0.21815
500	0.31055	0.58400	0.58715
1000	1.24500	1.23510	1.13060
2000	4.81611	4.18300	3.90930

A tabela (6.5) e a figura (6.7) mostram para o algoritmo Newton-GMRES, que a partir de $n = 500$ é menor o tempo por iteração quando é usado tres processadores. Entretanto para o algoritmo Broyden, a tabela (6.6) e a figura (6.8) monstram que para um sistema de $n = 1000$ é menor o tempo por iteração quando é usado mais de um processador.

6.5.2.1 "Speed-up" como uma função de paralelismo e número de processadores

A tabela (6.7) e as figuras (6.9), (6.10) mostram a aceleração do algoritmo Newton-GMRES paralelo em relação ao algoritmo Newton-GMRES sequencial.

Tabela 6.7 "Speed-up": Newton-GMRES

n	p = 2	p = 3
100	0.77350	0.56190
500	0.99090	0.96990
1000	1.10130	1.18340
2000	1.12360	1.23830

A tabela (6.8) e as figuras (6.11), (6.12) seguintes mostram a aceleração do algoritmo Broyden-paralelo em relação ao algoritmo Broyden-sequencial.

Tabela 6.8 "Speed-up": Broyden

n	p = 2	p = 3
100	0.2366	0.10080
500	0.5318	0.5289
1000	1.3008	1.10120
2000	1.1514	1.23197

6.6 Análise de complexidade da implementação paralela

Segundo os resultados obtidos em nossa implementação, uma solução eficiente depende dos valores que tenham n,p e c (dimensão da malha, número de processadores e base respectivamente). Para descrever esta situação temos desenvolvido analíticamente o tempo necessário T_{flop} para executar as operações de ponto-flutuante, e a comunicação em cada algoritmo paralelo implementado.

Devido ao particionamento de dados, o número de operações difere entre os processadores, e depende das posições relativas na malha dos processadores. A expressão para T_{flop} na implementação paralela é obtida para o processador com o maior número de operações, e determinado por $T_{flop} = \mathcal{O}_{nop}\,\mathcal{O}_{*}$, onde $\mathcal{O}_{nop}$ e $\mathcal{O}_{*}$ indicam o número de operações de ponto flutuante e o tempo por cada operação flutuante respectivamente. O número de operações para cada um dos algoritmos implementados é:

Algoritmo Newton-GMRES:

$$\begin{aligned}\mathcal{O}_{nop} &= (1+3K)\mathcal{O}_F(n,p) + (1+7K)(n/p) + K\,[\,\mathcal{O}_{GMRES}(n,p,K1) + \\ &\quad \mathcal{O}_{parab}(n,p) + \mathcal{O}_{gold}(n,p) + \mathcal{O}_{jac}(n,p)\,]\end{aligned}$$

onde

$$\begin{aligned}\mathcal{O}_{GMRES} &= 3(n/p) + \mathcal{O}_{mv}(n,p) + K1\,[\,c(\mathcal{O}_{mv}(n,p) + (n/p)] \\ &\quad (c^2+c)(n/p+2) + c\,log_2 p\,\mathcal{O}_C(1) + (\frac{n^2-n}{2}) + \\ &\quad (c+1)n/p + \mathcal{O}_{mv}(n,p)\,]\end{aligned}$$

Algoritmo Broyden:

$$\begin{aligned}\mathcal{O}_{nop} &= (K + K/c + 1)\mathcal{O}_F(n,p) + (\frac{2n+1}{p}) + K\,[\,(\frac{7n+1}{p}) + \\ &\quad (c^2 + c)n/p + log_2 p\ \mathcal{O}_C(2)\,] + [K/c]n/p\end{aligned}$$

Aqui, $\mathcal{O}_C(w) = \alpha + \beta\ w$ é o tempo de comunicação entre dois processadores, onde α indica a latência(em UT) e β é o tempo de comunicação por palavra [da Cunha 94]. Estes parâmetros são dependentes do software de comunicação e da máquina. Para o caso do Fortran 77/PVM, $\alpha = 2.70 \times 10^{-4}$s,$\beta = 8.66 \times 10^{-6}$s, e $\mathcal{O}_* = 1.7 \times 10^{-6}$ (valores típicos de rede Ethernet e estações SUN).

As figuras (6.16), (6.13), (6.14), (6.15) mostram os gráficos do "speed-up" para diferentes valores de n, c e p usando as equações correspondentes para os modelos das implementações. Pode ser visto que quando o tamanho do problema aumenta é possível resolver o sistema eficientemente para um incremento de processadores com um valor maior de c (note que c é sempre escolhido para ser consideravelmente menor que n). Isto pode permitir uma redução no tempo total de execução devido ao número menor de iterações requerido para convergência quando se usa dimensões maiores na base.

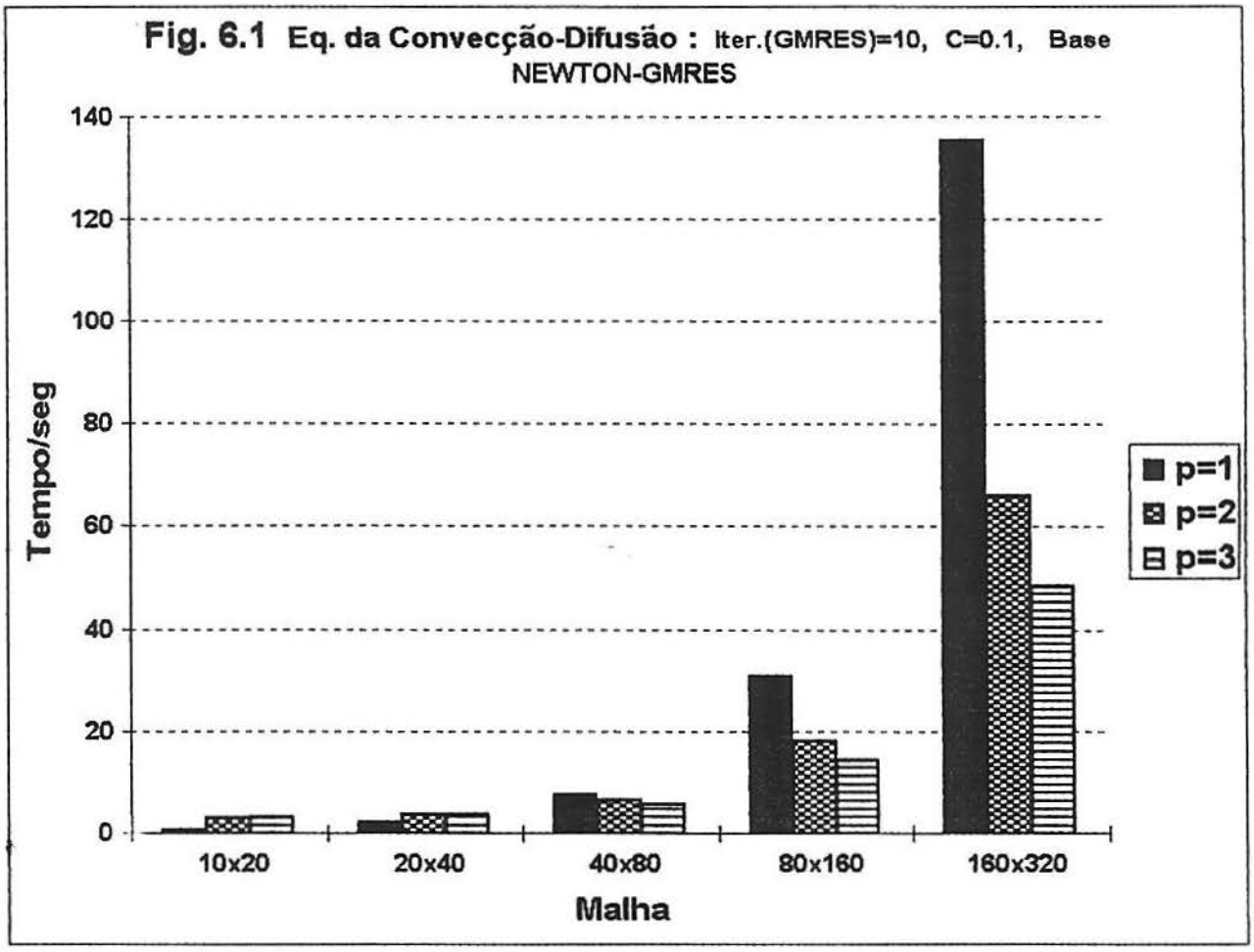


Fig. 6.1 Eq. da Convecção-Difusão : Iter.(GMRES)=10, C=0.1, Base NEWTON-GMRES
140
120
100
80
60
40
20
0
Tempo/seg
10x20
20x40
40x80
80x160
160x320
Malha
p=1
p=2
p=3

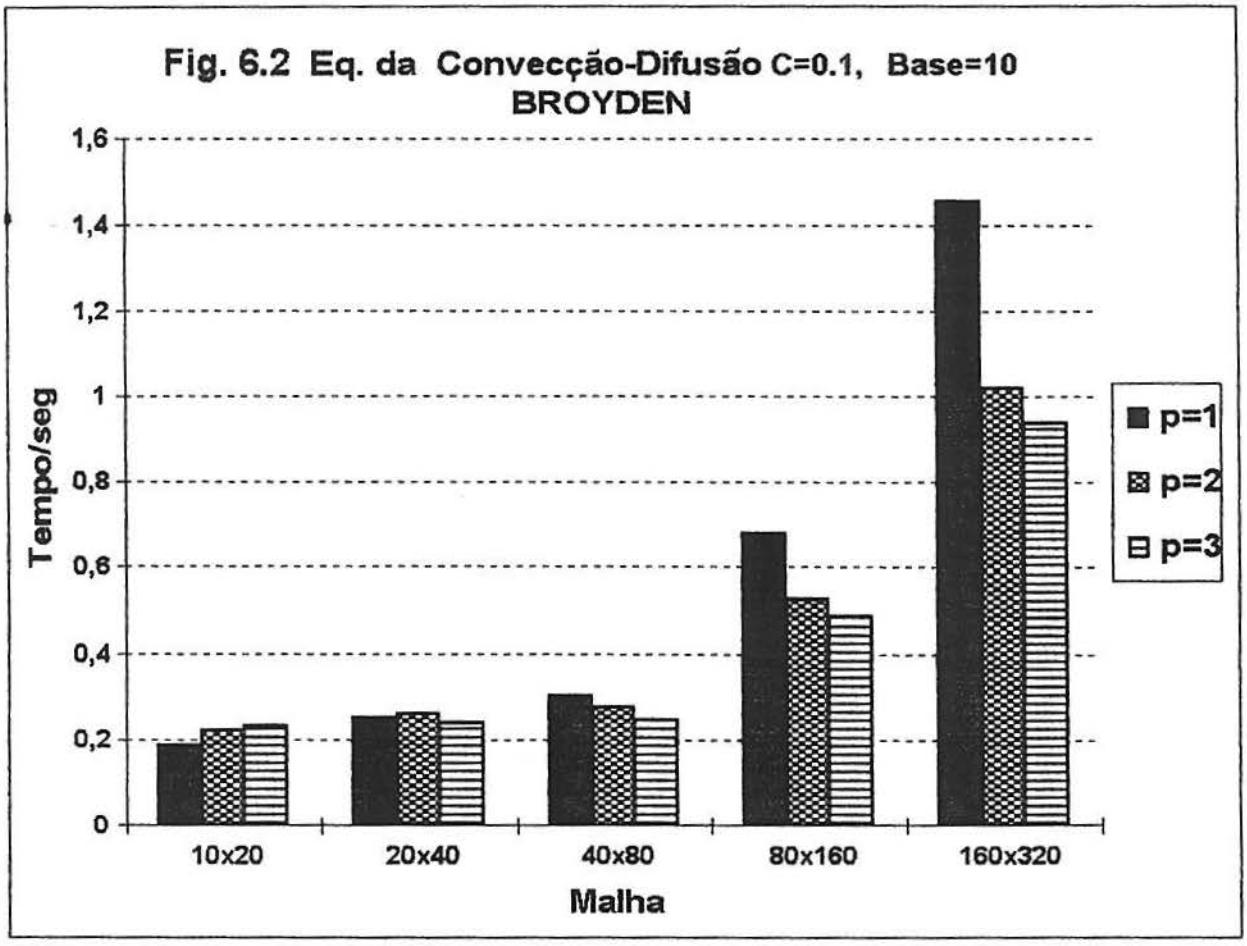
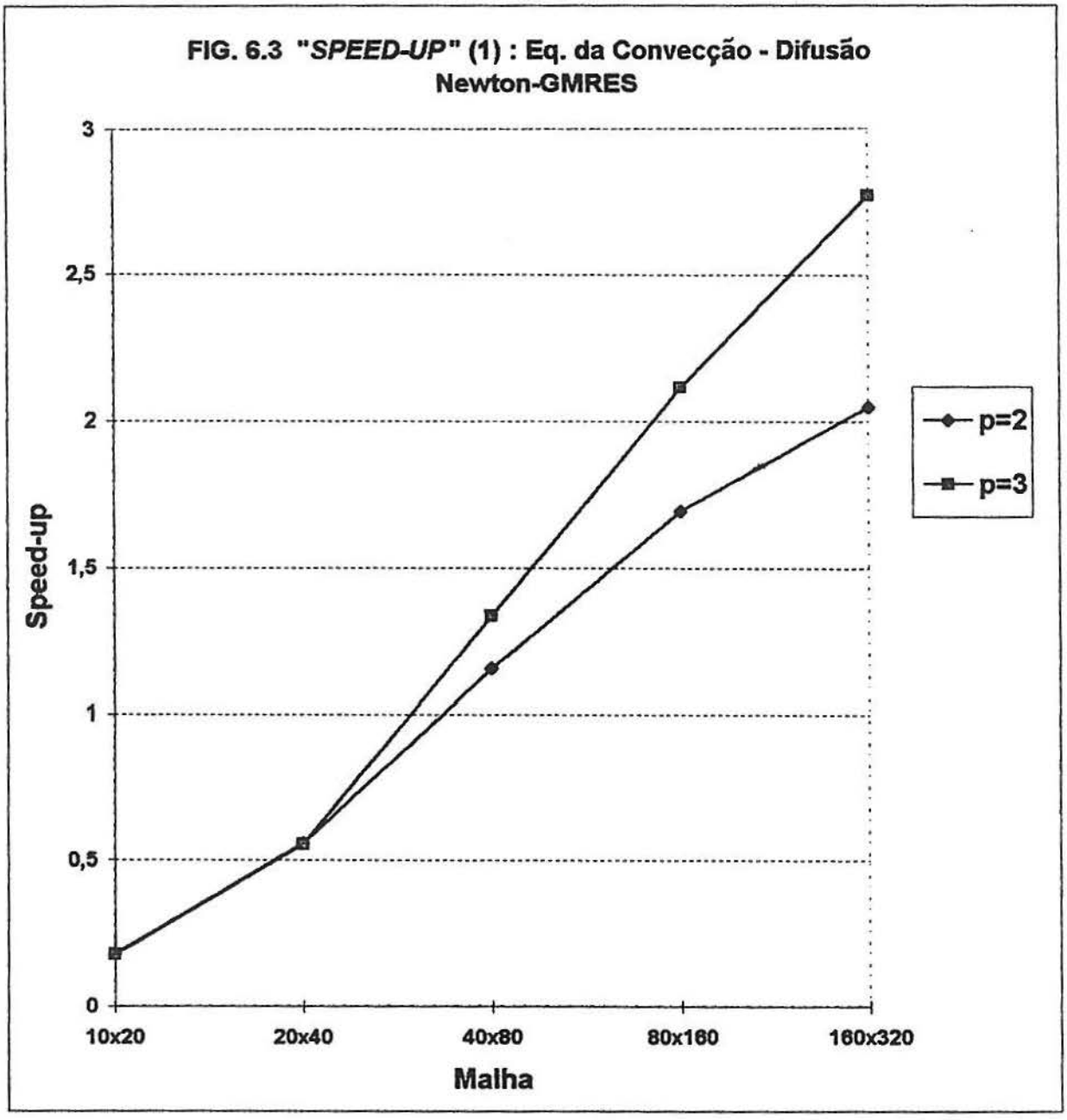


Fig. 6.2 Eq. da Convecção-Difusão C=0.1, Base=10
BROYDEN
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Tempo/seg
10x20
20x40
40x80
80x160
160x320
Malha
p=1
p=2
p=3



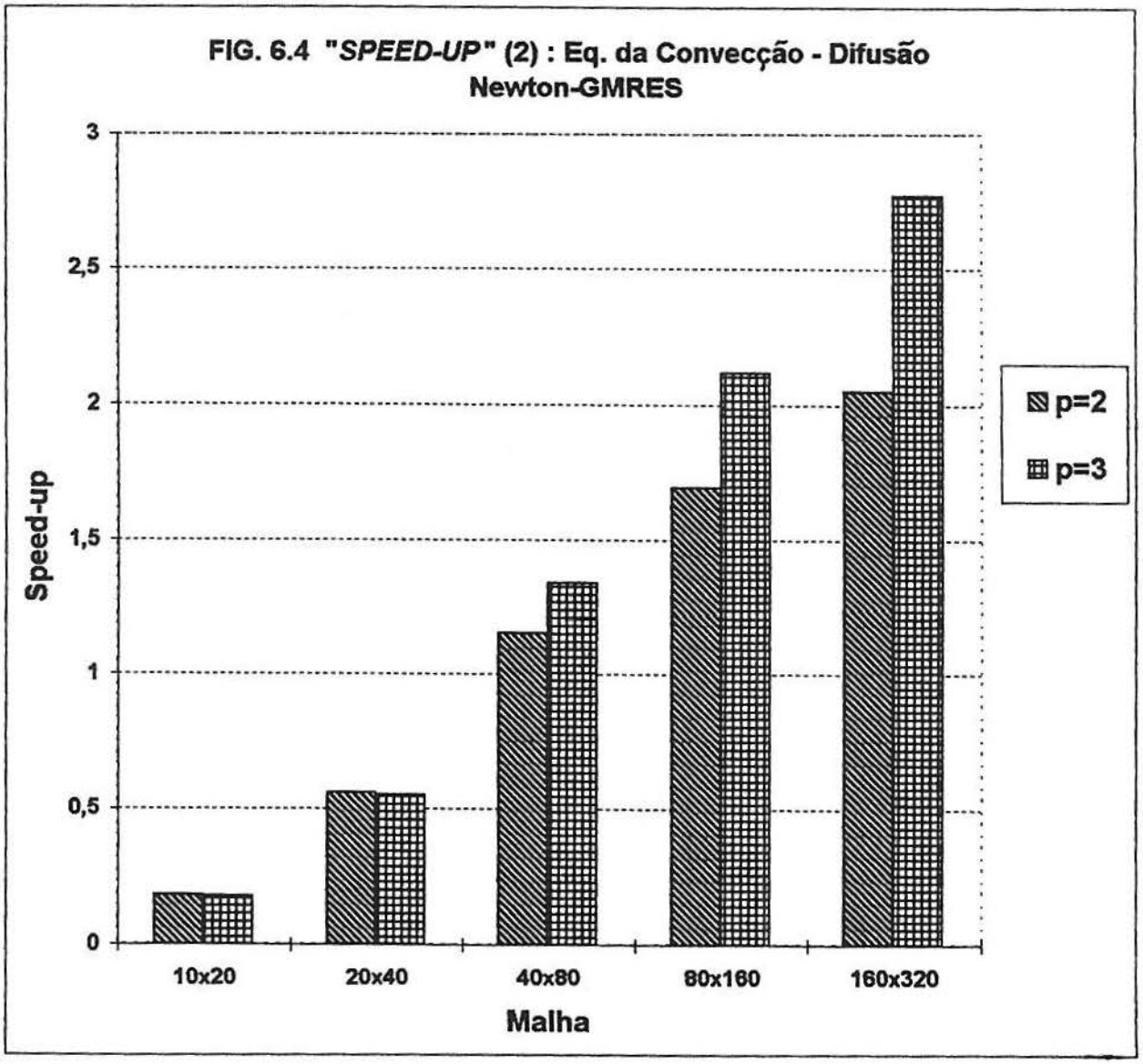


FIG. 6.4 "SPEED-UP" (2) : Eq. da Convecção - Difusão
Newton-GMRES
Speed-up
3
2,5
2
1,5
1
0,5
0
10x20
20x40
40x80
80x160
160x320
Malha
p=2
p=3

FIG. 6.5 "SPEED-UP" (1) : Eq. da Convecção-Difusão
BROYDEN

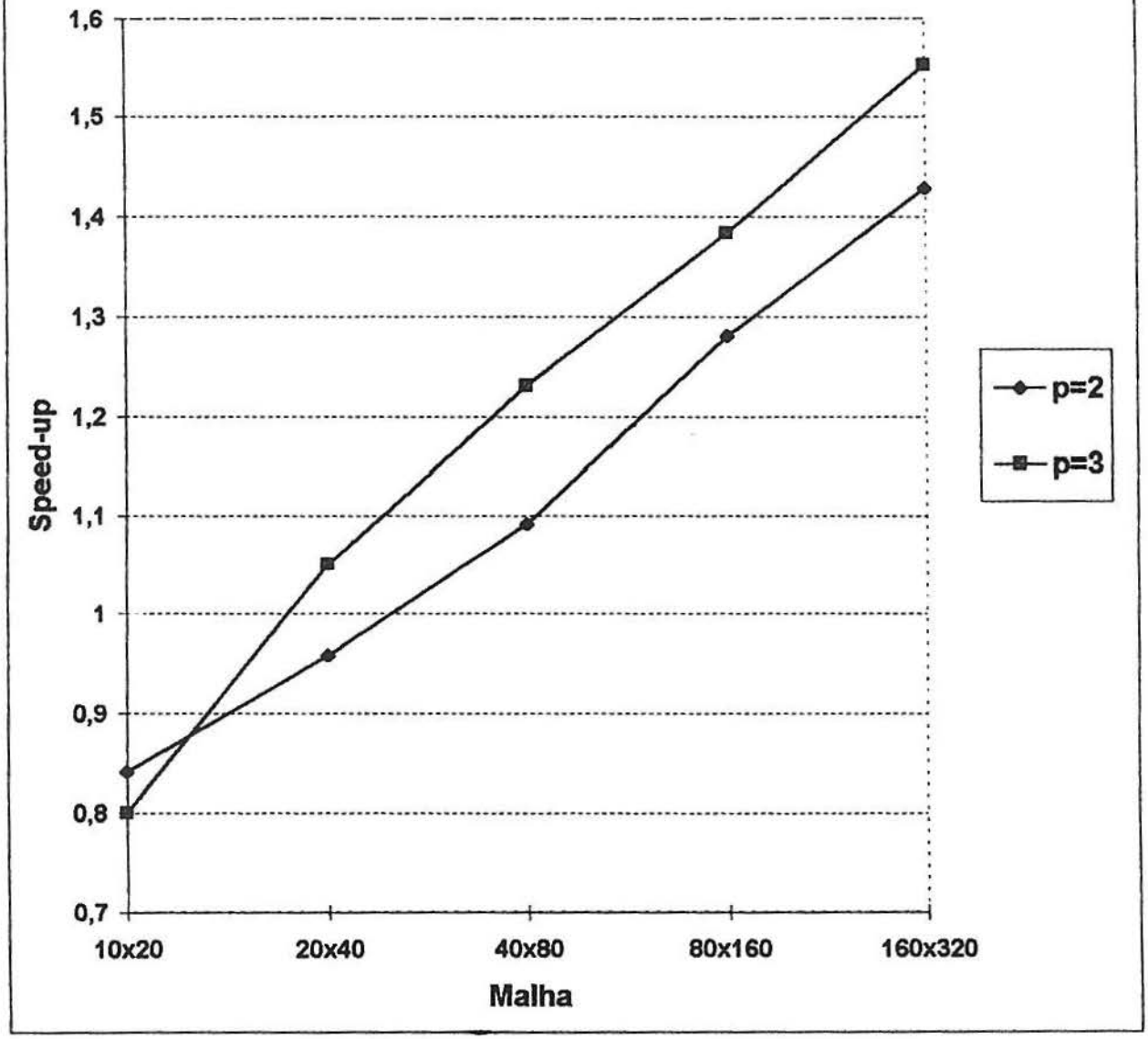
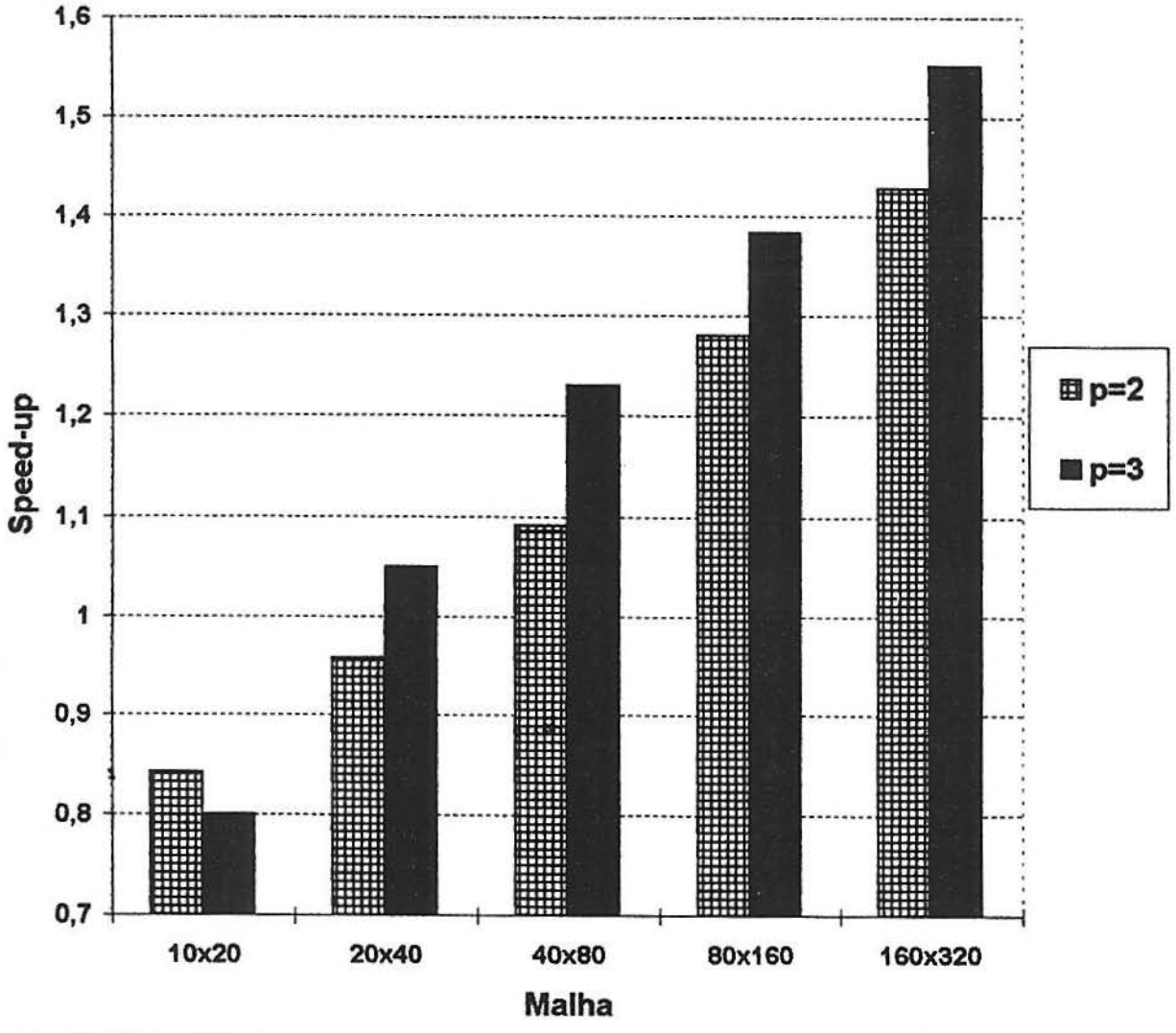


FIG. 6.6 "SPEED-UP" (2) : Eq. da Convecção-Difusão
BROYDEN



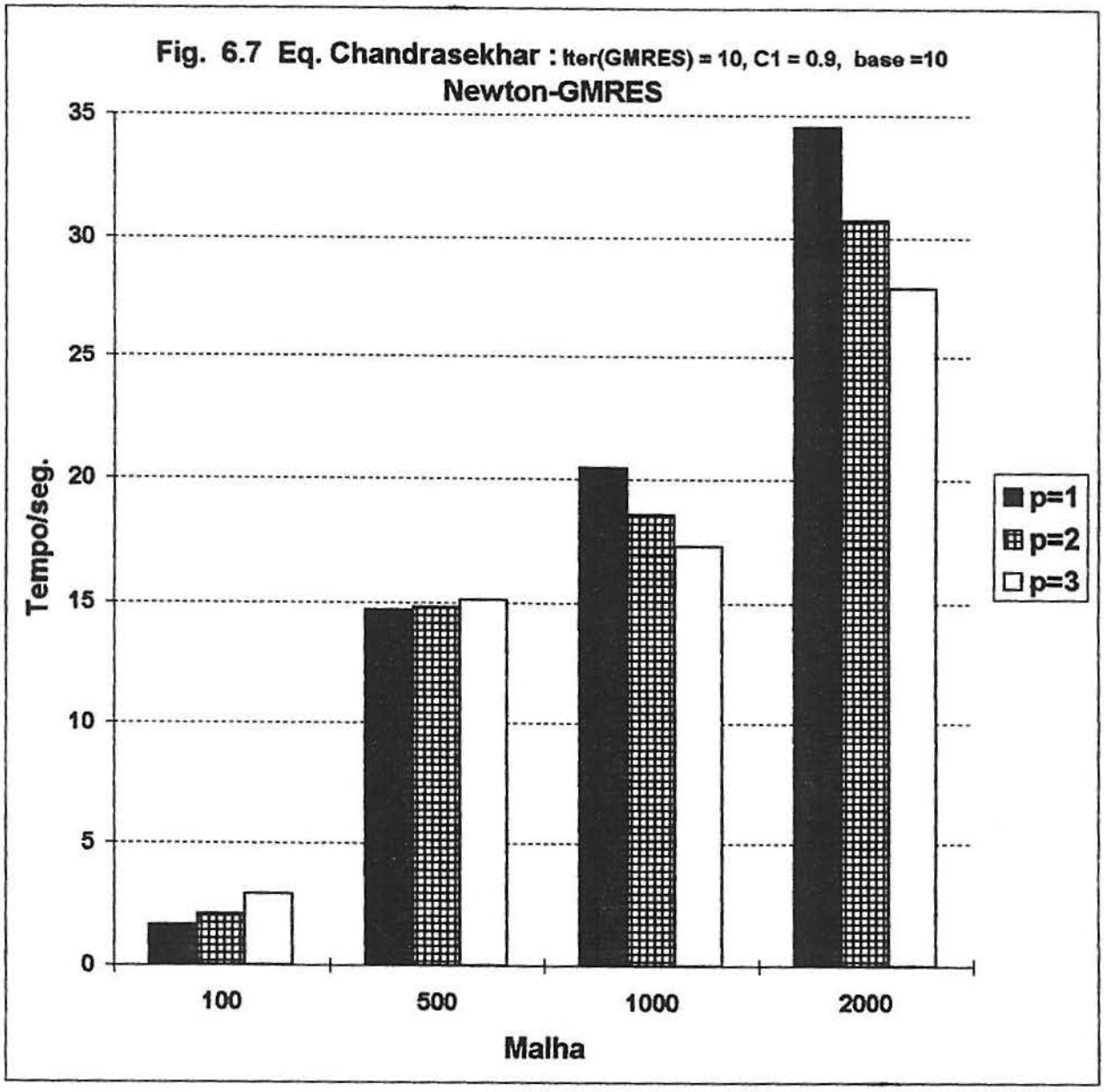


Fig. 6.7 Eq. Chandrasekhar : Iter(GMRES) = 10, C1 = 0.9, base =10
Newton-GMRES
35
30
25
20
15
10
5
0
Tempo/seg.
100
500
1000
2000
Malha
p=1
p=2
p=3

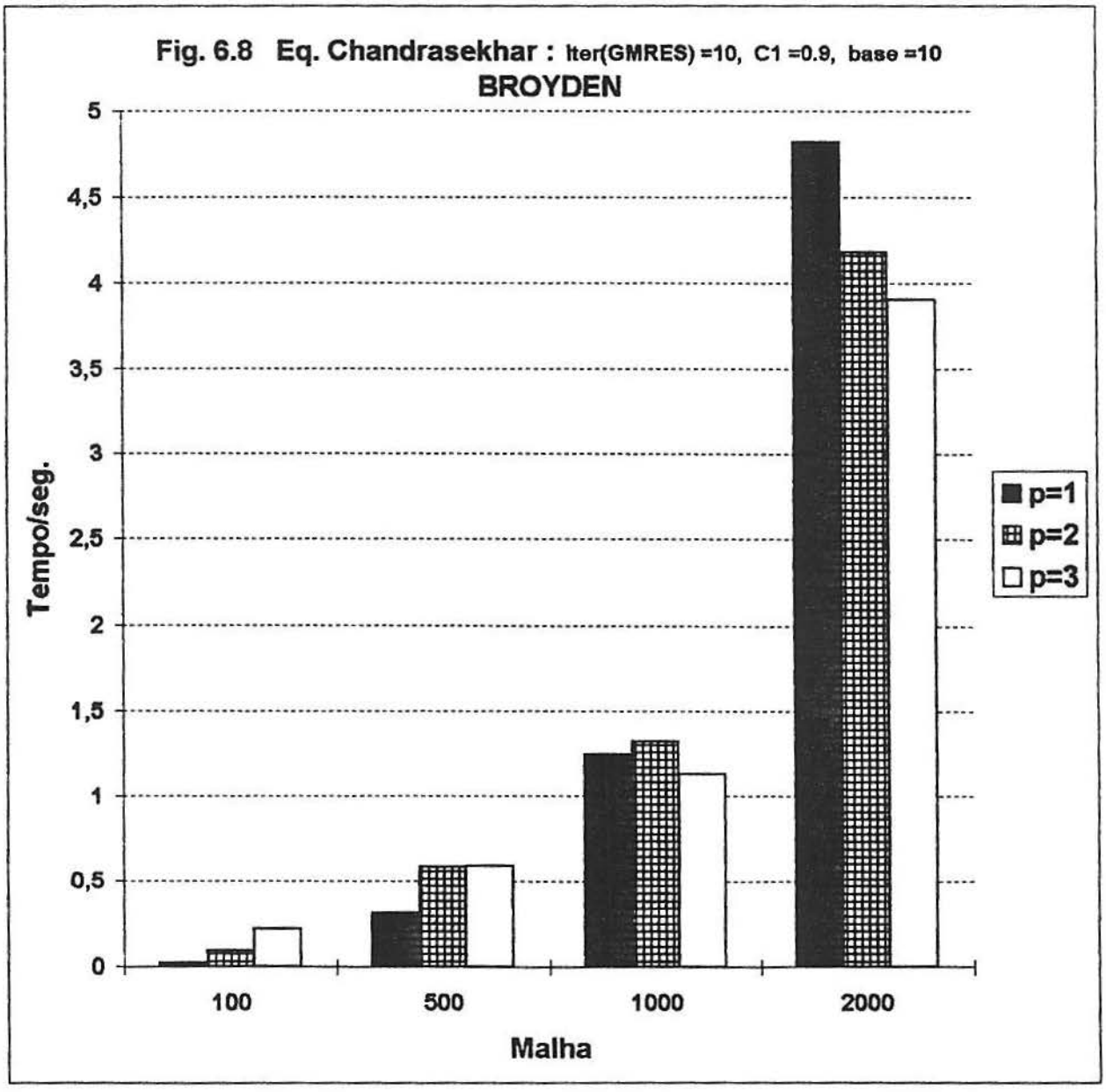
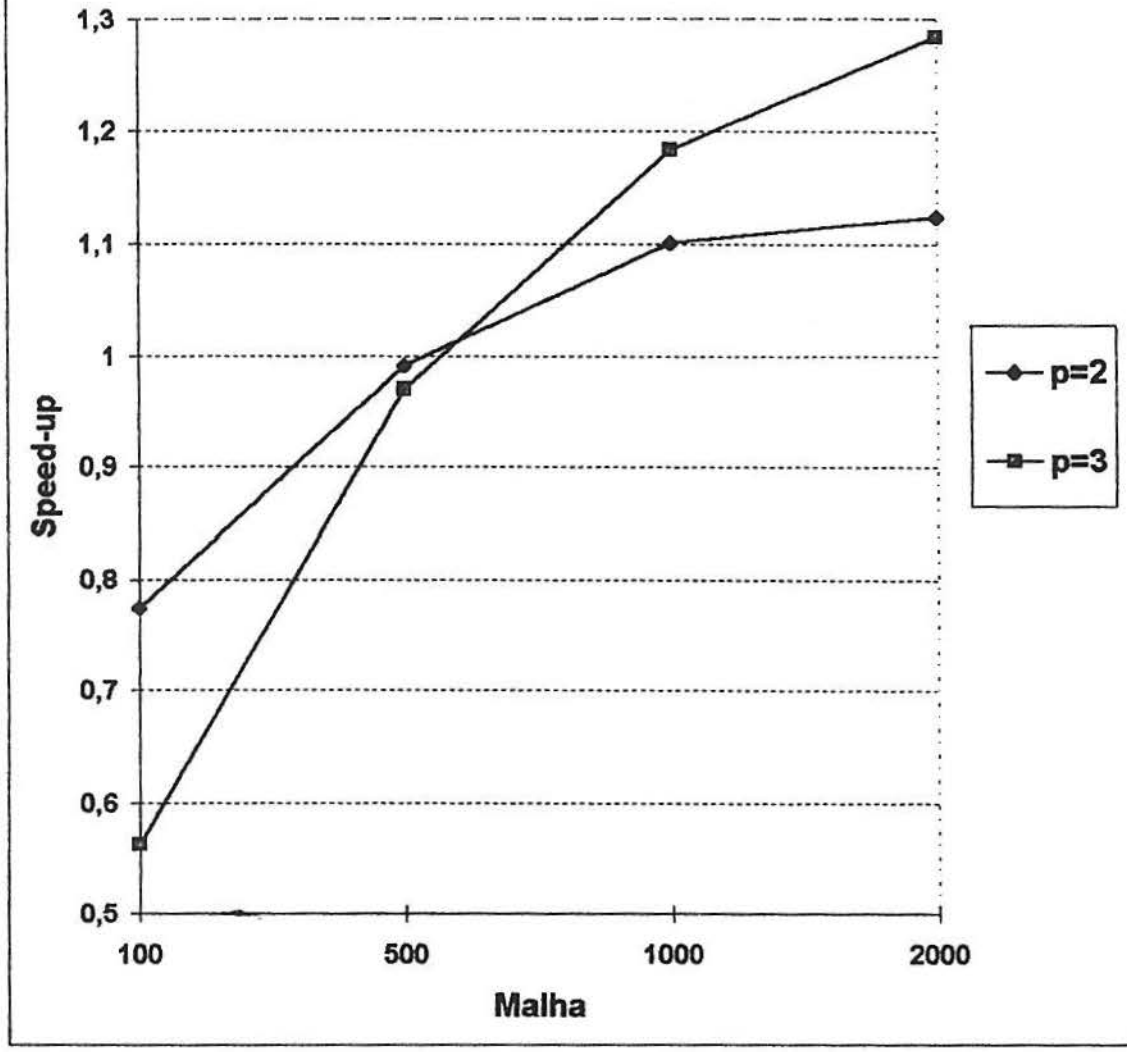


Fig. 6.9 "SPEED-UP"(1) : Eq. de Chandrasekhar Newton-GMRES



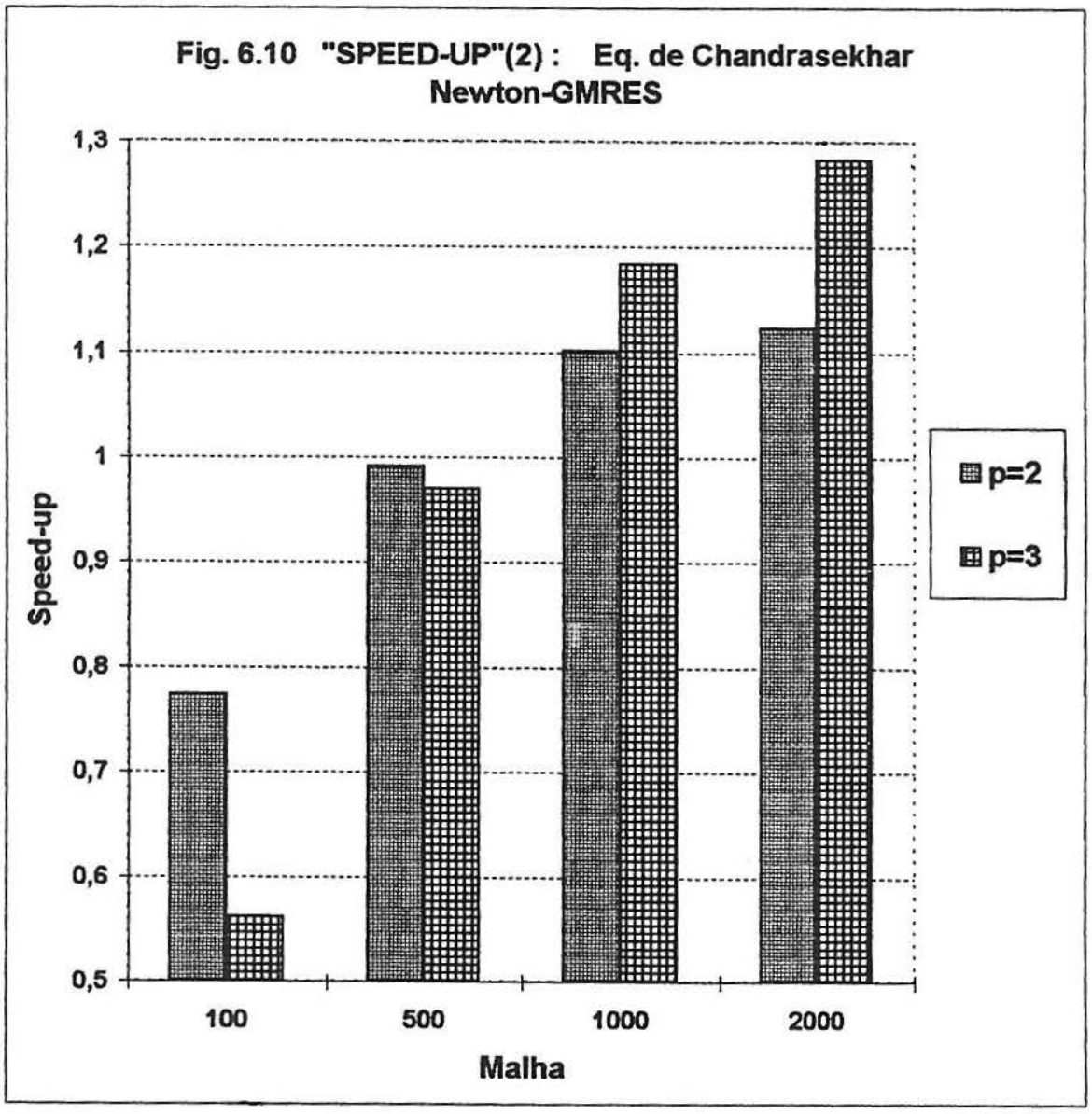


Fig. 6.10 "SPEED-UP"(2) : Eq. de Chandrasekhar
Newton-GMRES
1,3
1,2
1,1
1
0,9
0,8
0,7
0,6
0,5
Speed-up
100
500
1000
2000
Malha
p=2
p=3

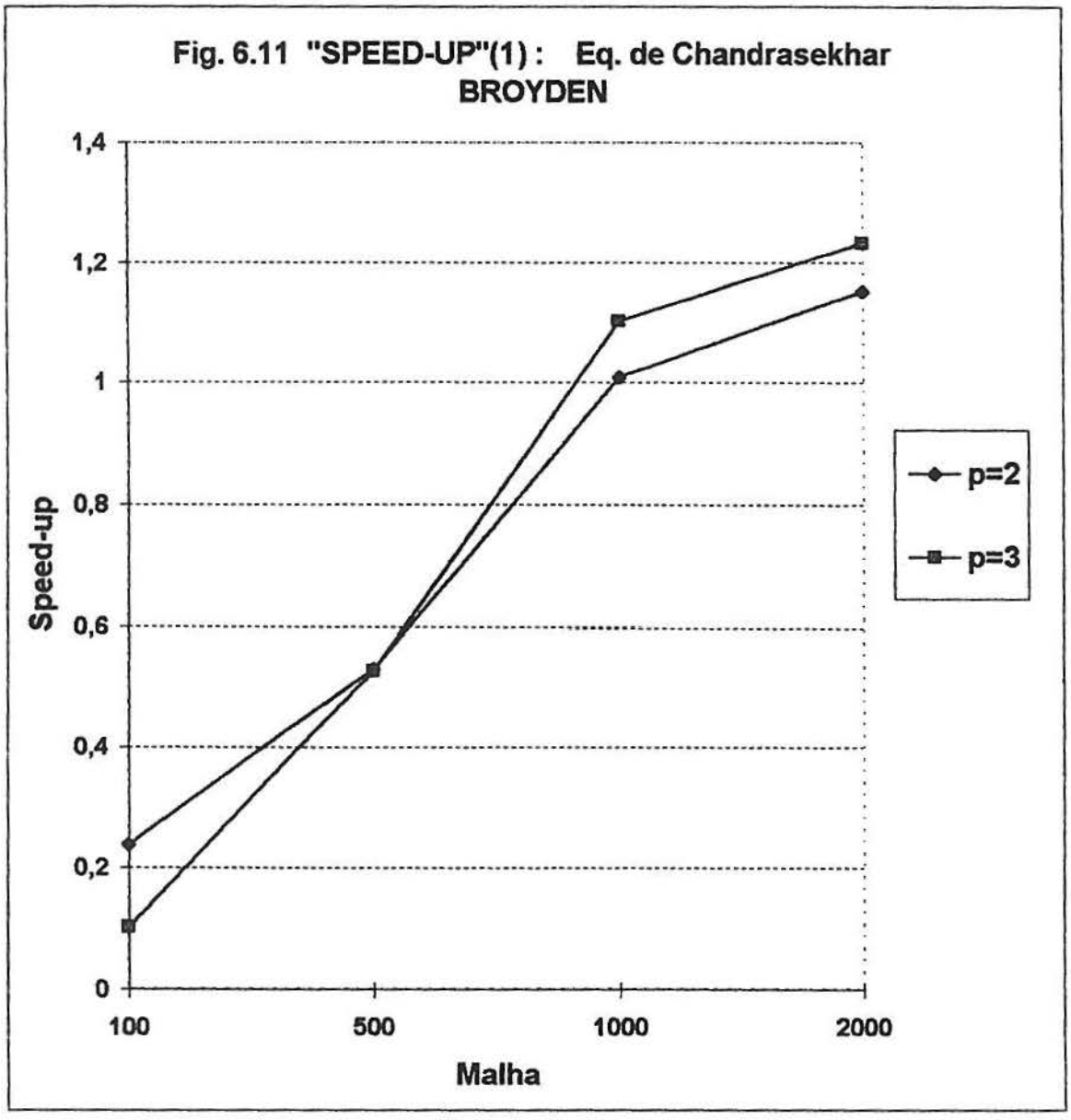
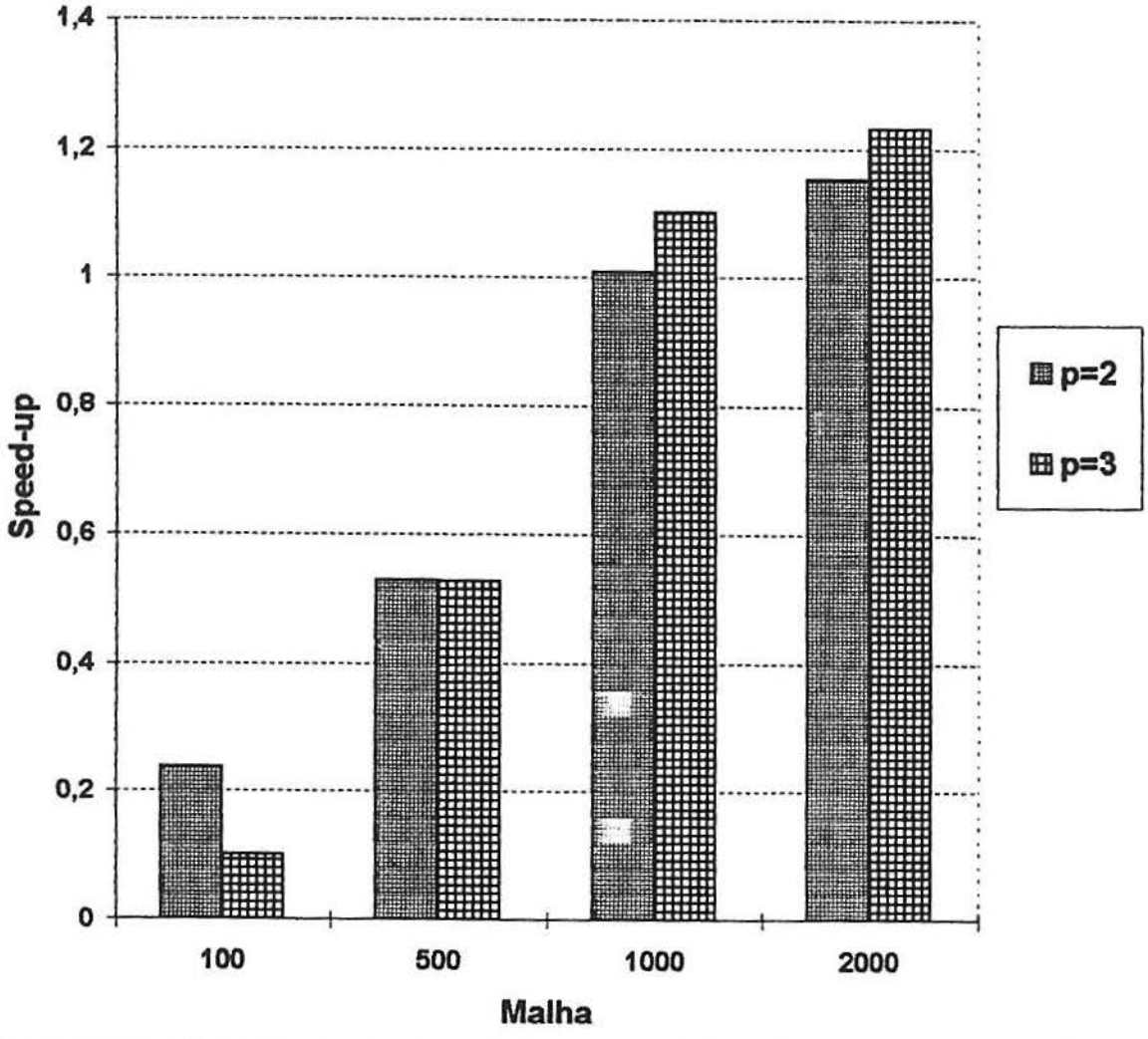


Fig. 6.11 "SPEED-UP"(1) : Eq. de Chandrasekhar
BROYDEN
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Speed-up
100
500
1000
2000
Malha
p=2
p=3

Fig. 6.12 "SPEED-UP"(2) : Eq. de Chandrasekhar BROYDEN



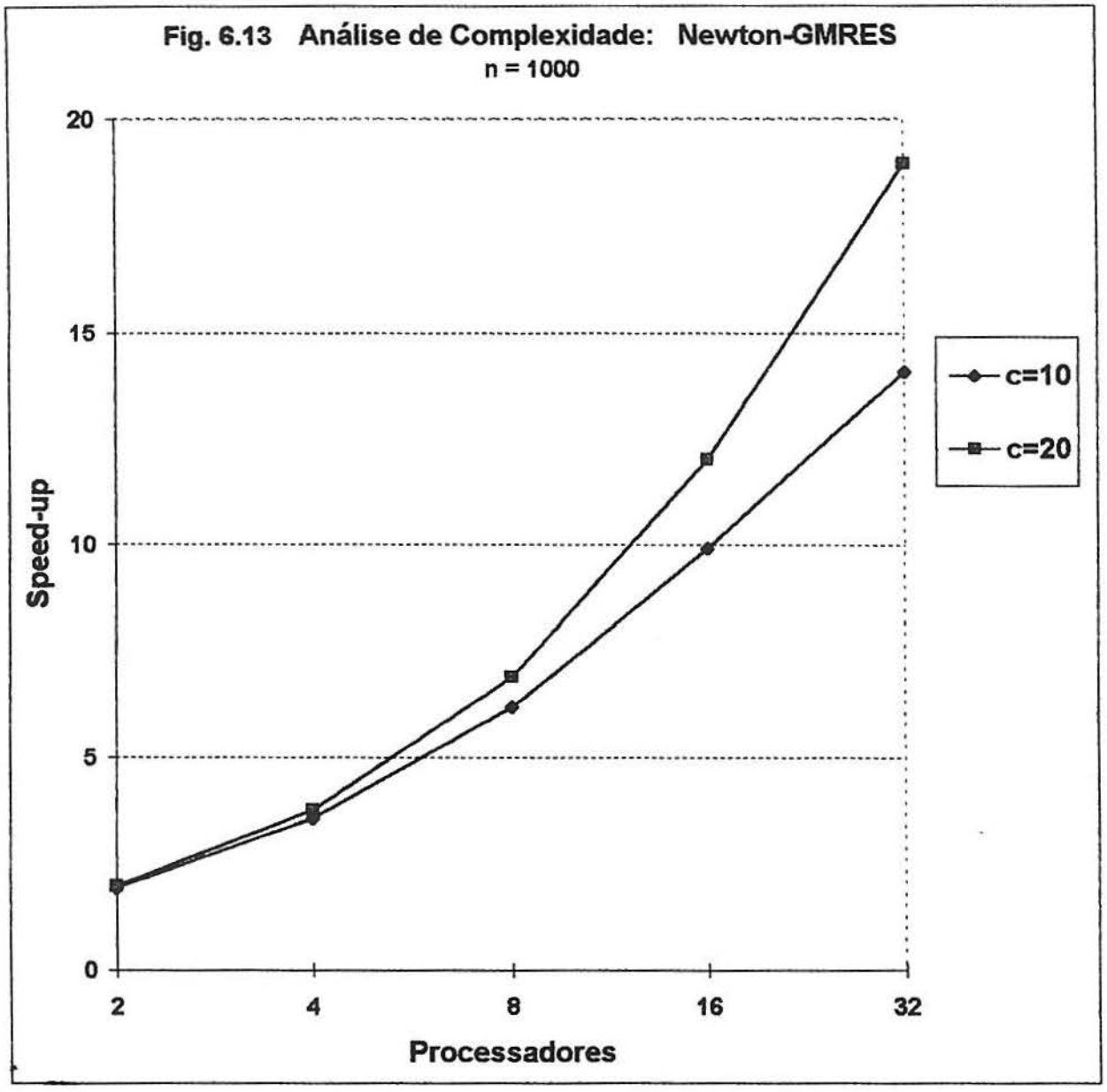


Fig. 6.13 Análise de Complexidade: Newton-GMRES
n = 1000
20
15
10
5
0
Speed-up
2
4
8
16
32
Processadores
c=10
c=20

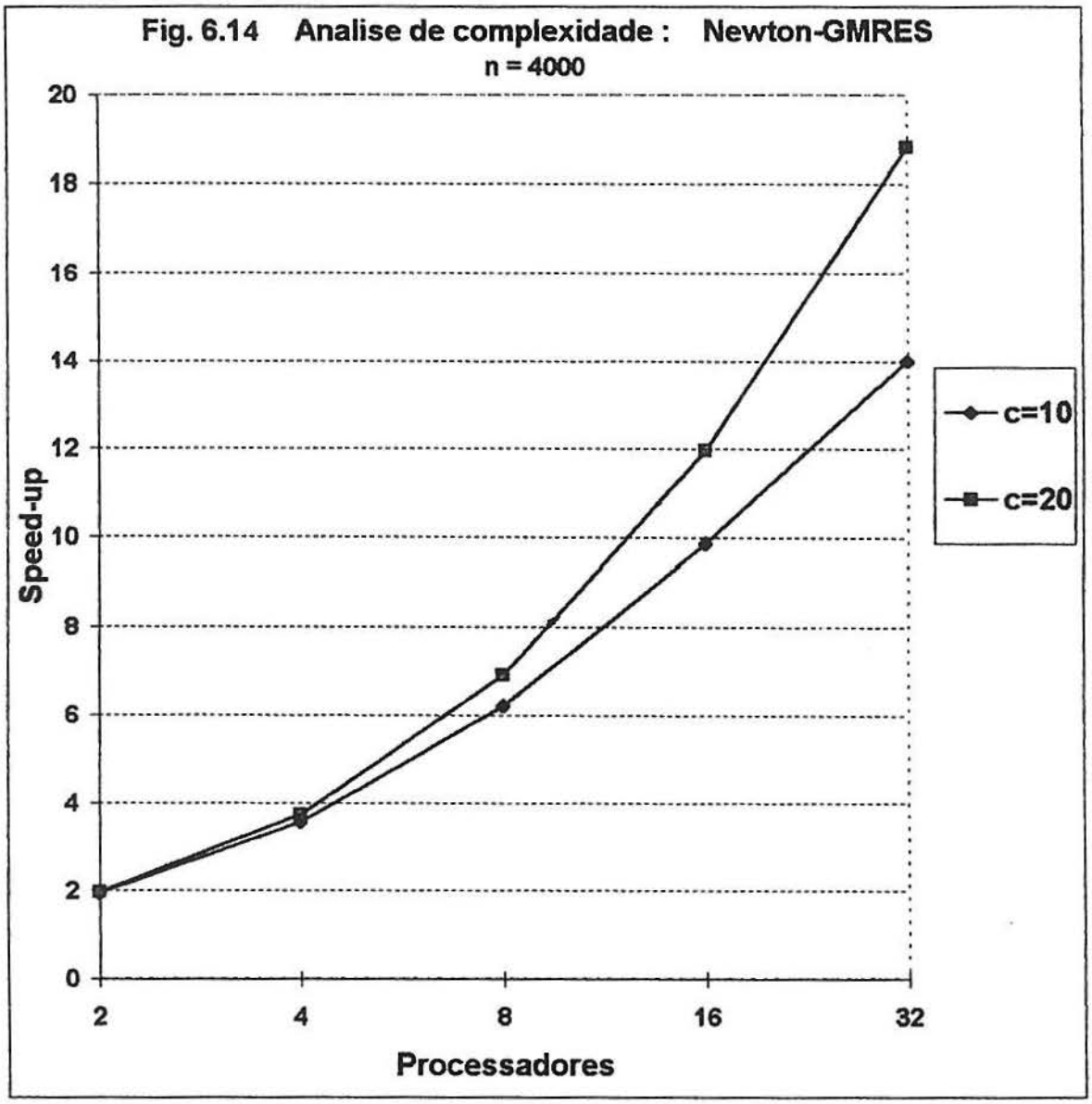


Fig. 6.14 Analise de complexidade : Newton-GMRES
n = 4000
Speed-up
20
18
16
14
12
10
8
6
4
2
0
2
4
8
16
32
Processadores
c=10
c=20

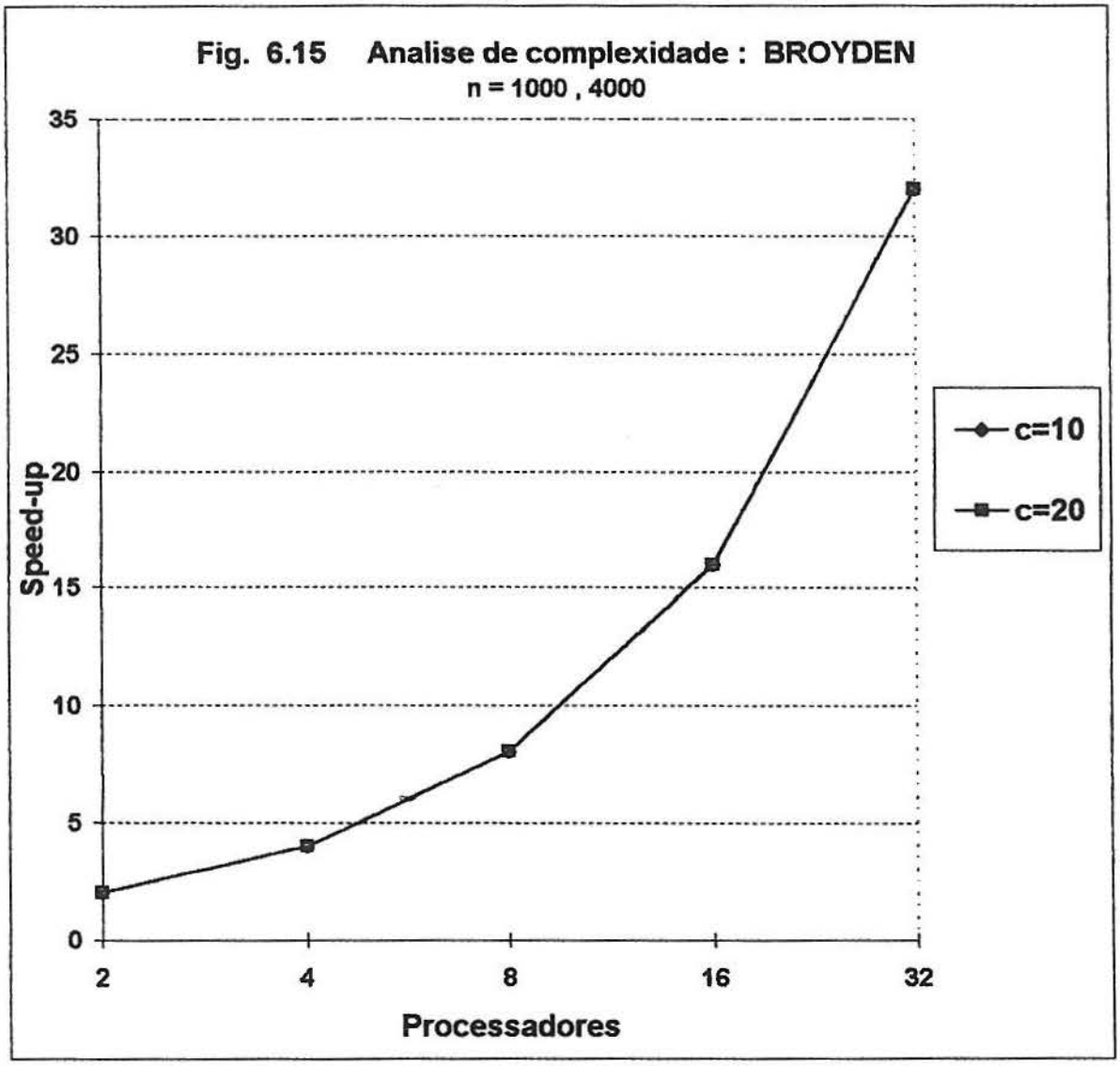
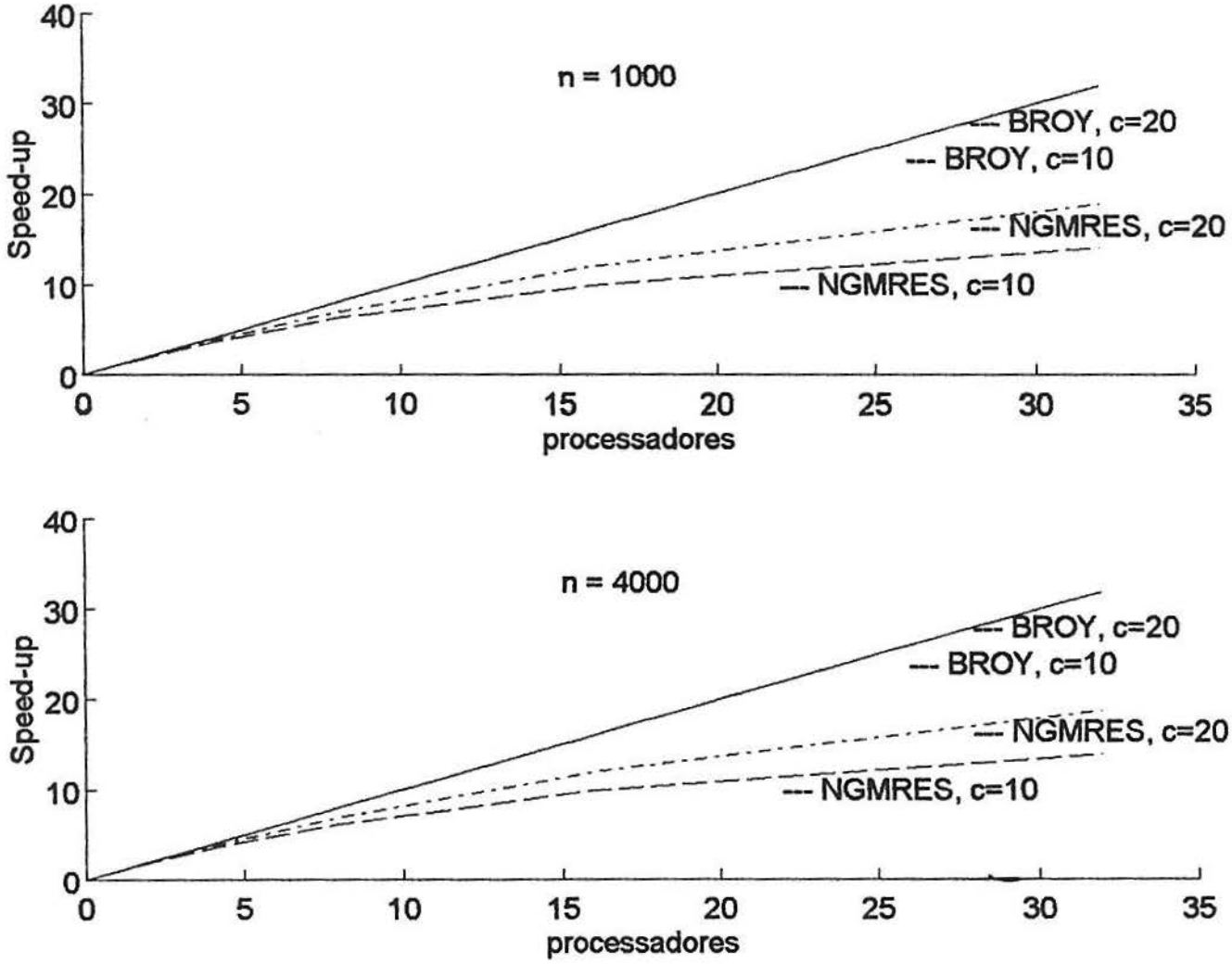


FIG. 6.16 Implementação Fortran 77/PVM



7 CONCLUSÃO E FUTUROS DESENVOLVIMENTOS

Nesta disertação estudamos a solução de equações não-lineares usando dois algoritmos iterativos do tipo quasi-Newton - *Newton-GMRES* e *Broyden* - em PVM.

Temos centrado nossos esforços em obter soluções mais rapidamente de ditas equações. Isto foi realizado através da paralelização destes algoritmos paralelos iterativos do tipo quasi-Newton para tirar vantagem das capacidades oferecidas pelo PVM e da particular estrutura que a função e seu Jacobiano podem apresentar.

As implementações mostraram ser escaláveis com respeito ao número de processadores, especialmente para problemas grandes. A principal implicação de tudo isto é que, para os algoritmos apresentados aqui, é possível usar redes de estações de trabalho ainda com altas latências de comunicação em "hardware".

Ambos os métodos estudados trabalham com a noção de "base", i.e., constrói-se uma aproximação para a solução envolvendo um mínimo de vetores (se comparado à dimensão do problema). Se a base é grande, a convergência é acelerada, exigindo no entanto, mais área de armazenagem. O custo relativo nas avaliações de funções para o método de Broyden é um para cada iteração não linear e para o Newton-GMRES, um para cada iteração externa e um para cada iteração interna. Em consequência, se a avaliação de função é muito custosa, o custo de uma solução pelo método de Broyden poderá ser muito menor do que o do método Newton-GMRES. Porém, se a memória disponível é em grande quantidade, e o número de iterações é grande, o método de Broyden poderá estar em desvantagem, pois precisará ser reinicializado muitas vezes. Nossa recomendação geral é tentar ambos.

Esperamos expandir este trabalho tanto em sua generalidade como na escolha de outros algoritmos alternativos para outros casos. Quanto a generalidade do código desenvolvido podemos fazer com que a rotina `PIMSRGMRES` seja considerada um argumento mais, o qual permitirá usar outras rotinas para resolver o sistema (5.3), especialmente em casos específicos - por exemplo, se a função for quadrática, então a rotina `PIMSCG` (que implementa o método dos Gradientes-Conjugados) será mais indicada. Além disso, nossa metodologia de implementação permite a utilização de pré-condicionadores como argumentos, o qual pode permitir a mais rápida convergência dos métodos.

Finalmente, ainda que os métodos iterativos tenham tido grande progresso nos últimos anos, estes ainda estão em sua infância, pois ainda não existe um método capaz de resolver eficientemente qualquer sistema linear. Métodos que convergem eficientemente para um sistema, podem falhar miseravelmente em outro. Os resultados apresentados neste trabalho demonstram que o mesmo é verdade no âmbito dos problemas não lineares e, portanto, gostaríamos de enfatizar que os métodos aqui estudados poderão vir a apresentar comportamentos diferentes ao serem utilizados na solução de outros problemas.

8 MANUAL DE REFERÊNCIA

Neste capítulo proporcionaremos detalhes de cada subrotina individual de nossa implementação. Cada entrada descreve o propósito, o nome e a lista de parâmetros, requerimentos de armazenagem, dependências de funções e restrições da própria rotina. Vetores e escalares são denotados com letras minúsculas e matrizes por letras maiúsculas.

A.1 Descrição de parâmetros

Os parâmetros usados em uma rotina dos métodos iterativos são

IPAR (entrada)

Elemento	Descrição
1	Número de colunas da Matriz A
2	Número de linhas da Matriz A
3	Tamanho do bloco de particionamento
4	Número de elementos armazenados localmente
5	Parâmetro de reinicialização (BASE)
6	Número de processadores
7	Identificador do processo
8	Pré-condicionamento
	0: Pré-condicionamento nenhum
	1: Pré-condicionamento pela esquerda
	2: Pré-condicionamento pela direita
	3: Pré-condicionamento simétrico

IPAR (entrada)

9	Critério de parada
10	Número máximo de iterações permitido
11	Número de iterações
12	Estado de saída
13	Se IPAR(12) é -2 ou -3, dá o número do passo no algoritmo onde um "break-down" tem ocorrido

SPAR (entrada)

Elemento	Descrição
1	Constante de Lipschitz (γ)
2	Valor máximo da constante (η)
3	Tolerância relativa (τ_r)
4	Tolerância absoluta (τ_a)

A.2 Rotinas Externas

Propósito: Para calcular o produto matriz-vetor, o valor do segundo membro do sistema de equações, soma global de um vetor, o valor da função a matriz dos coeficientes dos termos convectivos e difusivos, a matriz Jacobiana e monitorar o progresso das iterações.

Nota: A matriz dos coeficientes A, B e J; a constante C da equação diferencial e o segundo membro do sistema podem ser disponíveis para FEVAL, MATVEC usando blocos COMMON.

Sinopse:

Produto matriz-vetor v=Au

```
SUBROTINA MATVEC(U,V,IPAR)
```

precisão U(*), V(*)

INTEGER IPAR(*)

Parâmetro	Tipo
U	ENTRADA
V	SAÍDA
IPAR	ENTRADA

Valor da função v=f(u)

`SUBROTINA FEVAL(U,V,IPAR)`

precisão U(*), V(*)

INTEGER IPAR(*)

Parâmetro	Tipo
U	ENTRADA
V	SAÍDA
IPAR	ENTRADA

Segundo membro do sistema: F

`SUBROTINA F2MEM(INÍCIO,M,N,H1,H2,C,F)`

precisão F(*)

INTEGER M,N,INÍCIO

REAL C,H1,H2

Parâmetro	Tipo
M,N,INICIO	ENTRADA
C,H1,H2	ENTRADA
F	SAÍDA

Matriz Jacobiana L

`SUBROTINA JACPA(U,IPAR)`

precisão U(*)

INTEGER IPAR(*)

Parâmetro	Tipo
U	ENTRADA
IPAR	ENTRADA

Soma Paralela

```
SUBROTINA PSSUM(ISIZE,X)
```

precisão X(*)

INTEGER ISIZE(*)

Parâmetro	Tipo
ISIZE	ENTRADA
X	ENTRADA/SAÍDA

Vetor Norma paralelo

```
SUBROTINA PNRM(LOCLEN,U)
```

precisão U(*)

INTEGER LOCLEN(*)

Parâmetro	Tipo
LOCLEN	ENTRADA
U	ENTRADA

Rotina de monitoramento

```
SUBROTINA PROGRESS(LOCLEN,ITNO,NORM,
          X,RES,TRUERES)
```

precisão NORM,X(*),RES(*),TRUERES(*)

INTEGER LOCLEN,ITNO

Parâmetro	Tipo
LOCLEN	ENTRADA
ITNO	ENTRADA
NORM	ENTRADA
X	ENTRADA
RES	ENTRADA
TRUERES	ENTRADA

Notas:

1. Substituir *precisão* por REAL, DOUBLE PRECISION, COMPLEX, DOUBLE COMPLEX.

2. Na rotina de monitoramento PROGRESS, o arrango TRUERES contém o valor do resíduo verdadeiro $r_k = b - Ax_k$ se e somente se IPAR(9) é 1, 2 ou 3.

A.3 Newtar

Propósito: Resolver o sistema $AX = b$ usando o método Newton-GMRES

Sinopse:

NEWTAR(U,WRK,IPAR,SPAR,MATVEC,FEVAL,JACPA,PRECON, PRECON,PSSUM,PSNRM2,PROGRESS)

Requerimento de armazenagem:

Parâmetro	No. Palavras
U	IPAR(4)
WRK	9*IPAR(4) +(4 + IPAR(5))*IPAR(4))
IPAR	13
SPAR	6

Dependência de funções :

BLAS: SAXPY, SCOPY, SINIT, SSCAL

LIBPIM: PIMSRGMRES

LIBNPIM: GOLDEN, PARAB3P

A.4 BROYDEN

Propósito: Resolver o sistema $AX = b$ usando o método de Broyden para aproximar o Jacobiano

Sinopse:

BROYDEN(U,FEVAL,IPAR,SPAR,WRK,PSSUM,PSNRM2)

Requerimento de armazenagem:

Parâmetro	No. Palavras
U	IPAR(4)
WRK	3*IPAR(4) + NMAX*IPAR(4)
IPAR	13
SPAR	6

Dependência de funções:

BLAS: SAXPY, SCOPY, SSCAL, SDOT

LAPACK: SLAMCH

BIBLIOGRAFIA

[Ashby 90] ASHBY S.F., MANTEUFFEL, T. A., e SAYLOR P.E. A Taxonomy for Conjugate Gradient Methods. *SIAM Journal of Numerical Analysis*, 27, 1990, pp.1542-1568.

[Brenan 89] BRENAN K., CAMPBELL J. e PETZOLD L. *The Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*. Elsevier, New Tork, 1989.

[Brown 87] BROWN, P.N. A Local Convergence Theory for Combined Inexact-Newton/Finite-Diference Projection Methods. *SIAM J. Numer. Anal., 24* 1987, pp.407-434.

[Brown 89] BROWN P. e HINDMARSH, A. Reduced Storage Matrix Methods in Stiff ODE Systems. *J. Appl. Math. Comput., 1989, 31* pp.40-91.

[Brown 94] BROWN, P. e SAAD, Y. Convergence Theory of Nonlinear Newton-Krylov Algorithms. *SIAM J. Optim.*, 4, 1994, pp. 297-330.

[Broyden 65] BROYDEN, C. G. *A class of Methods for Solving Nonlinear Simultaneous Equations.*Math. Comp., V. 19, 1965, pp. 577-593.

[Busbrigde 60] BUSBRIGDE I. W. *The Matematics of Radiative Transfer.* Cambrigde Tracts, No. 50, Cambrigde Univ. Press, Cambrigde, 1960.

[Concus 76] CONCUS P., GOLUB G.H., e O'LEARY D.P. *A Generalized Conjugate Gradient Method for the numerical solution of elliptic partial Differential Equations* chapter V. Sparse Matrix Computations. Academic Press, New York, 1976, pp. 309-332.

[Craig 55] CRAIG E.J. The N-step Iteration Procedures. *Journal of Mathematical Physics*, Vol. 34, 1955, pp. 64-73.

[da Cunha 92] da CUNHA R. D. *A Study on Iterative Methods for the Solution of Systems of Linear Equations on Transputer Networks*. PhD thesis. Computer Science Dept., Kent University, Canterbury,1992.

[da Cunha 95] da CUNHA R.D. e HOPKINS T.R. PIM 2.0 - the Parallel Iterative Methods Package for Systems of Linear Equations User'Guide - Fortran 77 version. TR95-11, Curso de Pós-Graduação em Matemática Aplicada , UFRGS - Porto Alegre, Brasil, June 1995.

[da Cunha 94] da CUNHA R.D. e HOPKINS T.R. A Parallel Implementation of the Restarted GMRES Iterative Method for Nonsymetric Systems of Linear Equations. *Advances in Computational Mathematics*, 2(3):261-277, April 1994. Also as TR-7-93, Computing Laboratory, University of Kent at Canterbury.

[Demmel 93] DEMMEL J., HEATH M. e VAN DER VORST H. *Parallel Linear Algebra*, in Acta Numerica, vol. 2, Cambrigde Press, New York, 1993.

[Dennis 83] DENNIS, J. E. e SCHNABEL R. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall Series in Comput. Math., Prentice-Hall, NJ, 1983.

[Dembo 82] DEMBO R., EISENSTAT T., e STEIHAUG T. Inexact Newton Methods. *SIAM J. Numer. Anal.*, 19, 1982, pp. 400-408.

[Deuflhard 90] DEUFLHARD P., FREUND W. e WALTER A. Fast Secant Methods for the Iterative Solution of Large Nonsymmetric Linear Systems Impact of Computing in Science and Engineering, 2, 1990, pp.244-276.

[Dongarra 88] DONGARRA J.J., DU CROZ J., HAMMARLING S. e HANSON R.J. Um Extended Set of FORTRAN Basic Linear Algebra Subprogramas. *ACM Transactions on Mathematical Software*, 1988. 14(1):1-17.

[Eisenstat 94] EISENSTAT S.C. e WALKER H.F., *Chossing the Forcing Terms in an Inexact Newton method*. Tech. Report 6/94/75, Mathematics and Statistic Department, Utah State University, June, 1994.

[Engelman 81] ENGELMAN M., STRANG G., e BATHE K.J. The Application of Quasi-Newton Methods in Fluid Mechanics.*Internat. J. Numer. Methods Engrg.*, 17, 1981, pp. 707-718.

[Elman 82] ELMAN H.C. *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations*, PhD. thesis, Computer Science Dept., Yale University, 1982.

[Gear 71] GEAR C. W. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-hall, Englewood Cliffs, NJ, 1971.

[Geist 87] GEIST, G.A., HEATHYAN, M. T., e NG, E. *Parallel Algorithms for Matrix Computations*. The Characteristics of Parallel Algorithms. MIT Press, Cambrigde, Massachusetts, 1987, pp. 233-251.

[Gerber 81] GERBER, R.R., e LUK F.T. A Generalized Broyden's Method for Solving Simultaneous Linear Equations Methods for Solving Nonlinear Simultaneous Equations.*SIAM J. Numer. Anal.*, 18, 1981, pp. 882-890.

[Glowinski 85] GLOWINSKY R., KELLER, H., e REINHART L. Continuation-Conjugate Gradient Methods for the Least Squares Solution of Nonlinear Boundary Value Problems. *SIAM J. Sci. Stat. Comp.*, 6, 1985, pp.793-832.

[Golub 89] GOLUB G.H., e O'LEARY D. P. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[Greenbaum 96] GREENBAUM A., VLASTIMIL P. e ZDENEK S. Any Nonincreasing Convergence Curve is possible for GMRES. *SIAM J. Matrix Anal. and Appl.*, Vol. 17, No. 3, July 1996, pp. 465-469.

[Hestenes 52] HESTENES M. R. e STIEFEL E.L. Method of Conjugate Gradients for Solving Linear Systems. *J. of Research National Bureau of Standards*, 49, 1952, pp.435-498.

[Keller 70] KELLER, H. Newton's Method under Mild Differentiability Conditions. *J. Comput. Sys.Sci, 4* , 1970, pp.15-28.

[Kelley 95] KELLEY, C. T. *Iteratives Methods for Linear and Nonlinear Equations* . SIAM. Philadelphia, 1995.

[McCormick 89] McCORMICK S. *Adaptive Procedure for Estimating Parameters for the Nonsymetric Tchebychev Iteration*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[Meurant 91] MEURANT, G. The future of Scientific Computing on Parallel Computers. In M. Durand and F.El Dabagh, editor, *Invited lectures of the 2nd Symposium on High-Perfomance Computing*, Montpellier, 1991.

[Mullikin 68] MULLIKIN, S. W. Some Probabilty Distributions for Neutron Transport in a Half Space. *J. Appl. Probab.*, 5 (1968), pp. 357-374.

[Ortega 70] ORTEGA, J.M. e RHEINBOLDT, W.C. *Iterative Solution of Nonlinear Equations in Several Variables*. New York, Academic Press 1970.

[PVM97] GEIST A., BEGUELIN, A., DONGARRA J. e Others. *PVM User's Guide and Reference Manual*. Prepared by the Oak Ridge National Laboratory, Oak Ridge, Tennesssee 37831, 1994.

[Saad 86] SAAD, Y. e SCHULTZ, M. GMRES a Generalized Minimal Residual Algorithm for Solving Nonsymetric Linear Systems. *SIAM J. Sci. Statist. Comput.*, 7 1986, pp.856-869.

[Sherman 49] SHERMAN J. e MORRINSON W.J. *Adjustment of an Inverse Matrix corresponding to changes in the Elements of a given column or a given row of the original matrix* (abstract). Ann. Math. Statist., 20, 1949, pp.621.

[Sonneveld 89] SONNEVELD, P. CGS, a Fast Lanczos-Type Solver for Nonsymmetric Linear Systems. for Structures under Seismic Excitations. *SIAM Journal of Scientific and Statistical Computing*, 10, 1989, pp. 36-52.

[Van der Vorst 92] VAN DER VORST. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal of Scientific and Statistical Computing*, 13, 1092, pp. 631-644. Also as Report No. 90-50, Mathematical Institute, University of Utrecht.

[Walker 88] WALKER, H.F. Implementation of the GMRES Method using Householder Transformations. *SIAM Sci. Statist. Comp.*, 9, 1989, pp. 152-163.

[Walker 90] WALKER H.F. Newton-like Methods for Undetermined Systems. In L. Allgower and K. Georg, editor. *Computational Solution of Nonlinear Systems of Equations*. Invited lectures in Applied Mathematics, vol. 26, SIAM-AMS,1990.

[Young 80] YOUNG, D. M. e JEA K.C. Generalized Conjugate-Gradient Acceleration of Non-symmetrizable Iterative Methods. *Linear Algebra and its Applications*, 34, 1980, pp. 159-194.

[Ypma 83] YPMA T.J. The Effect of Rounding Errors on Newton-like Methods. *IMA J. Numer. Anal.*, 3, 1983, pp. 109-118.