

3 SOLUÇÃO NUMÉRICA DA ANÁLISE LIMITE

O problema matemático resultante da aplicação de teoria da Análise Limite (AL) associado ao Método dos Elementos Finitos (MEF) Equações (2.28) e (2.30), pode ser representado pelas Equações 3.1-3.3. Estas equações configuram um problema de otimização e que pode ser resolvido usando técnicas de otimização.

$$\text{Max } f(\alpha) = \alpha \quad (3.1)$$

$$\text{s.t. } \mathbf{g}(\mathbf{x}) \leq 0 \quad (3.2)$$

$$\mathbf{B}\mathbf{x} - \mathbf{b}\alpha = 0 \quad (3.3)$$

onde:

$\alpha \in \mathfrak{R}$	Fator de colapso
$\mathbf{x} \in \mathfrak{R}^n$	Campo de tensões
$\mathbf{g}(\mathbf{x}) \in \mathfrak{R}^m$	Função de escoamento
$\mathbf{B} \in \mathfrak{R}^{n \times n}$	Matriz de equilíbrio
$\mathbf{b} \in \mathfrak{R}^p$	Vetor de cargas nodais iniciais

Na tentativa de resolver eficientemente o problema da Análise Limite expresso pelas Equações (3.1), (3.2) e (3.3), foram pesquisados e testados vários otimizadores comerciais existentes baseados nas técnicas de programação matemática e otimizadores baseados em algoritmos genéticos. O uso de otimizadores baseados em algoritmos genéticos para resolver o problema expresso pelas Equações (3.1), (3.2) e (3.3) não foi bem sucedido e os otimizadores comerciais baseados nas técnicas de programação matemática mostraram ser ineficientes para problemas de grande escala como mostram os resultados de testes apresentados na seção 3.1. Finalmente, foi implementado um otimizador específico para o tipo de problema da análise limite baseado em técnicas de programação matemática (PM).

3.1. Otimizadores Matemáticos Testados

Os otimizadores baseados em técnicas de programação matemática que foram pesquisados e testados são LINGO (Lindo system inc., 1997), MINOS (Murtagh e Saunders, 1998) e LANCELOT (Conn, Gould e Toin, 1992).

Os problemas de otimização podem ser classificados como de pequena, média e de grande escala. Segundo Murtagh e Saunders (1998) os problemas são considerados de “pequena escala” quando o número de restrições lineares mais o número de restrições não lineares são menores que 100, de “média escala” quando o número de restrições são aproximadamente 1000 ou 2000 e de “grande escala” quando o número de restrições são maiores que 5000. O objetivo deste trabalho é resolver problemas da Análise Limite de grande escala, para os quais os otimizadores comerciais são testados.

Para testar a eficiência dos otimizadores indicados, um problema geotécnico de estabilidade de talude em 2D é formulado. A Figura 3.1 mostra a geometria, as condições de contorno e as propriedades do material do problema a ser analisado.

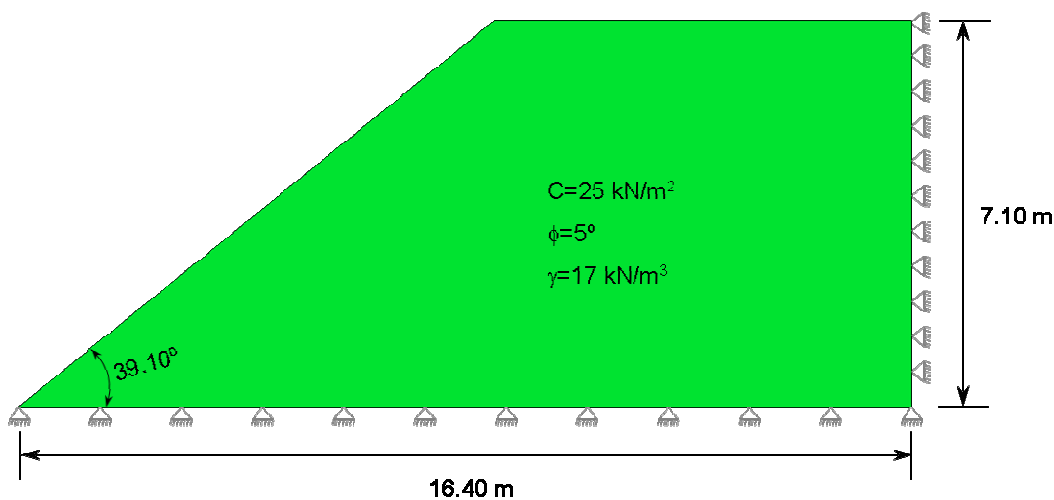


Figura 3.1 – Problema para teste de otimizadores.

Para se ter uma idéia comparativa do desempenho dos otimizadores, sete análises foram feitas mantendo constantes os seguintes parâmetros: geometria do problema, condições de contorno, propriedades do material (Figura 3.1) e os mesmos recursos de hardware (Pentium IV com processador Intel de 3.07 GHz e memória RAM de 1.4 Gb).

Para aumentar a complexidade do problema de otimização, a geometria do problema (Figura 3.1) foi modelada com malhas de Elementos Finitos com número de elementos variáveis, como mostrado pela Figura 3.2.

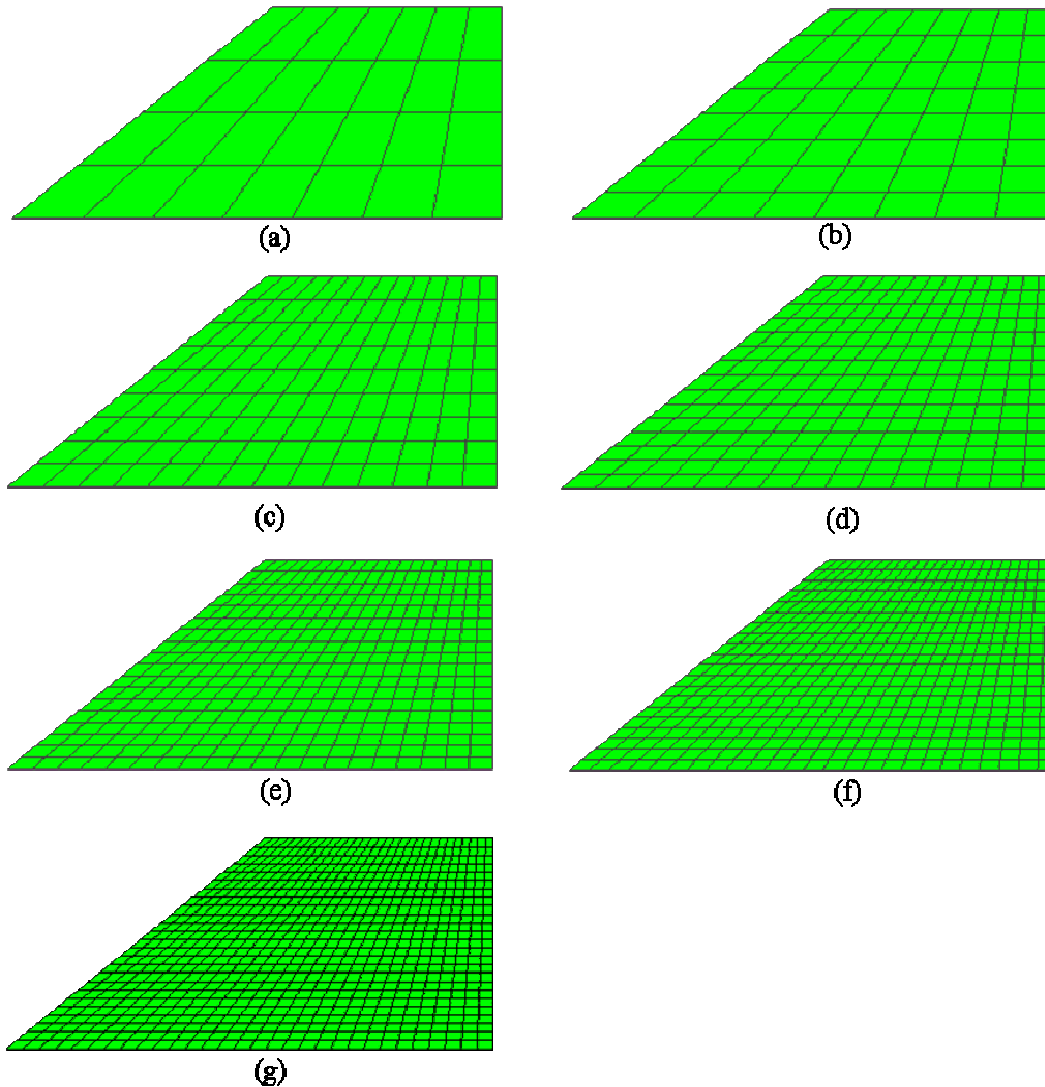


Figura 3.2 – Malhas: (a) 28, (b) 64, (c) 126, (d) 225, (e) 360, (f) 500 e (g) 750 elementos.

3.1.1. Otimizador Lingo

O programa LINGO desenvolvido pelo Lindo System Inc. (Lindo system inc., 1997) é um otimizador para resolver problemas de otimização não lineares.

A grande vantagem deste otimizador é a sua facilidade de uso já que o problema pode ser formulado facilmente no seu editor de texto ou ser importado a partir de um arquivo de texto. O uso do programa é fácil porque tanto a função objetivo, as restrições lineares e não lineares são escritas em forma de equações

explícitas. Uma outra vantagem deste programa é o pouco número de parâmetros a serem controlados pelo usuário.

Na Tabela 3.1 apresentam-se os resultados obtidos a partir dos testes realizados para o problema da Figura 3.1 com malhas da Figura 3.2, onde C é a coesão, ϕ ângulo de atrito e γ peso específico do material; n é o número de variáveis do problema, m é número de restrições não lineares e p número de restrições lineares; Mem é a quantidade de memória usada pelo otimizador, iter é o número de iterações, t é o tempo requerido para otimizar e α é o fator de colapso obtido.

Análise	Material			Malha		Problema				Lingo			
	C	ϕ	γ	Elem.	Nós	n	m	p	$m+p$	Mem.(MB)	α	iter.	t(seg)
1	25	5	17	28	40	85	28	56	84	0.200	3.76732	93	0
2	25	5	17	64	81	193	64	128	192	0.450	3.58688	245	2
3	25	5	17	126	150	379	126	252	378	0.875	3.52012	482	13
4	25	5	17	225	256	676	225	450	675	1.674	3.49650	857	110
5	25	5	17	360	399	1081	360	720	1080	2.790	3.47713	1492	775
6	25	5	17	500	546	1501	500	1000	1500	3.906	3.46887	3524	2305
7	25	5	17	750	806	2251	750	1500	2250	5.668	0.027280	43	2453

Tabela 3.1 – Testes realizados com o programa Lingo.

Nas Figuras 3.3a, 3.3b, 3.3c e 3.3d ilustram-se os resultados obtidos com o otimizador LINGO, onde, a Figura 3.3(b) indica a variação do número de iterações requeridas pelo otimizador com o incremento de complexidade do problema (número de elementos da malha). A Figura 3.3(c) indica o desempenho do otimizador em termos de tempo em segundos com o incremento de complexidade do problema. A Figura 3.3(a) indica a memória requerida pelo otimizador com o incremento de complexidade do problema. A Figura 3.3(d) indica o fator de colapso obtido com o refinamento da malha.

Os resultados indicam que o uso da memória cresce linearmente com a complexidade do problema, enquanto que o número de iterações tem uma variação quase linear até malhas com 35 elementos aproximadamente e logo aumenta fortemente. O resultado mais importante é o desempenho do otimizador em termos de tempo, a Figura 3.3c indica que o tempo de otimização cresce exponencialmente com o número de elementos da malha. A Figura 3.3d mostra a variação do fator de colapso com o refinamento da malha, este comportamento era de se esperar, porque equações de equilíbrio obtidas por malhas menos refinadas

obtidas por elementos finitos tendem a fornecer modelos mais rígidos e resistentes, superestimando as cargas de colapso. Isso explica porque o fator de colapso se aproxima pelo limite superior ao fator de colapso real como acontece na formulação pelo limite superior.

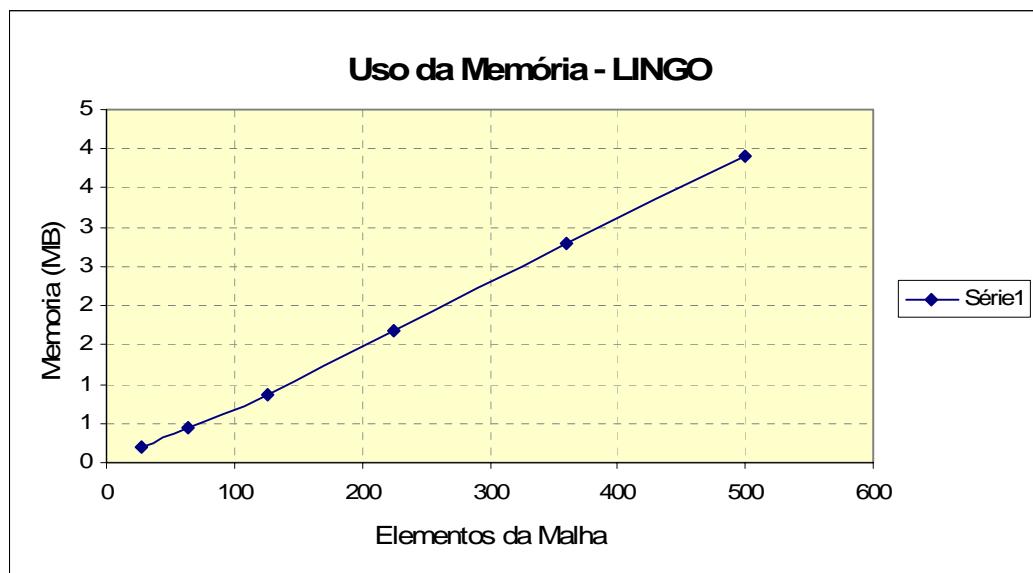


Figura 3.3a – Variação da memória usada pelo otimizador LINGO.

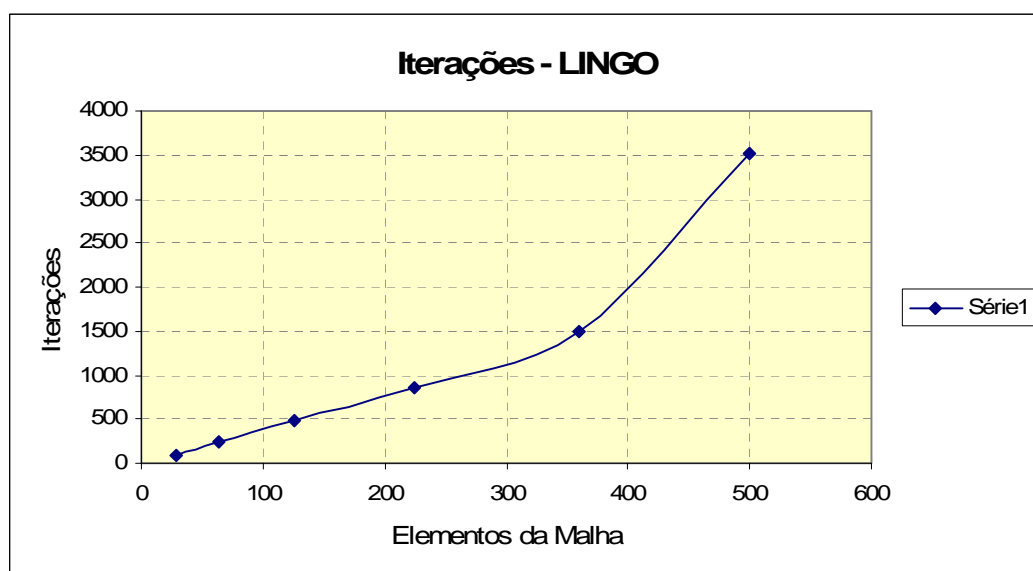


Figura 3.3b – Variação de número de iterações do otimizador LINGO.

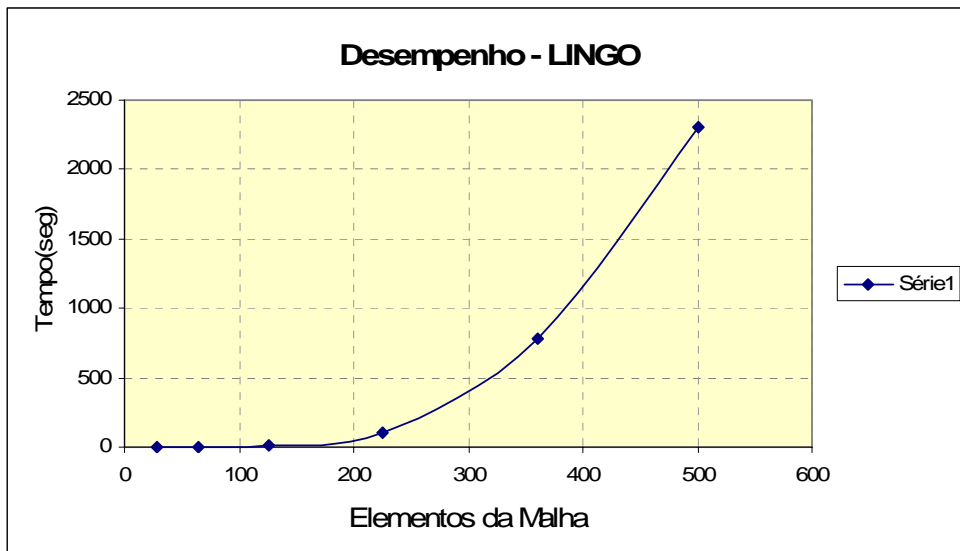


Figura 3.3c – Desempenho do otimizador LINGO.

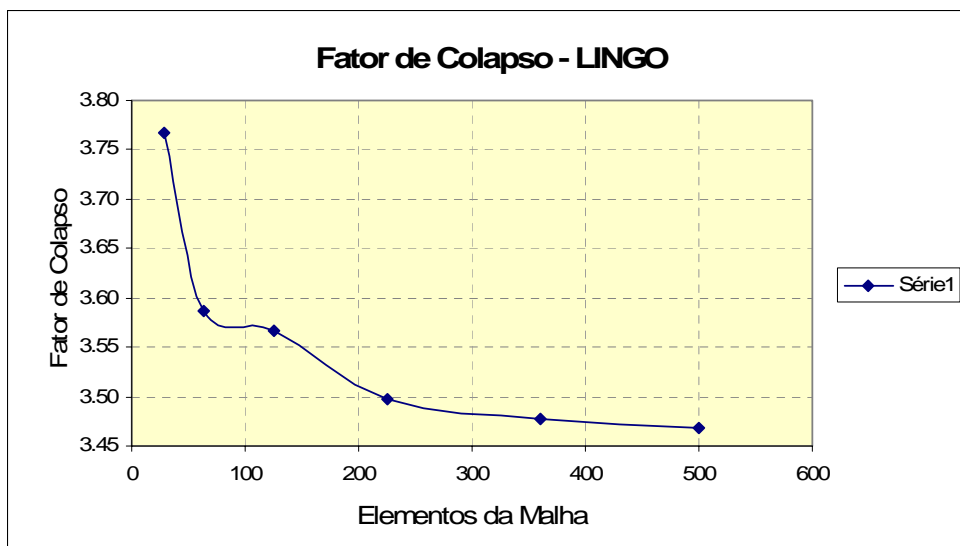


Figura 3.3d – Variação do fator de colapso obtido com LINGO.

Dos testes realizados com otimizador LINGO conclui-se que, para problemas pequenos com malhas de elementos finitos com número de elementos de até 200, a eficiência deste otimizador é aceitável como pode ser observado pelas análises 1, 2, 3 e 4 da Tabela 3.1, mas, para malhas com número de elementos maiores de 200, o tempo de otimização cresce exponencialmente (Figura 3.3c). Da Tabela 3.1, pode-se observar que para malha com 750 elementos este otimizador converge a uma solução não ótima, portanto, é de se esperar que o número de iterações e o tempo empregado não sejam reais e por isso a análise 7 não é usada para comparações.

3.1.2. Otimizador Minos

MINOS é um programa de otimização para resolver problemas de otimização lineares e não-lineares, desenvolvido na universidade de Stanford por Bruce A. Murtagh e Michael A. Saunders (Murtagh e Saunders, 1998).

O programa MINOS apresenta uma maior dificuldade para seu uso em comparação ao programa LINGO. Entre as dificuldades que apresenta MINOS estão a quantidade de parâmetros, 94 no total, que o usuário precisa controlar, uso de arquivo com formato MPS (Mathematical Programming System) para a entrada de dados das restrições lineares e um arquivo com formato SPECS (Specification) para especificar a configuração dos parâmetros.

O teste com este otimizador foi feito para o problema apresentado pela Figura 3.1, com malhas de Elementos Finitos apresentadas na Figura 3.2 e nas mesmas condições que os testes realizados com o otimizador LINGO.

Os resultados dos testes realizados com o otimizador MINOS são apresentados na Tabela 3.2 e nas Figuras 3.4a, 3.4b, 3.4c e 3.4d são ilustrados graficamente estes resultados.

Análise	Material			Malha		Problema				Minos			
	C	ϕ	γ	Elem.	Nós	n	m	p	m+p	Mem(MB)	α	iter.	t(seg)
1	25	5	17	28	40	85	28	56	84	253	3.767329	225	1
2	25	5	17	64	81	193	64	128	192	253	3.586883	599	4
3	25	5	17	126	150	379	126	252	378	253	3.566153	1602	6
4	25	5	17	225	256	676	225	450	675	253	3.497155	3556	16
5	25	5	17	360	399	1081	360	720	1080	253	3.478615	6561	72
6	25	5	17	500	546	1501	500	1000	1500	253	3.470797	10182	202
7	25	5	17	750	806	2251	750	1500	2250	295	3.453981	18067	671

Tabela 3.2 – Testes realizadas com o otimizador Minos.

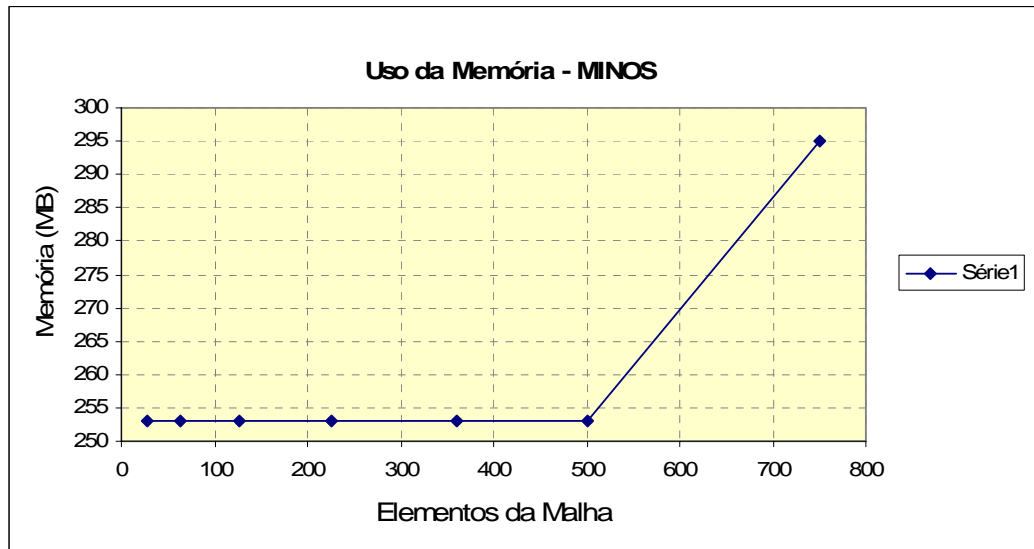


Figura 3.4a – Variação da memória usada pelo otimizador MINOS.

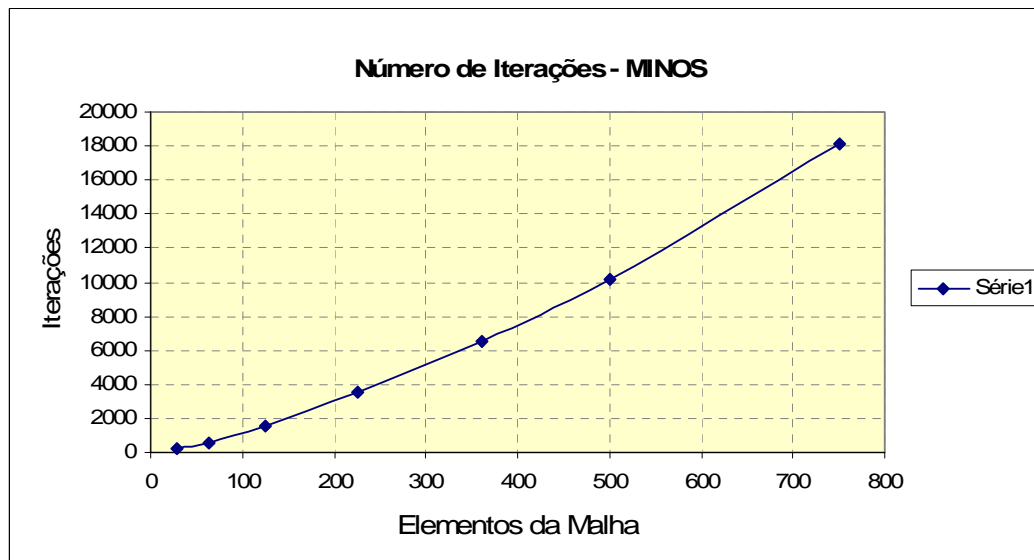


Figura 3.4b – variação de número de iterações do otimizador MINOS.

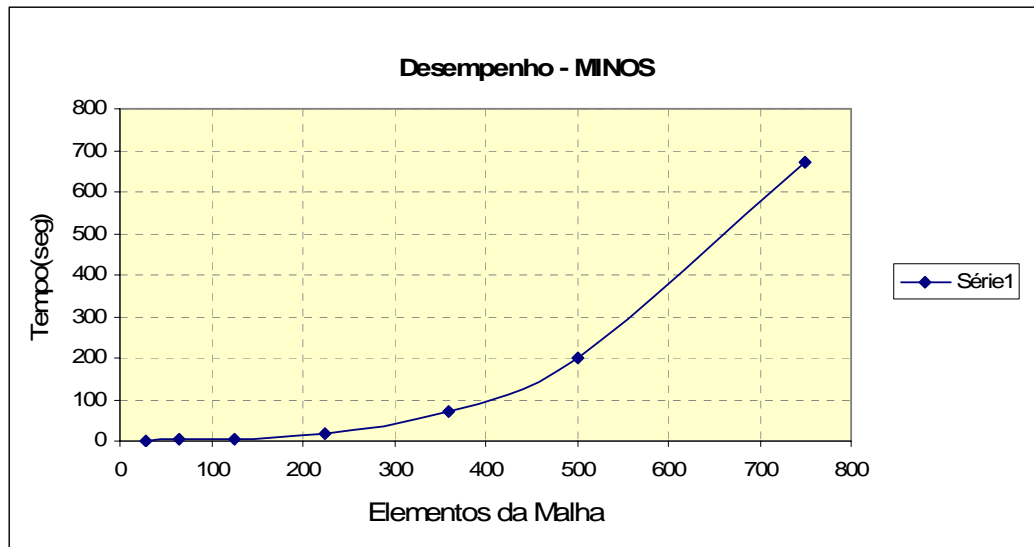


Figura 3.4c – Desempenho do otimizador MINOS.

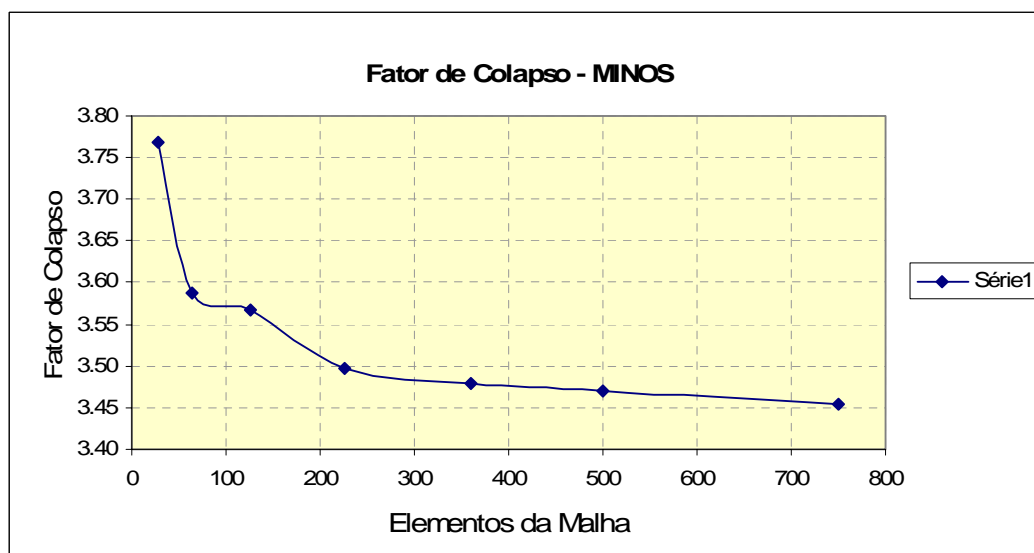


Figura 3.4d – Variação do fator de colapso obtido com MINOS.

Os resultados dos testes realizados mostram que este otimizador faz uma alocação prévia de 253 MB de memória como apresentado nas análises 1, 2, 3, 4, 5 e 6 da Tabela 3.2 e somente quando a aplicação precisa de mais memória, o programa faz uma realocação de memória como mostrado na análise 7 da Tabela 3.2.

Este otimizador apresenta um melhor desempenho que o otimizador LINGO, mas para malhas com mais 300 elementos o tempo de desempenho também cresce quase exponencialmente (Figura 3.4c).

O teste também mostra a mesma variação do fator de colapso com o incremento de número de elementos da malha obtidos com o otimizador LINGO (Figura 3.4d).

3.1.3. Otimizador Lancelot

O otimizador LANCELOT (Linear And Nonlinear Constrained Extended Lagrangian Optimization Techniques), segundo o manual do usuário (Conn, Gould e Toin, 1992) é um programa para resolver problemas de otimização não linear de grande escala.

Este programa foi desenvolvido pelos seguintes institutos de pesquisa: Facultés Universitaires Notre-Dame de la Paix, Namur – Bélgica; Harwell Laboratory, UK; IBM, USA; Rutherford Appleton Laboratory, UK e a University of Waterloo, Canada.

Este programa tem muitos parâmetros a serem controlados pelo usuário e para a especificação de dados utiliza arquivos com formato SIF (Standard Input Format). Este programa apesar de ser multi-plataforma não apresenta suporte ou não tem uma versão para o sistema operacional Windows. Os sistemas operacionais para os quais este programa apresenta suporte são:

- CRAY Unicos
- DEC OSF1
- DEC Ultrix
- DEC VMS
- HP HP-UX
- IBM AIX
- IBM CMS
- IBM DOS (Salford and Waterloo Fortran)
- Linux g77
- Silicon Graphics IRIX
- Sun SunOS and Solaris

No processo a pesquisa e adequação do código de LANCELOT para rodar no sistema operacional Windows, encontrou-se um trabalho de testes de eficiência e comparação realizados entre o programa LANCELOT e MINOS. O trabalho é denominado “A Numerical Comparison Between the LANCELOT and MINOS Package for Large-Scale Nonlinear Optimization” (Bongartz et al., 1997).

Na Tabela 3.3 apresentam-se os resultados de testes realizadas por Bongartz e outros (Bongartz et al., 1997), onde LP indica problemas lineares, QP problemas quadráticos, BC problemas sem restrição ou com restrições de contorno simplesmente, LC problemas com restrições lineares e NC problemas com restrições não lineares. A primeira coluna indica número total de problemas testados e as colunas LANCELOT e MINOS indicam o tempo total acumulado em segundos empregados pelos otimizadores.

	LANCELOT	MINOS
73 LP problems	45812	466
197 QP problems	2222	4703
253 BC problems	2758	14388
50 LC problems	1526	219
153 NC problems	15754	1093

Tabela 3.3 – Comparação da eficiência entre Lancelot e Minos (Bongartz et al, 1997).

Para os objetivos do presente trabalho de resolver eficientemente o problema da Análise Limite com restrições lineares e não lineares, os resultados mais importantes da Tabela 3.3 são as duas últimas linhas, onde se mostra que para problemas com restrições lineares (LC) o programa MINOS tem uma melhor eficiência com um tempo acumulado de 219 segundos contra 1526 segundos do programa LANCELOT e para problemas com restrições não lineares (NC) o programa MINOS também tem uma melhor eficiência com um tempo acumulado de 1093 segundos contra 15754 segundos de LANCELOT.

Pelos resultados apresentados no trabalho citado era de se esperar que não seria obtida nenhuma melhora em termos da eficiência (tempo de otimização) com o uso do programa LANCELOT e se decidiu não seguir com a pesquisa sobre o uso deste otimizador.

3.1.4. Comparação de Desempenho de Otimizadores Testados

Na Figura 3.5, apresenta-se uma comparação do desempenho dos otimizadores pesquisados e testados. A Figura 3.5a mostra que o otimizador LINGO faz um melhor uso da memória do computador, mas o resultado mais importante é que MINOS tem um melhor desempenho em termos de tempo empregado para otimizar o problema (Figura 3.5c). Ambos os otimizadores mostraram a mesma variação do fator de colapso com o número de elementos da malha (Figura 3.5d).

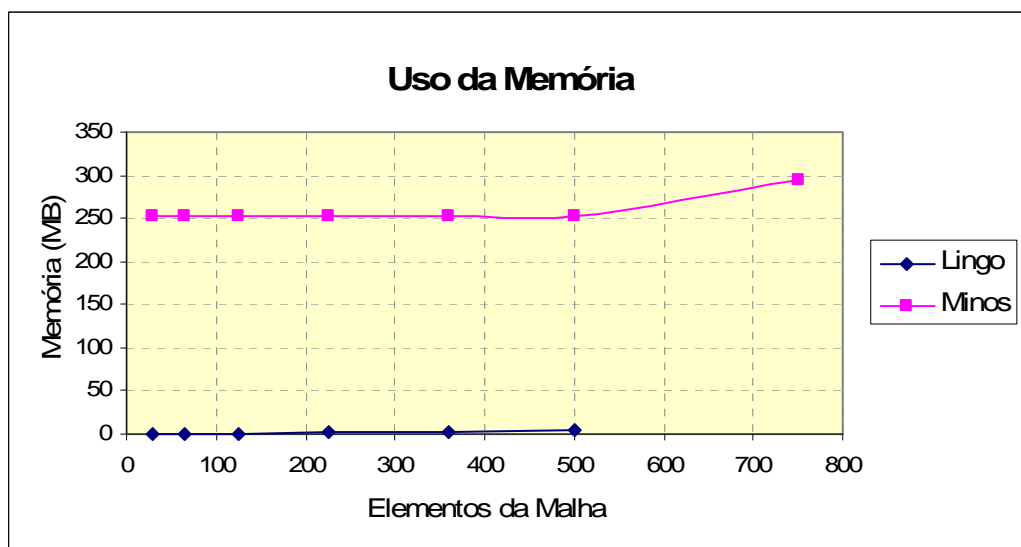


Figura 3.5a – Comparação de uso da memória entre Lingo e Minos

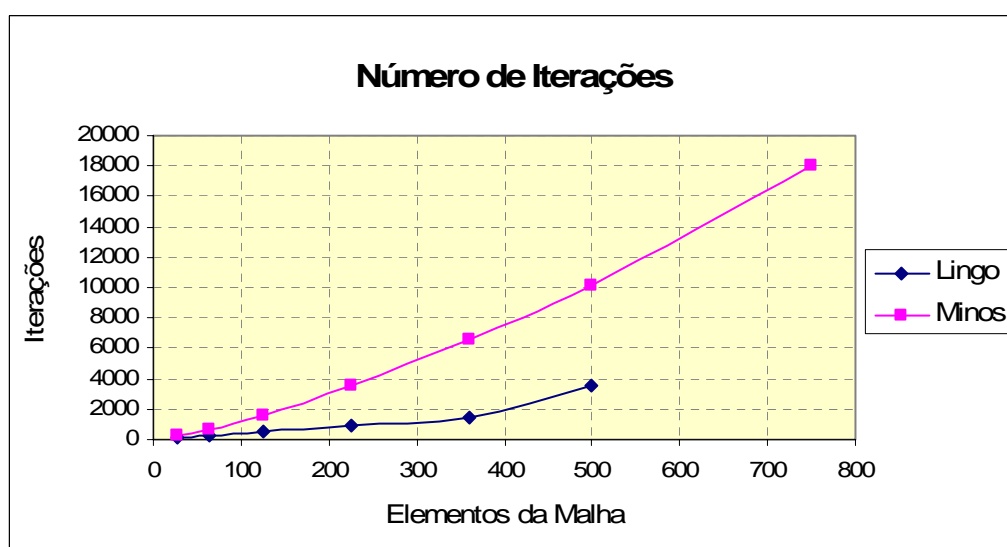


Figura 3.5b – Comparação de número de iterações entre Lingo e Minos

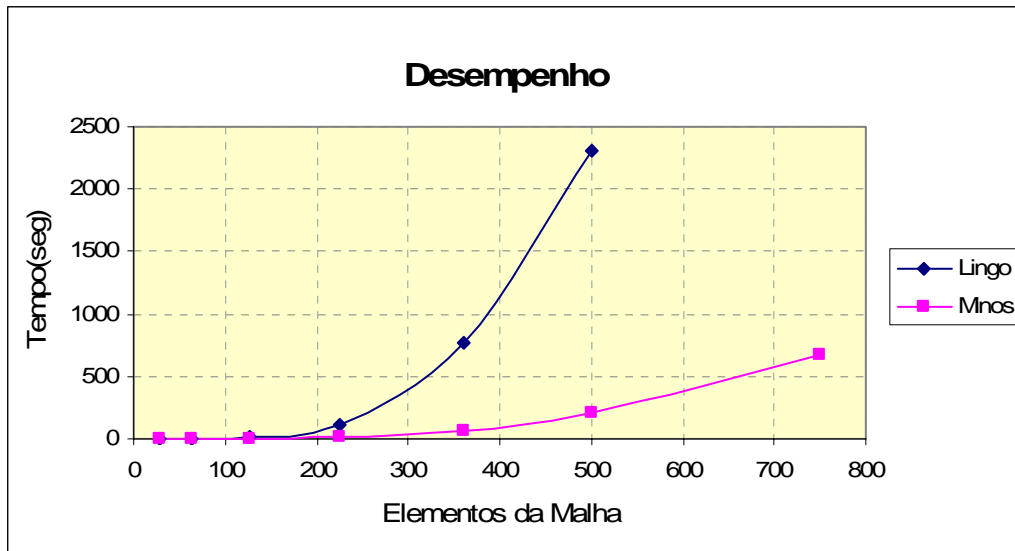


Figura 3.5c – Comparação de desempenho entre Lingo e Minos

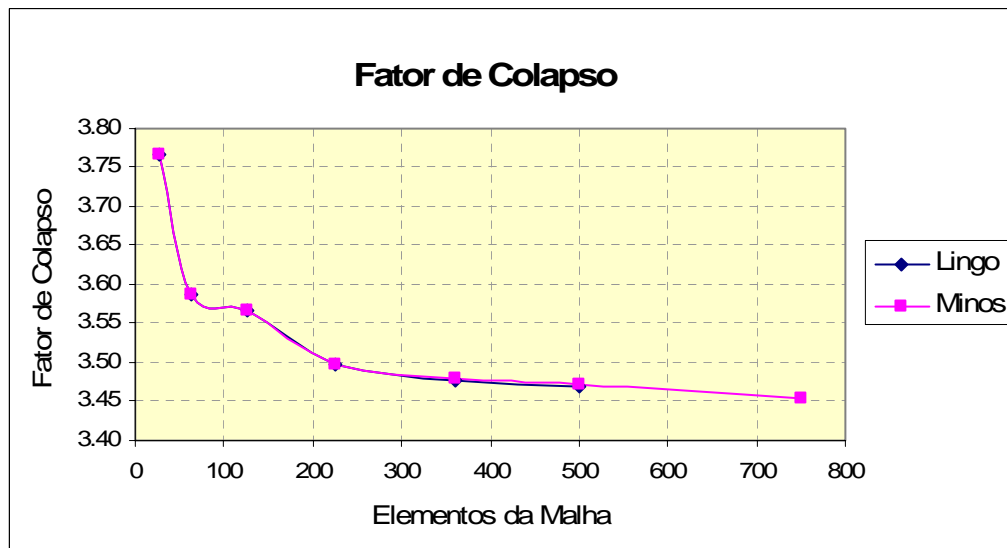


Figura 3.5d – Comparação de variação de fator de colapso, obtidos com Lingo e Minos.

Da pesquisa e dos testes realizados com os otimizadores comerciais existentes conclui-se que o otimizador LINGO pode ser usado eficientemente para problemas da Análise Limite de pequena escala e o otimizadores MINOS poderia ser usado eficientemente para problemas da Análise Limite de até média escala.

A Análise Limite de problemas em 3D tem uma complexidade maior se comparada com problemas em 2D porque o número de variáveis e o número de restrições é muito maior. Portanto, o uso destes otimizadores pesquisados é quase inviável. Em geral para problemas de Análise Limite de grande escala o uso destes otimizadores não é recomendável.

3.2. Otimizador Implementado

Após a tentativa de usar otimizadores comerciais como Lingo e Minos e sendo o desempenho alcançado não satisfatório para resolver problemas geotécnicos pela Análise Limite com malhas suficientemente refinadas ou problemas de grande escala, optou-se neste trabalho por desenvolver um otimizador exclusivo para o tipo de problema da Análise Limite. Assim, nesta pesquisa, implementou-se o algoritmo de Pontos Interiores. Sendo o objetivo da tese a aplicabilidade prática da Programação Matemática para resolver eficientemente o problema da análise limite, neste trabalho são expostos apenas alguns conceitos que permitem o entendimento do algoritmo implementado.

3.2.1. Condições de Otimalidade

As condições de otimalidade, conhecidas também como as condições de Karush, Kuhn e Tucker (Condições KKT), são o resultado teórico mais importante no campo da programação não linear. Estas condições devem de ser satisfeitas para a solução ótima de qualquer problema. Estas condições são as bases para o desenvolvimento de muitos algoritmos computacionais e proporciona um critério de parada para muitos outros (Castillo et all, 2002).

Para $S = \nabla g(x)$, as condições de otimalidade para o problema de otimização expresso pelas equações (3.1), (3.2) e (3.3), são escritas como segue:

$$\lambda^t S + \mu^t B = 0 \quad (3.4)$$

$$\mu^t b = -1 \quad (3.5)$$

$$B x - b \alpha = 0 \quad (3.6)$$

$$\lambda^t g(x) = 0 \quad (3.7)$$

$$\lambda_j \geq 0 \quad (3.8)$$

$$g(x) \leq 0 \quad (3.9)$$

onde: $\lambda \in \mathfrak{R}^m$ e $\mu \in \mathfrak{R}^p$ são os multiplicadores de Lagrange associados respectivamente às restrições (3.3) e (3.2). As condições (3.6) e (3.9) são conhecidas como condições de factibilidade primal, a condição (3.7) é conhecida como condição de complementaridade e a condição (3.8) indica a não negatividade do multiplicador de Lagrange e é conhecida como condição de factibilidade dual.

3.2.2. Algoritmo de Pontos Interiores

Este algoritmo foi proposto por Herskovits (1986 e 1989) (Farfan, 2000). Ele já foi usado em vários trabalhos para resolver problemas da Análise Limite como em Borges (1991), Zouain e outros (1993), Lyamin e Sloan (1997), e Farfan (2000).

Os algoritmos são chamados de Pontos Interiores por que todo ponto gerado deve de estar na região viável, ou seja, para cada iteração deve ser preservada a factibilidade das restrições de desigualdade $\mathbf{g}(\mathbf{x}) \leq 0$ e $\lambda \geq 0$. A condição $\lambda \geq 0$ é garantida pelo critério de atualização utilizado ao final de cada iteração. Portanto, deve-se garantir a admissibilidade plástica $\mathbf{g}(\mathbf{x}) \leq 0$.

Na literatura pesquisada encontraram-se dois algoritmos de pontos interiores; o primeiro, conhecido como Newton e Técnica de Deflexão e o segundo conhecido como Newton e Técnica de relaxação-contração. Neste trabalho, propõe-se uma outra técnica, denominada Vetorial.

Como discutido na seção 3.2.4 os três algoritmos se diferenciam na técnica utilizada para garantir a factibilidade das desigualdades. Sendo o objetivo deste trabalho implementar um otimizador eficiente, os três algoritmos foram implementados e testados.

3.2.3. Inicialização

A inicialização das variáveis primárias (Equações 3.10 e 3.11) é feita considerando que o ponto inicial deve satisfazer as equações de equilíbrio e a admissibilidade plástica do material. Para satisfazer a condição de otimalidade

(Equação 3.8) a variável dual λ deve ser positiva, portanto, esta variável é iniciada conforme sugerido por Borges (1991).

$$\mathbf{x}_0 = 0 \quad (3.10)$$

$$\alpha_0 = 0 \quad (3.11)$$

$$\lambda_0 = -\frac{1}{g(\mathbf{x}_0)} \quad (3.12)$$

3.2.4. Direção de Busca

Para o cálculo da direção de busca, o algoritmo de Newton-Rapson é aplicado nas equações de otimalidade (3.4-3.7) e para $\mathbf{H} = \lambda^k [\nabla^2 \mathbf{g}(\mathbf{x}^k)]$, $\mathbf{G} = \mathbf{g}_i(\mathbf{x}^k) / \lambda_i^k$ (matriz diagonal), $\mathbf{d}_x^k = \mathbf{x}^{k+1} - \mathbf{x}^k$ e $d_\alpha^k = \alpha^{k+1} - \alpha^k$ são obtidas as equações incrementais:

$$\mathbf{H} \mathbf{d}_x^k + \mathbf{S} \lambda^{k+1} + \mathbf{B} \boldsymbol{\mu}^{k+1} = \mathbf{0} \quad (3.13)$$

$$\mathbf{b}^t \boldsymbol{\mu}^{k+1} = -1 \quad (3.14)$$

$$\mathbf{B} \mathbf{d}_x^k - \mathbf{b} d_\alpha^k = 0 \quad (3.15)$$

$$\mathbf{S} \mathbf{d}_x^k + \mathbf{G} \lambda^{k+1} = \mathbf{0} \quad (3.16)$$

As Equações (3.13)-(3.16) formam um sistema de quatro blocos de equações com quatro grupos de incógnitas onde as variáveis ou incógnitas a serem determinadas são \mathbf{d}_x^k , λ^{k+1} , $\boldsymbol{\mu}^{k+1}$ e d_α^k . A solução deste sistema de equações é discutida na seção 3.2.8

Para o cálculo da direção de busca encontraram-se na literatura duas técnicas e uma terceira técnica é proposta neste trabalho.

A primeira técnica implica o cálculo de ângulo de deflexão e o sistema de equações é resolvido duas vezes (Zouain e outros, 1993). Na segunda técnica, fatores de relaxação-contração são usados para determinar a direção viável e o sistema de equações é resolvido somente uma vez (Borges, 1991). Uma terceira técnica vetorial é proposta neste trabalho, onde a direção viável é determinada mediante álgebra vetorial a partir do cálculo do ângulo de deflexão e o sistema de equações é resolvido uma única vez.

3.2.4.1. Técnica de deflexão

Nesta técnica, um ângulo de deflexão θ é calculado a partir da solução do sistema de equações (3.13-3.16). O ângulo de deflexão é calculado de tal forma a garantir uma direção viável para d_α^k e d_x^k como mostrada pela Equação 3.17, onde $\beta \in [0,1]$ é um parâmetro prefixado (Zouain et al,1993). Uma demonstração completa para o cálculo de θ é apresentada por Borges (1991) e Zouain e outros (1993).

$$\theta = \frac{(1-\beta)d_\alpha^k}{\sum \left(\frac{\lambda_i^{k+1}}{\lambda_i^k} \right)} \quad (3.17)$$

O sub-índice i indica que o somatório é calculado para todos os elementos da malha.

Na Figura 3.6 o ângulo de deflexão θ é ilustrado graficamente para uma restrição $g(x)$, onde a direção d_0 é uma direção não viável calculada a partir da solução do sistema de equações 3.13-3.17.

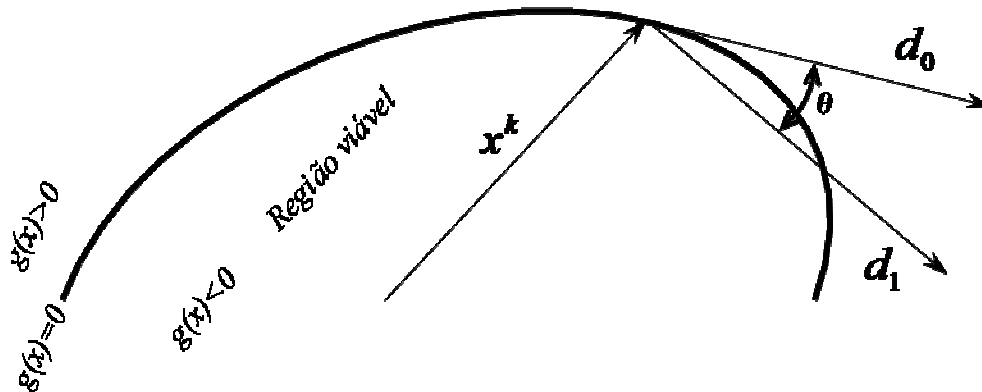


Figura 3.6 – Técnica de deflexão.

Após o cálculo do ângulo de deflexão θ , a direção de busca viável d_1 é calculada. Para isso o sistema de equações (3.13-3.16) é resolvido novamente mas desta vez considerando o ângulo de deflexão como mostrado pelas Equações (3.18-3.21).

$$H d_x^k + S \lambda^{k+1} + B \mu^{k+1} = 0 \quad (3.18)$$

$$b^t \mu^{k+1} = -1 \quad (3.19)$$

$$\mathbf{B} \mathbf{d}_x^k - \mathbf{b} \mathbf{d}_\alpha^k = 0 \quad (3.20)$$

$$\mathbf{S} \mathbf{d}_x^k + \mathbf{G} \lambda^{k+1} = -\theta \mathbf{e} \quad (3.21)$$

onde \mathbf{e} é um vetor com todos os componentes unitários.

3.2.4.2. Técnica de relaxação-contração

Neste algoritmo, a factibilidade das tensões é garantida propondo-se uma contração uniforme para as tensões e o fator de colapso. O procedimento consiste em incrementar as tensões e o fator de colapso a partir da solução das Equações (3.13-3.16), relaxada de um fator r . Em seguida calcula-se um fator de contração c para garantir a factibilidade das tensões. A contração do fator de colapso pelo mesmo fator c garante que a condição de equilíbrio (3.15) seja preservada ao final da iteração, enquanto o fator de relaxação r é calculado para garantir a condição de factibilidade na iteração.

Nesta técnica o novo ponto é calculado como mostrado na Equação 3.22 sem calcular direção de busca viável nem o comprimento de passo.

$$\mathbf{x}^{k+1} = c(\mathbf{x}^k + r \mathbf{d}) \quad (3.22)$$

Para ilustrar esta técnica apresenta-se a Figura 3.7, onde a direção \mathbf{d}_0 é calculada a partir da solução das Equações (3.13-3.16) e \mathbf{x}^k são as variáveis da iteração anterior. Esta técnica consiste em que para um fator de relaxação r o vetor $\mathbf{AM} = r \mathbf{d}_0$ é calculado, a seguir o vetor $\mathbf{OM} = \mathbf{OA} + \mathbf{AM}$ é calculado e finalmente o vetor $\mathbf{ON} = c \mathbf{OM}$ é calculado. Este processo é realizado para diferentes valores do fator de relaxação r até que se cumpra a seguinte condição de acréscimo $\alpha^{k+1} > \alpha^k$. O fator de contração c neste processo é determinado pela busca linear na direção do vetor \mathbf{OM} .

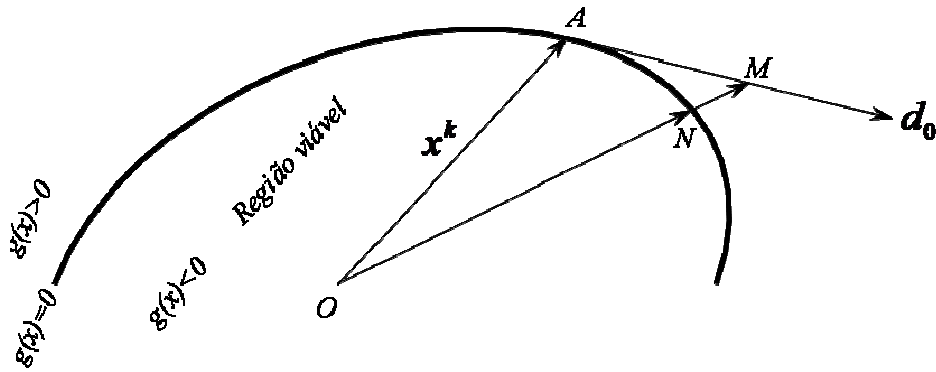


Figura 3.7 – Técnica de Relaxação-Contração.

3.2.4.3. Técnica Vetorial - Proposta

Pelo fato de que, resolver duas vezes o sistemas de equações como proposto pela primeira técnica (de deflexão) é computacionalmente custoso e isto afeta diretamente o desempenho do otimizador implementado, neste trabalho propõe-se uma técnica vetorial para o cálculo da direção de busca viável. Esta técnica é descrita a seguir.

Da solução do sistema de Equações (3.13-3.16) a direção tangente não viável $d_0 = d_x^k$ é determinada como mostra a Figura 3.8, a seguir um ângulo de deflexão é calculado como expresso pela Equação 3.17 e finalmente a direção viável d_1 é calculada vetorialmente mediante álgebra vetorial, como a soma de dois vetores (Figura 3.8), onde um vetor unitário na direção negativa do gradiente é determinado pela Equação 3.23, a seguir a norma euclidiana do vetor oposto ao ângulo de deflexão é calculada pela Equação 3.24 e logo a direção de busca viável d_1 é obtida como a soma de dois vetores conforme Equação 3.25.

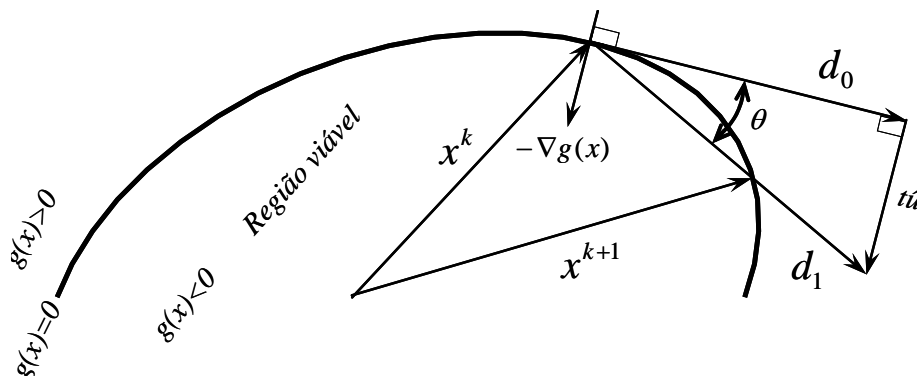


Figura 3.8– Técnica Vetorial - Proposta.

$$\hat{\mathbf{u}} = \frac{-\nabla \mathbf{g}(\mathbf{x})}{\|\nabla \mathbf{g}(\mathbf{x})\|} \quad (3.23)$$

$$t = \|\mathbf{d}_0\| \tan(\theta) \quad (3.24)$$

$$\mathbf{d}_1 = \mathbf{d}_0 + t \hat{\mathbf{u}} \quad (3.25)$$

onde $\nabla \mathbf{g}(\mathbf{x})$, é o vetor gradiente com n elementos formados pelos gradientes das restrições $g_i(\mathbf{x})$ de cada elemento, $\hat{\mathbf{u}}$ é um vetor unitário no sentido oposto ao vetor gradiente e t é o módulo do vetor oposto ao ângulo de deflexão θ .

3.2.5. Comprimento de Passo

O comprimento de passo indicado na Figura 3.9 é determinado somente para a primeira e a terceira técnica.

Determinada a direção de busca viável \mathbf{d}_1 na etapa anterior, é necessário determinar o comprimento de passo. As restrições não lineares neste trabalho são apresentadas pelos critérios de escoamento (Equações 2.8 e 2.9). Por sugestão de Herskovits (2008), estas equações foram transformadas na sua forma quadrática equivalente e substituindo na Equação 3.26 é obtida uma equação quadrática em função de s como expresso na Equação 3.27, onde $a = f(\mathbf{d}_1, \phi)$, $c = f(\mathbf{x}^k, \mathbf{d}_1, C, \phi)$ e $q = f(\mathbf{x}^k, C, \phi, \gamma_f)$. Finalmente, o comprimento de passo s é obtido resolvendo a equação quadrática 3.27.

$$g(\mathbf{x}^k + s \mathbf{d}_1) - \gamma_f . g(\mathbf{x}^k) = 0 \quad (3.26)$$

$$a s^2 + c s + q = 0 \quad (3.27)$$

onde γ_f é um parâmetro (Borges, 1991), C e ϕ são as propriedades do material.

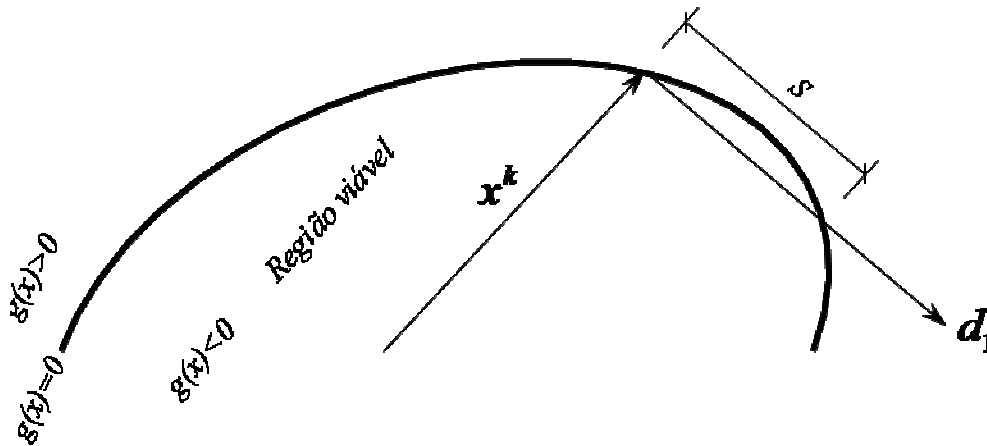


Figura 3.9 – Comprimento de passo s .

3.2.6. Atualização das Variáveis

Calculada a direção viável e o comprimento de passo nas etapas anteriores as variáveis x e α são atualizadas conforme Equações 3.28 e 3.29. Graficamente, este processo de atualização é ilustrado na Figura 3.10, onde se vê que o x^{k+1} é atualizado como a soma dos vetores x^k e $s d_1$.

$$x^{k+1} = x^k + s d_x^k \quad (3.28)$$

$$\alpha^{k+1} = \alpha^k + s d_\alpha^k \quad (3.29)$$

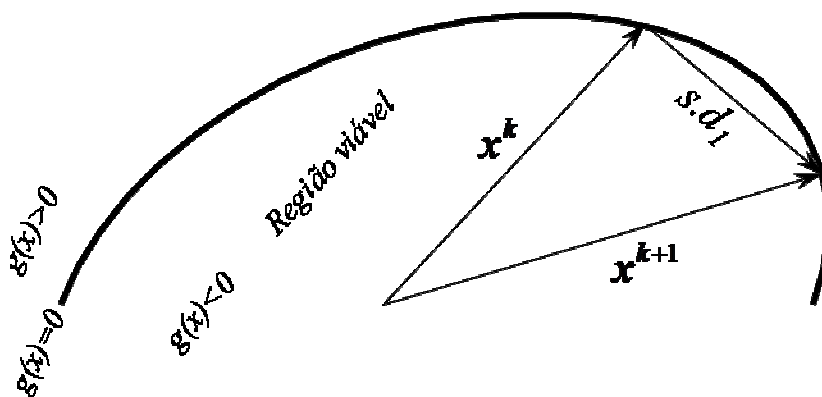


Figura 3.10 – Atualização da variável x .

Pela condição de Karush-Kuhn-Tucker (Equação 3.8) a variável dual λ tem que ser sempre positiva. A atualização desta variável é feita da seguinte forma:

$$\lambda_i^{k+1} = \max(\lambda_i^k, \gamma_\lambda \lambda_\infty) \quad (3.30)$$

onde $\lambda_\infty = \max\{\lambda_1^k, \lambda_2^k, \dots, \lambda_m^k\}$ e γ_λ é um parâmetro (Borges, 1991).

3.2.7. Teste de Desempenho dos Algoritmos

Para ilustrar o desempenho dos algoritmos, um talude 2D como indicado na Figura 3.11 é analisado. As análises são feitas para as mesmas condições, ou seja, considerando-se as mesmas propriedades do material coesão ($C=25$ kN/m²), ângulo de atrito ($\phi = 5^\circ$) e peso específico ($\gamma = 17$ kN/m³); a mesma geometria com o mesmo número de elementos da malha; o mesmo critério de escoamento, Mohr-Coulomb; e usando o mesmo computador (Pentium IV com processador de 3.07 GHz e memória RAM de 1.4 Gb).

Na Tabela 3.4, apresentam-se as comparações de fator de colapso, número de iterações, tempo de otimização e o mecanismo de ruptura obtido com cada um dos algoritmos.

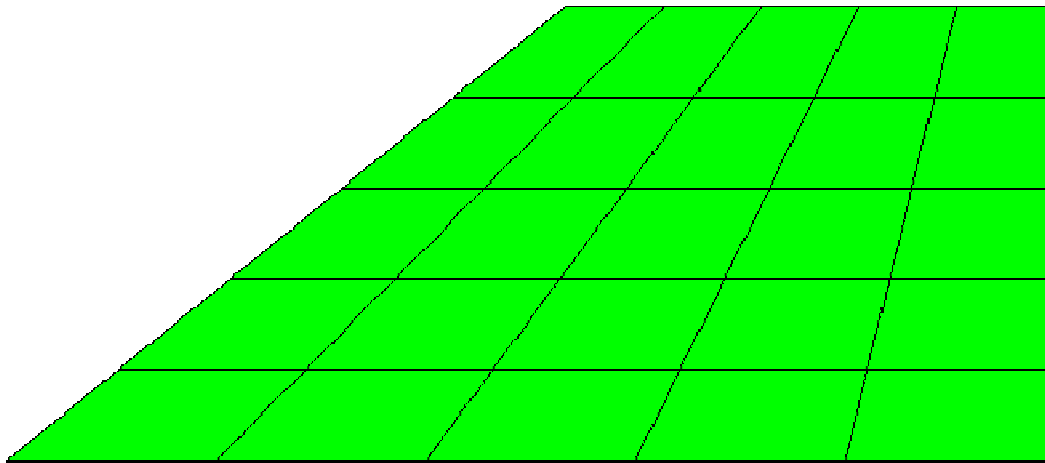


Figura 3.11 Geometria do problema para teste de algoritmos (malha de 25 elementos).

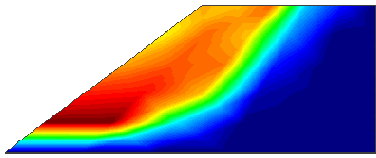
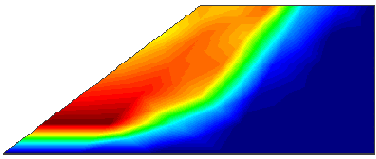
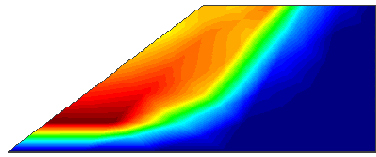
Algoritmo	α	Iter	t(seg)	Superfície de Falha
Deflexão	3.717250	13	1.040	
Relax.-Cont.	3.717627	18	2.250	
Vetorial	3.668298	13	0.687	

Tabela 3.4 – Desempenho dos algoritmos implementados

3.2.8. Manipulação de sistemas lineares a serem resolvidos

Para resolver os sistemas de Equações (3.13-3.16) e (3.18-3.21), na literatura pesquisada foram encontradas duas formas de reduzir estes sistemas para logo serem resolvidos por algum método direto ou indireto, onde o sistema a ser resolvido tem a mesma ordem que o número de restrições lineares do problema. A primeira, é uma manipulação matricial global como mostrado por Lyamin & Sloan (1997) ou por Farfan (2002). A segunda é uma manipulação matricial por elementos como mostrado por Borges (1991) ou Zouain e outros (1993). Neste trabalho, é proposta uma terceira forma de resolver estes sistemas mediante uma solução direta sem manipulação prévia.

Para $\lambda^k = \lambda_0$, $x^k = x_0$ e $\lambda^k = \lambda_0$, as equações (3.13), (3.14), (3.15) e (3.16) podem ser re-escritas como as Equações (3.31), (3.32), (3.33) e (3.34) representado um sistema de 4 blocos de equações com 4 grupos variáveis (d_x^k , λ^{k+1} , μ^{k+1} e d_α^k).

$$Hd_x^k + S^T \lambda^{k+1} + B \mu^{k+1} = 0 \quad (3.31)$$

$$S d_x^k + G \lambda^{k+1} = 0 \quad (3.32)$$

$$\mathbf{B} \mathbf{d}_x^k - \mathbf{b} d_\alpha^k = 0 \quad (3.33)$$

$$\mathbf{b}^t \boldsymbol{\mu}^{k+1} = -1 \quad (3.34)$$

Neste trabalho foram implementadas e testadas as três formas de resolver o sistema de Equações 3.31-3.34 descritas acima, tendo como objetivo a busca de um melhor desempenho do otimizador implementado. As vantagens e desvantagens de cada uma das formas de resolver são descritas a seguir e finalmente é apresentada uma comparação de desempenho testado.

3.2.8.1. Manipulação matricial global

Nesta forma de manipulação as matrizes \mathbf{H} , \mathbf{S} , \mathbf{G} e \mathbf{B} são calculadas para todo o sistema, onde $\mathbf{H} \in \mathfrak{R}^{n \times n}$, $\mathbf{S} \in \mathfrak{R}^{m \times n}$, $\mathbf{G} \in \mathfrak{R}^{m \times m}$ e $\mathbf{B} \in \mathfrak{R}^{p \times n}$; n é o número de variáveis do problema, m é o número de restrições não lineares e p é o número de restrições lineares. O vetor \mathbf{d}_x^k pode ser explicitado na Equação 3.31 gerando a Equação 3.35. Substituindo-se em seguida \mathbf{d}_x^k da Equação 3.35 na Equação 3.32 e considerando que $\mathbf{Q} = \mathbf{S} \mathbf{H}^{-1}$ e $\mathbf{W} = \mathbf{Q} \mathbf{S}^T - \mathbf{G}$ a Equação 3.36 é obtida.

$$\mathbf{d}_x^k = -\mathbf{H}^{-1} (\mathbf{S}^T \boldsymbol{\lambda}^{k+1} + \mathbf{B} \boldsymbol{\mu}^{k+1}) \quad (3.35)$$

$$\boldsymbol{\lambda}^{k+1} = -\mathbf{W}^{-1} \mathbf{Q} \mathbf{B}^T \mathbf{u}^{k+1} \quad (3.36)$$

Para $\mathbf{D} = \mathbf{H}^{-1} - \mathbf{Q}^T \mathbf{W}^{-1} \mathbf{Q}$ e substituindo a Equação 3.36 em 3.35, a Equação 3.37 é obtida. Substituindo a Equação 3.37 na Equação 3.33, e para $\mathbf{K} = \mathbf{B} \mathbf{D} \mathbf{B}^T$ e $\mathbf{v} = \mathbf{u}^{k+1} (\mathbf{d}_\alpha^k)^{-1}$, a Equação 3.38 é obtida.

$$\mathbf{d}_x^k = -\mathbf{D} \mathbf{B}^T \mathbf{u}^{k+1} \quad (3.37)$$

$$\mathbf{K} \mathbf{v} = -\mathbf{b} \quad (3.38)$$

A matriz \mathbf{K} pode ser chamado de uma pseudo matriz de rigidez já que ela relaciona forças aplicadas e velocidades.

A Equação 3.38 é um sistema de equações lineares de ordem igual ao número de restrições lineares (p) e pode ser resolvido por qualquer dos métodos discutidos na seção 3.2.9.

Conhecido o vetor v e substituindo $\mathbf{u}^{k+1} = v d_\alpha^k$ na Equação 3.34, a variável d_α^k é determinada e, logo, \mathbf{u}^{k+1} é conhecido também. Substituindo \mathbf{u}^{k+1} nas Equações 3.37 e 3.36 as variáveis d_x^k e λ^{k+1} são calculados.

O processo de solução do sistema de Equações 3.18, 3.19, 3.20 e 3.21 considerando o ângulo de deflexão θ são similares e pode ser encontrado em Lyamin e Sloan (1997) ou Farfan (2000).

Observa-se que o uso deste processo implica realizar operações como a obtenção da inversa de matriz de ordem n (número de variáveis do problema), assim como o produto de matrizes e produto de matrizes por vetores. Sendo estas operações computacionalmente caras é de se esperar que esta implementação seja ineficiente como ilustrado com os testes realizados.

3.2.8.2. Manipulação matricial por elementos

Uma segunda forma de manipulação na solução de sistema de Equações (3.31), (3.32), (3.33) e (3.34) é apresentada por Borges (1991) e por Zouain e outros em (1993). Nesta forma, a matriz \mathbf{K} da Equação 3.38 é obtida pela montagem de matrizes de rigidez elástico-plástica de cada elemento finito.

Nesta forma de manipulação as variáveis \mathbf{H} , \mathbf{S} , \mathbf{Q} , \mathbf{W} , \mathbf{D}^{ep} e \mathbf{K}^e são calculadas para cada elemento finito como indicado nas equações a seguir:

$$\mathbf{H} = \lambda[\nabla^2 \mathbf{g}(\mathbf{x})] \quad (3.39)$$

$$\mathbf{S} = \nabla \mathbf{g}(\mathbf{x}) \quad (3.40)$$

$$\mathbf{G} = \mathbf{g}(\mathbf{x}) / \lambda^k \quad (3.41)$$

$$\mathbf{Q} = \mathbf{S} \mathbf{H}^{-1} \quad (3.42)$$

$$\mathbf{W} = \mathbf{Q} \mathbf{S}^T - \mathbf{G} \quad (3.43)$$

$$\mathbf{D}^{ep} = \mathbf{H}^{-1} - \mathbf{Q}^T \mathbf{W}^{-1} \mathbf{Q} \quad (3.44)$$

$$\mathbf{K}^e = \mathbf{B}^T \mathbf{D}^{ep} \mathbf{B} \quad (3.45)$$

onde, para os elementos finitos implementados, quadrilateral em 2D e hexaédrico em 3D, \mathbf{H} é uma matriz de 3x3 para problemas em 2D e de 6x6 para problemas em 3D, \mathbf{S} e \mathbf{Q} são vetores de 3 elementos para problemas em 2D e de 6

elementos para problemas em 3D, \mathbf{G} e \mathbf{W} são matrizes de 1x1, \mathbf{D}^{ep} é a matriz conhecida como módulo elástico-plástico de 3x3 para problemas em 2D e de 6x6 para problemas em 3D, \mathbf{B} é a matriz de equilíbrio para cada elemento finito obtido pelo MEF com dimensões de 8x3 para problemas 2D e de 24x6 para problemas em 3D e \mathbf{K}^e é a matriz de rigidez elástico-plástica de 8x8 para problemas em 2D e de 24x24 para problemas em 3D.

Da Equação 3.31, a variável \mathbf{d}_x^k para cada elemento pode ser expressa como mostra a Equação 3.46. Substituindo (3.46) em (3.32), a Equação 3.47 é obtida.

$$\mathbf{d}_x^k = -\mathbf{H}^{-1}(\mathbf{S}^T \boldsymbol{\lambda}^{k+1} + \mathbf{B} \boldsymbol{\mu}^{k+1}) \quad (3.46)$$

$$\boldsymbol{\lambda}^{k+1} = -\mathbf{W}^{-1} \mathbf{Q} \mathbf{B}^T \mathbf{u}^{k+1} \quad (3.47)$$

Substituindo a Equação 3.47 em 3.46 e para módulo elástico-plástico \mathbf{D}^{ep} expresso pela Equação 3.44 a Equação 3.48 é obtida. Substituindo a Equação 3.48 na Equação 3.33, para \mathbf{K}^{ep} expresso pela Equação 3.45 e para $\mathbf{u}^{k+1} = \mathbf{v}^e \mathbf{d}_\alpha^k$ a Equação 3.49 é obtida.

$$\mathbf{d}_x^k = -\mathbf{D}^{ep} \mathbf{B}^T \mathbf{u}^{k+1} \quad (3.48)$$

$$\mathbf{K}^e \mathbf{v}^e = \mathbf{b}^e \quad (3.49)$$

A partir da matriz de rigidez elástico-plástica \mathbf{K}^e de cada elemento (Equação 3.49), a matriz de rigidez elástico-plástico \mathbf{K} para todo o sistema é montada. Esta montagem é similar à montagem de matriz de rigidez global da análise convencional por Elementos Finitos. Uma vez montada a matriz \mathbf{K} e sendo \mathbf{b} vetor dos carregamentos nodais montado também a partir dos carregamentos de cada elemento, a Equação 3.49 é escrita para todo o sistema como indicado em Equação 3.50.

$$\mathbf{K} \mathbf{v} = \mathbf{b} \quad (3.50)$$

A Equação 3.50 é similar à Equação 3.38 e pode ser resolvida por algum dos métodos discutidos na seção 3.2.9. Determinado o vetor \mathbf{v} as variáveis \mathbf{d}_α^k e \mathbf{u}^{k+1} são determinadas como expressas pelas equações 3.51 e 3.52.

$$\mathbf{d}_\alpha^k = (\mathbf{b} \cdot \mathbf{v})^{-1} \quad (3.51)$$

$$\mathbf{u}^{k+1} = d_\alpha^k \mathbf{v} \quad (3.52)$$

Sendo o vetor \mathbf{u}^{k+1} um vetor global para todo o sistema, este vetor é decomposto facilmente para cada um dos elementos e logo os vetores \mathbf{d}_x^k e λ^{k+1} para cada elemento são calculados pelas Equações 3.48 e 3.47. Este mesmo processo também é seguido para resolver as Equações de 3.31 a 3.34, mas considerando o ângulo de deflexão θ . Uma abordagem completa pode ser encontrada em Borges (1991) e Zouain e outros (1993).

Esta forma de manipulação para resolver o sistema de Equações (3.31-3.34) é mais eficiente que o primeiro porque faz um melhor uso da memória do computador e os cálculos como a inversa da matriz, o produto de matrizes e o produto de matrizes por vetores são feitos no nível de cada elemento e não para todo o sistema.

3.2.8.3. Solução direta sem manipulação - proposta

Como uma alternativa às duas formas anteriores de resolver o sistema de Equações (3.31-3.34), neste trabalho propõe-se resolver diretamente este sistema de equações como mostra a Equação 3.53.

O sistema de Equações (3.18-3.21) ou (3.31-3.34) considerando o ângulo de deflexão θ pode ser resolvido também como indicado na Equação 3.54.

$$\begin{bmatrix} \mathbf{H} & \mathbf{S}^t & \mathbf{B}^t & \mathbf{0} \\ \mathbf{S} & \mathbf{G} & \mathbf{0} & \mathbf{0} \\ \mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{b} \\ \mathbf{0} & \mathbf{0} & -\mathbf{b}^t & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{d}_x^k \\ \lambda^{k+1} \\ \boldsymbol{\mu}^{k+1} \\ d_\alpha^k \end{Bmatrix} = \begin{Bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ 1 \end{Bmatrix} \quad (3.53)$$

$$\begin{bmatrix} \mathbf{H} & \mathbf{S}^t & \mathbf{B}^t & \mathbf{0} \\ \mathbf{S} & \mathbf{G} & \mathbf{0} & \mathbf{0} \\ \mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{b} \\ \mathbf{0} & \mathbf{0} & -\mathbf{b}^t & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{d}_x^k \\ \lambda^{k+1} \\ \boldsymbol{\mu}^{k+1} \\ d_\alpha^k \end{Bmatrix} = \begin{Bmatrix} \mathbf{0} \\ -\theta \mathbf{e} \\ \mathbf{0} \\ 1 \end{Bmatrix} \quad (3.54)$$

Esta forma de resolver os sistemas de Equações (3.31-3.34) tem vantagens e desvantagens. A vantagem é que não é necessário realizar nenhum tipo de cálculo ou manipulação prévia à solução do sistema. A desvantagem é que o sistema de equações a ser resolvido é de ordem de $(n + m + p + 1)$, onde $(n + 1)$ é o número de variáveis, m é o número de restrições não lineares de desigualdade e p é o número de restrições lineares de igualdade; isso indicaria que é necessário realizar mais esforço computacional em termos de FLOPS (FLoating-point OPerationS) e também uso de mais recursos do computador como a memória para armazenar a matriz de coeficientes.

Sendo a matriz dos coeficientes uma matriz esparsa, as desvantagens indicadas no parágrafo anterior como número de FLOPS e requerimento de mais memória para armazenar a matriz de coeficientes são resolvidas neste trabalho mediante o uso de uma técnica de tratamento de matrizes esparsas como vetores, como discutido na seção 3.3.1.

3.2.8.4. Teste de Desempenho das Manipulações

Neste trabalho, as três formas de manipular e resolver o sistema de Equações (3.31-3.34) foram implementadas e testadas. A eficiência das formas de manipular e resolver os sistemas de equações é medida em termos de quantidade de memória requerida e de tempo empregado na manipulação e solução do sistema.

Os testes foram feitos utilizando cada uma das manipulações para o problema apresentado na Figura 3.1 com malhas apresentadas na Figura 3.2. Nos testes, para facilitar a identificação de cada método, a manipulação matricial global é simplesmente denominada como “Matriz”, a manipulação matricial por elementos é denominada como “Elementos” e a solução direta sem manipulação é denominada como “Direta”.

a) Teste de manipulação matricial global

A Tabela 3.5 apresenta os resultados dos testes realizados com a implementação mediante a manipulação matricial global. Nas Figuras 3.12 e 3.13 estes resultados são apresentados graficamente para uma melhor interpretação.

Como se pode verificar nas Figuras 3.12 e 3.13 tanto a quantidade de memória requerida como o tempo empregado para resolver o sistema cresce exponencialmente neste método. Devido ao fato de que estes valores têm um crescimento exponencial os testes somente foram feitos para malhas de 28, 64 e 126 elementos.

Da Figura 3.13 pode-se observar também que entre 80 a 90 % de tempo total é empregado na manipulação do sistema e de 10 a 20 % de tempo é empregado na solução do sistema resultante.

Manipulação	Malha		Problema				Mem. (MB)	Tempo(seg)		
	Elem.	Nós	n	m	p	$m+p$		Manip.	Resolv.	total
Matriz	28	40	85	28	56	84	0.54	0.125	0.015	0.140
	64	81	193	64	128	192	1.50	0.750	0.250	1.000
	126	150	379	126	252	378	47.00	6.844	1.172	8.016
	225	256	676	225	450	675				
	360	399	1081	360	720	1080				
	500	546	1501	500	1000	1500				
	750	806	2251	750	1500	2250				

Tabela 3.5 – Teste da manipulação matricial global.

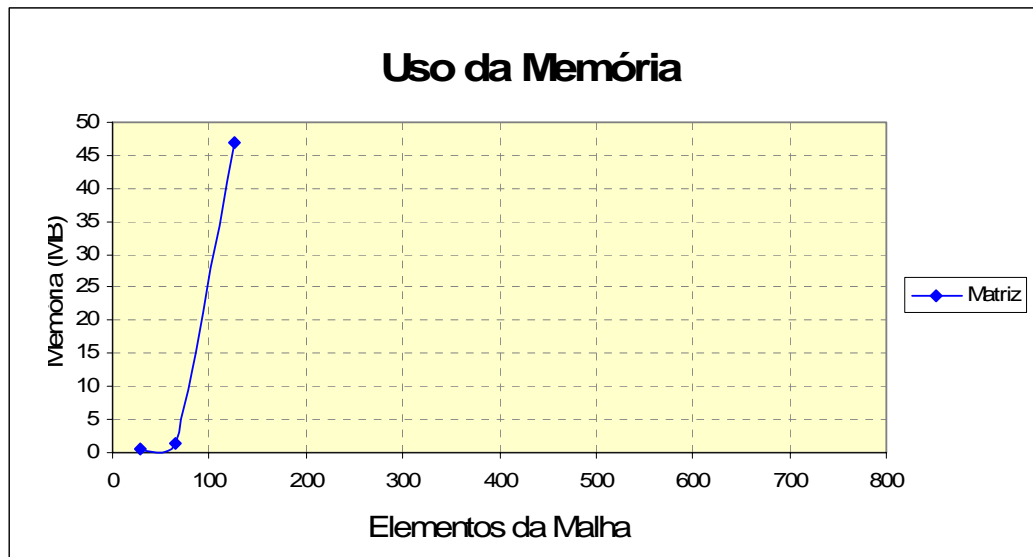


Figura 3.12 – Memória requerida pela manipulação matricial global.

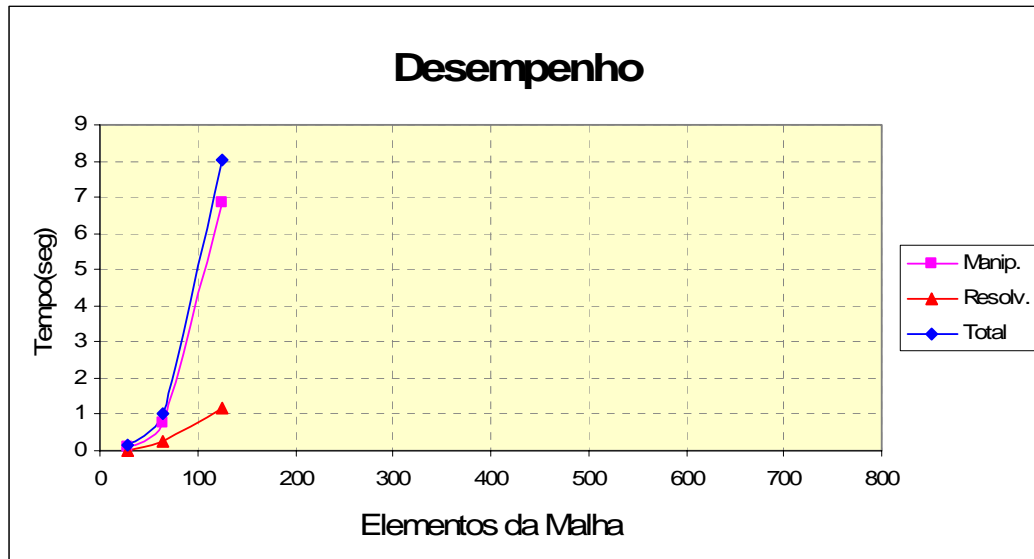


Figura 3.13 – Tempo empregado na manipulação e solução do sistema.

b) Teste de manipulação matricial por elementos

A Tabela 3.6 apresenta os resultados dos testes realizados com a implementação mediante a manipulação matricial por elementos. Nas Figuras 3.14 e 3.15 estes resultados são apresentados graficamente para uma melhor interpretação.

Como se pode observar nas Figuras 3.14 e 3.15, esta forma de resolver o sistema de equações tem um melhor desempenho que a técnica anterior.

A Figura 3.14 mostra que o uso de memória tem um crescimento exponencial suave, mas a Figura 3.15 mostra um crescimento exponencial em relação ao tempo requerido para resolver o sistema.

A Figura 3.15 mostra também que 90 % do tempo é empregado na manipulação, sendo o restante do tempo empregado na solução do sistema resultante.

Manipulação	Malha		Problema				Mem. (MB)	Tempo(seg)		
	Elem.	Nós	n	m	p	$m+p$		Manip.	Resolv.	total
Elementos	28	40	85	28	56	84	0.45	0.000	0.015	0.015
	64	81	193	64	128	192	0.81	0.078	0.031	0.109
	126	150	379	126	252	378	2.00	0.547	0.140	0.687
	225	256	676	225	450	675	7.00	3.044	0.438	3.482
	360	399	1081	360	720	1080	16.00	13.828	1.125	14.953
	500	546	1501	500	1000	1500	29.00	50.281	4.812	55.093
	750	806	2251	750	1500	2250				

Tabela 3.6 – Teste da manipulação matricial por elementos.

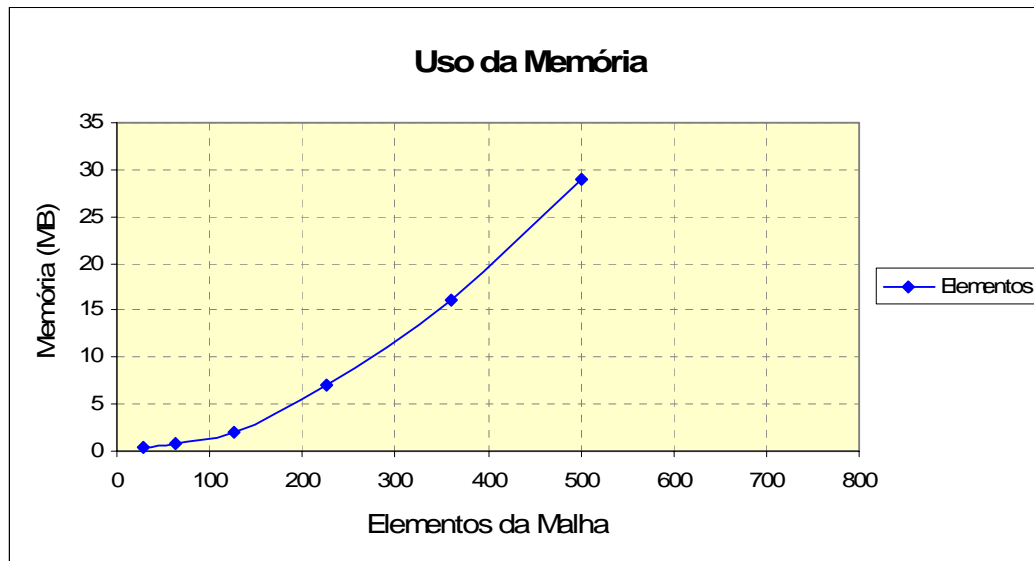


Figura 3.14 – Memória requerida para a solução do sistema.

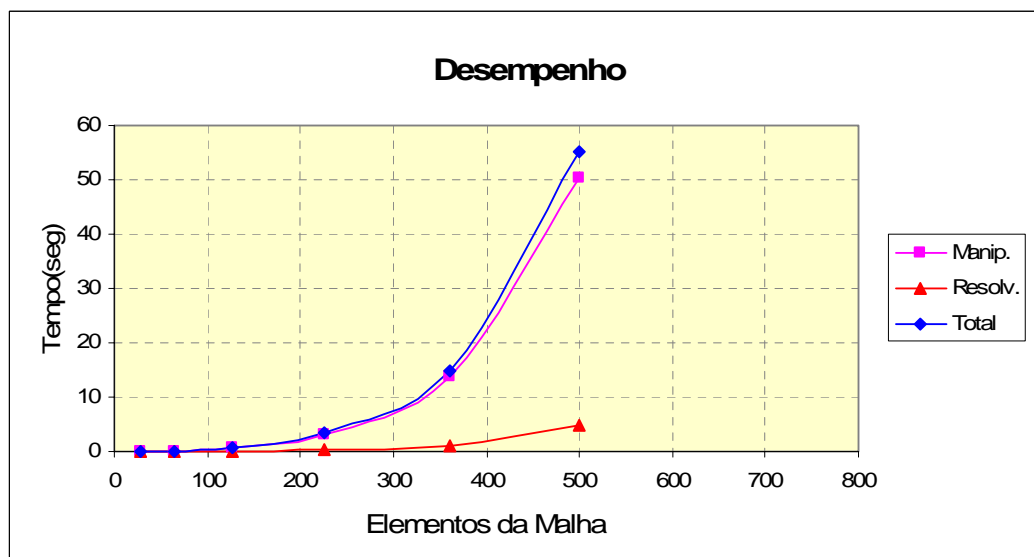


Figura 3.15 - Tempo empregado na manipulação e solução do sistema.

c) Teste de solução direta sem manipulação - proposta

Os resultados dos testes realizados com esta forma de resolver os sistemas de equações são apresentados na Tabela 3.7 e nas Figuras 3.16 e 3.17 estes resultados são apresentados graficamente.

Como esta forma de resolver o sistema de equações não tem manipulação, a coluna *Manip* da Tabela 3.7 tem todos os valores iguais a zero, sendo portanto o tempo total igual ao tempo empregado pelo resolvidor.

A Figura 3.16 indica que o requerimento de memória tem um crescimento linear com o número de elementos da malha. A Figura 3.17 mostra que o desempenho em termos de tempo requerido para resolver o sistema tem uma tendência quase linear com pequenas oscilações.

Estes resultados indicam que esta técnica tem muito melhor desempenho, tanto em relação ao uso da memória quanto em relação ao tempo requerido para resolver o sistema.

Manipulação	Malha		Problema				Mem. (MB)	Tempo(seg)		
	Elem.	Nós	n	m	p	$m+p$		Manip.	Resolv.	total
Direta	28	40	85	28	56	84	0.45	0	0.0930	0.093
	64	81	193	64	128	192	0.60	0	0.1250	0.125
	126	150	379	126	252	378	0.90	0	0.2340	0.234
	225	256	676	225	450	675	1.40	0	0.3750	0.375
	360	399	1081	360	720	1080	2.00	0	0.6870	0.687
	500	546	1501	500	1000	1500	2.70	0	0.9370	0.937
	750	806	2251	750	1500	2250	4.00	0	1.5620	1.562

Tabela 3.7 – Teste da solução direta sem manipulação.

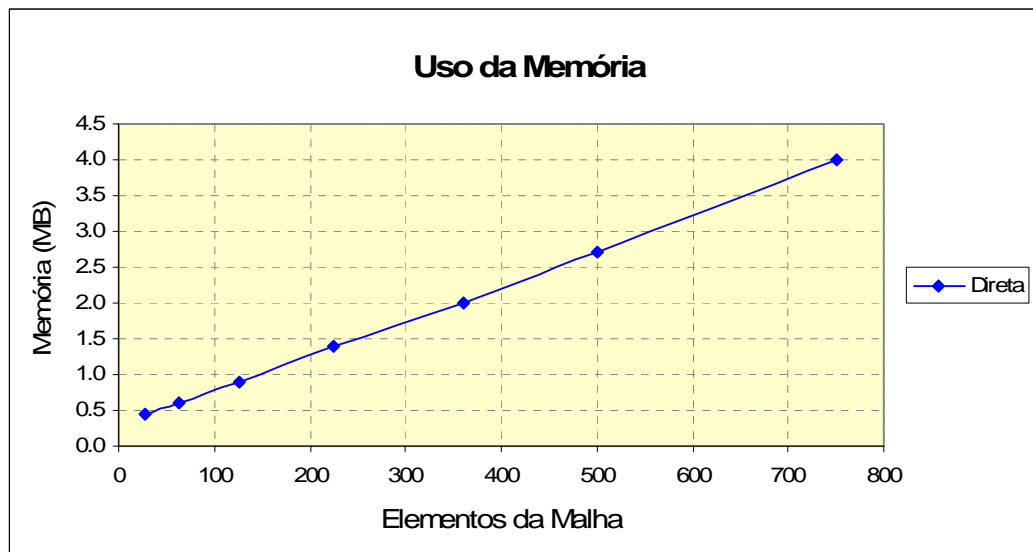


Figura 3.16 - Memória requerida para a solução do sistema.

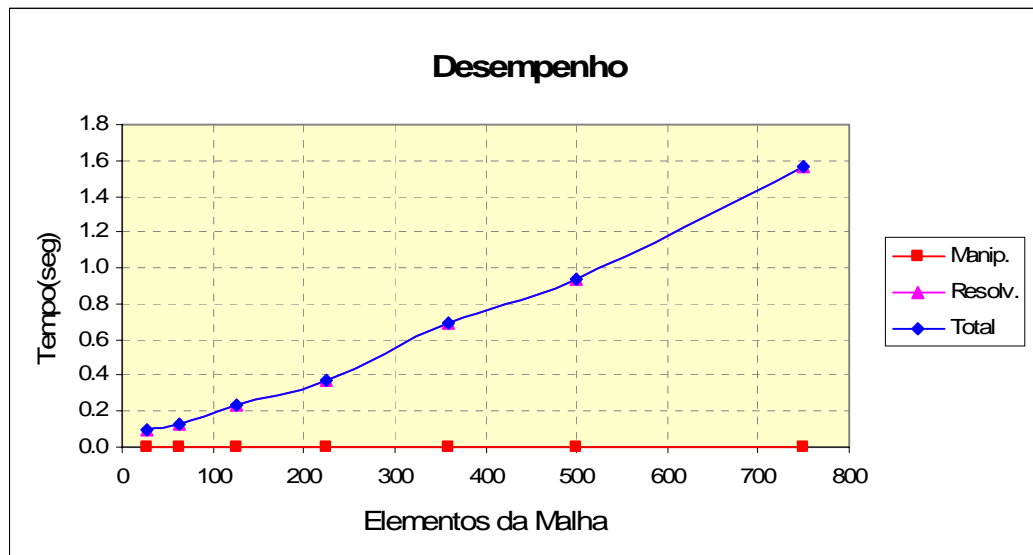


Figura 3.17 – Tempo empregado na solução do sistema.

d) Comparação das técnicas de solução

Nesta seção apresenta-se uma comparação dos testes realizados com as três técnicas implementadas para resolver sistemas de equações. A Tabela 3.8 apresenta um resumo dos testes realizados com cada uma das técnicas e as Figuras 3.18 e 3.19 apresentam as comparações do desempenho em termos de uso da memória e de tempo total requerido para manipular e resolver o sistema de equações.

A Figura 3.18 mostra que a solução direta proposta neste trabalho é muito mais eficiente em relação ao uso da memória do computador. A Figura 3.19 mostra também que a solução direta proposta neste trabalho é muito mais eficiente em comparação às duas técnicas encontradas na literatura pesquisada. O resultado mais importante mostrado pela Figura 3.19 é que o desempenho (em tempo) com a solução direta proposta, cresce quase linearmente com o refinamento da malha, este fato influencia diretamente no desempenho do otimizador implementado.

Manipulação	Malha		Problema				Mem. (MB)	Tempo(seg)		
	Elem.	Nós	n	m	p	$m+p$		Manip.	Resolv.	total
Matriz	28	40	85	28	56	84	0.54	0.125	0.015	0.140
	64	81	193	64	128	192	1.5	0.750	0.250	1.000
	126	150	379	126	252	378	47.00	6.844	1.172	8.016
	225	256	676	225	450	675				
	360	399	1081	360	720	1080				
	500	546	1501	500	1000	1500				
	750	806	2251	750	1500	2250				
Elementos	28	40	85	28	56	84	0.45	0.000	0.015	0.015
	64	81	193	64	128	192	0.81	0.078	0.031	0.109
	126	150	379	126	252	378	2.00	0.547	0.140	0.687
	225	256	676	225	450	675	7.00	3.044	0.438	3.482
	360	399	1081	360	720	1080	16.00	13.828	1.125	14.953
	500	546	1501	500	1000	1500	29.00	50.281	4.812	55.093
	750	806	2251	750	1500	2250				
Direta	28	40	85	28	56	84	0.45	0	0.093	0.093
	64	81	193	64	128	192	0.60	0	0.125	0.125
	126	150	379	126	252	378	0.90	0	0.234	0.234
	225	256	676	225	450	675	1.40	0	0.375	0.375
	360	399	1081	360	720	1080	2.00	0	0.687	0.687
	500	546	1501	500	1000	1500	2.70	0	0.937	0.937
	750	806	2251	750	1500	2250	4.00	0	1.562	1.562

Tabela 3.8 – Comparação de resultados com as três técnicas de solução.

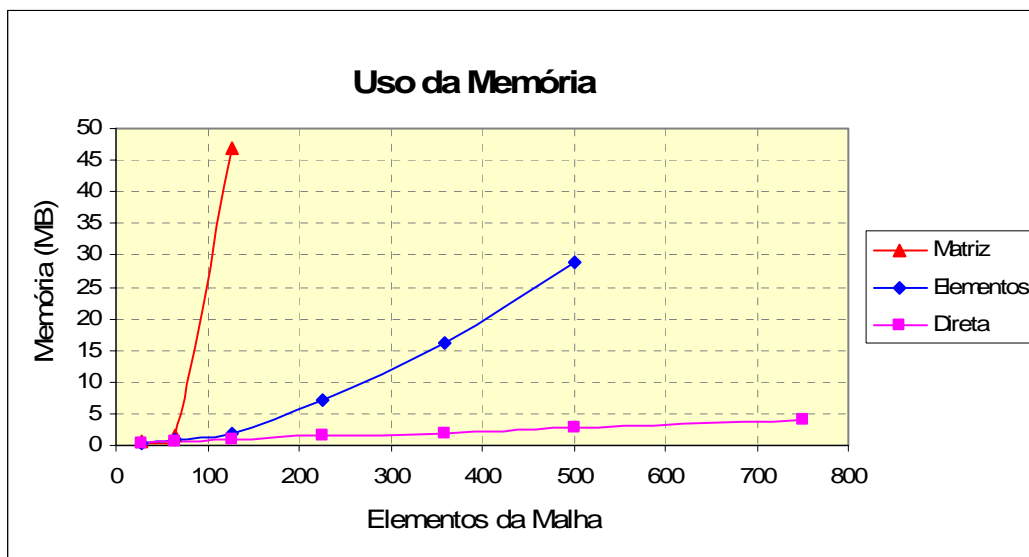


Figura 3.18 – Memória usada pelas técnicas.

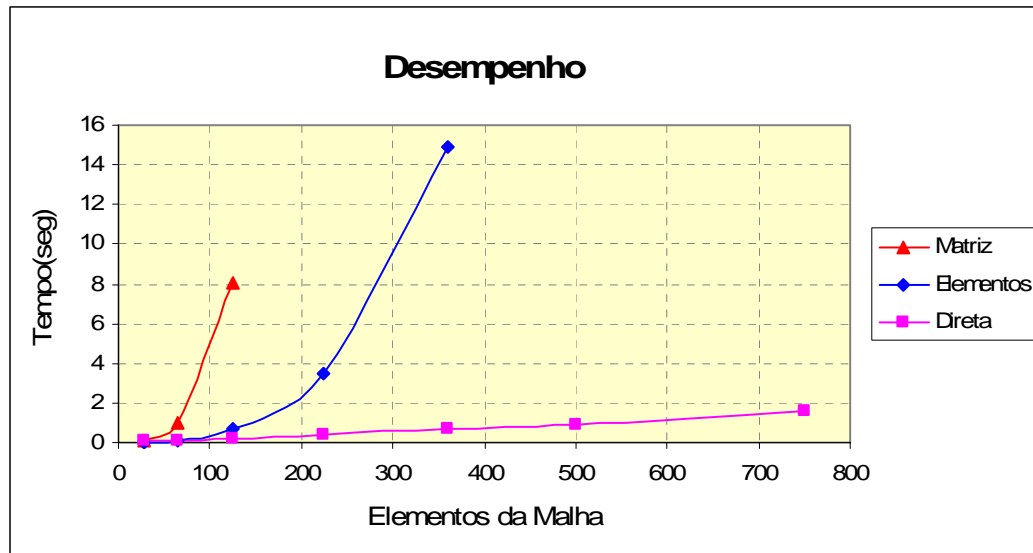


Figura 3.19 – Tempo empregado pelas técnicas.

3.2.9. Resolvedores Implementados

A eficiência do otimizador implementado depende muito da solução eficiente de sistema de equações lineares, por isso neste trabalho foram implementados e testados diferentes algoritmos diretos e indiretos que permitem resolver sistemas lineares da forma $Ax = b$.

Os primeiros testes do otimizador implementado foram feitos para resolver problemas pequenos com malha de 8 e 25 elementos usando o algoritmo de Gauss Jordan (GJ) (Hoffman, 1992 e Tsao, 1989). Nos testes para aplicações com malhas refinadas, o otimizador implementado usando Gauss-Jordan mostrou-se ineficiente, por isso, foram implementados e testados outros métodos de solução de sistemas lineares.

Entre os métodos diretos, o algoritmo de fatoração LL' de Cholesky (Ames, 2000 e Alkire, 2002) e fatoração LU de Doolittle (Gómez e Burguest, 2004) foram implementados e testados. Entre os métodos indiretos o método Quase Newton BFGS (Broyden-Fletcher-Goldfarb-Shanno) QN-BFGS (Bathe, 1996) e o método dos Gradientes Conjugados (CG) (Sandoval, 2006), foram implementados e testados também.

Uma das vantagens dos métodos indiretos é que são adequados para aplicar técnicas como tratamento da matriz esparsa e pré-condicionadores. Por isso,

foram implementados também o algoritmo dos Gradientes Conjugados com tratamento da matriz esparsa (CSR-CG) e o método Quase Newton BFGS com tratamento da matriz esparsa (CSR-QN-BFGS).

Dos testes realizados com os métodos implementados, o método dos Gradientes Conjugados com tratamento da matriz esparsa apresentou melhor desempenho, portanto neste trabalho somente é descrito o algoritmo dos Gradientes Conjugados. Os métodos diretos e o método BFGS foram implementados com base nas referências indicadas e pelo fato desses métodos serem amplamente conhecidos e difundidos na literatura, eles não foram apresentados no presente trabalho.

3.2.9.1. Método dos gradientes conjugados (CG)

O método dos Gradientes Conjugados, introduzido pela primeira vez por (Hestenes e Stifel, 1952), é uma técnica de otimização e tornou-se um dos mais populares métodos para a solução de sistemas lineares da forma $Ax = b$, onde o problema é resolvido como um problema de minimização de uma função objetiva quadrática $f(x)$ (Equação 3.55). A condição de primeira ordem para o mínimo de uma função $f(x)$ impõe que o gradiente da função objetivo $\nabla f(x) = Ax - b$ seja igual a zero, como mostra a Equação (3.56). Este método minimiza a função $f(x)$, onde o valor de x que minimiza a função é a solução do sistema $Ax = b$.

$$f(x) = \frac{1}{2} x^t Ax - b^t x \quad (3.55)$$

$$\nabla f(x) = Ax - b = 0 \quad (3.56)$$

O algoritmo de Gradiente Conjugado pré-condicionado (Figura 3.20), implementado no presente trabalho é a apresentado no trabalho de Sandoval em 2006 (Sandoval, 2006), onde M é a matriz pré-condicionadora.

\mathbf{x}^0
 Calcular $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$; $k = 0$
 Resolver $\mathbf{M}\mathbf{z}^0 = \mathbf{r}^0$; $\mathbf{p}^0 = \mathbf{z}^0$
 $\rho_0 = \mathbf{r}^{0T}\mathbf{z}^0$
 Enquanto (Não converge & $k < kmax$)
 $\mathbf{q}^k = \mathbf{A}\mathbf{p}^k$
 $\alpha_k = \rho_k / \mathbf{p}^{kT}\mathbf{q}^k$
 $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k\mathbf{p}^k$
 $\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k\mathbf{q}^k$
 resolver $\mathbf{M}\mathbf{z}^{k+1} = \mathbf{r}^{k+1}$
 $\rho_{k+1} = \mathbf{r}^{k+1T}\mathbf{z}^{k+1}$
 $\beta_k = \rho_{k+1} / \rho_k$
 $\mathbf{p}^{k+1} = \mathbf{z}^{k+1} + \beta_k\mathbf{p}^k$
 $k = k + 1$
 Terminar

Figura 3.20 – Algoritmo de Gradiente Conjugado pré-condicionado (Sandoval, 2006).

3.2.9.2.

Teste de desempenho de resolvedores implementados

Os testes de eficiência dos métodos implementados foram feitos para os problemas mostrados pela Figura 3.21. Na Tabela 3.9 são apresentados os resultados dos testes dos métodos com melhor desempenho. Estes resultados são mostrados graficamente na Figura 3.22. Da Tabela 3.9 pode-se observar que o método Quase Newton BFGS mostra-se ineficiente. Por isso não foram realizados mais testes e não foram apresentados resultados com esse algoritmo na Figura 3.22.

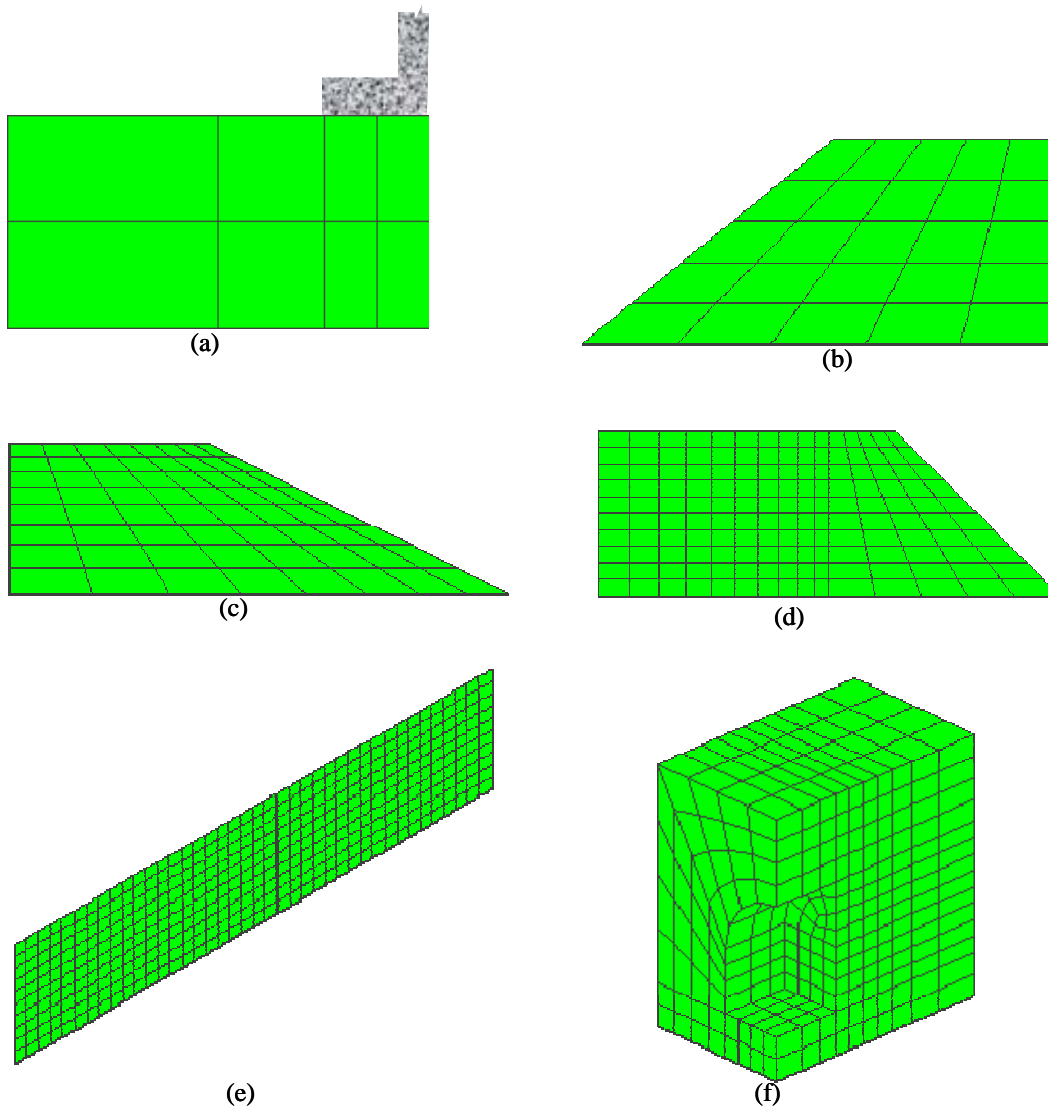


Figura 3.21 – Malhas: (a) 8, (b) 25, (c) 64, (d) 150, (e) 400 e (f) 676 elementos.

A Figura 3.22 mostra que o método baseado na decomposição LU tem um melhor desempenho para sistemas com número de variáveis N menores que 2400 (aproximadamente), mas para N maiores que 2400 o método de eliminação de Gauss-Jordan (GJ) tem melhor eficiência.

Técnica de tratamento de matriz esparsa foi aplicada aos métodos indiretos, para o qual o método de Gradiente Conjugado (CG) apresenta um ganho de eficiência como discutido na seção 3.3.1. Pré-condicionadores também foram implementados e aplicados no método de Gradiente Conjugado para melhorar a eficiência como descrito na seção 3.3.2.

Malha	N	Tempo (seg)			
		GJ	LU	QN_BFGS	CG
a	47	0.000	0.000	0.0	0.0
b	151	0.063	0.032	1.0	0.0
c	385	0.829	0.547	5.0	5.0
d	901	10.219	9.656	55.0	9.0
e	2381	284.672	185.984		994.0
f	6460	4854.000	6492.000		5958.0

Tabela 3.9 – Comparação do desempenho dos métodos implementados.

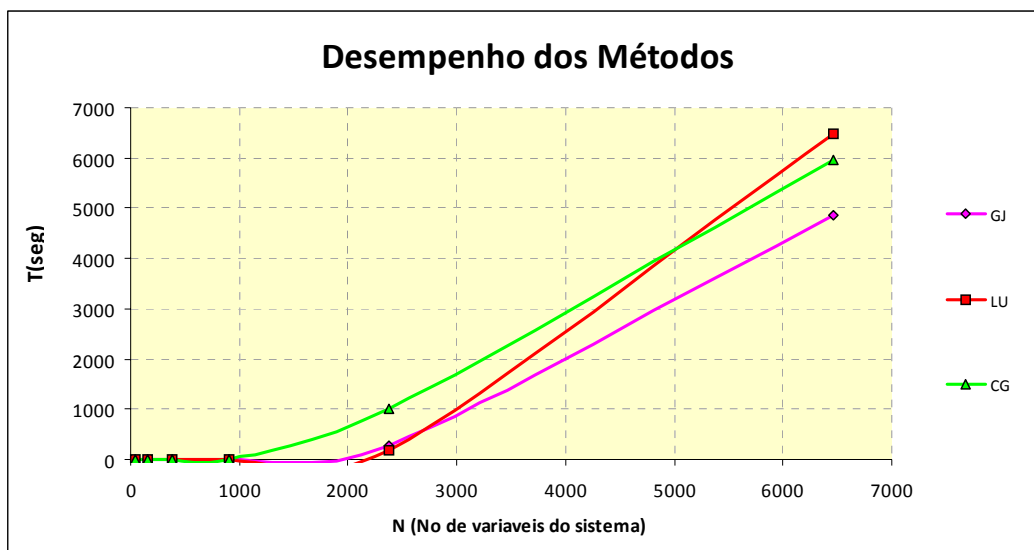


Figura 3.22 – Comparação do desempenho dos métodos implementados.

3.2.10. Resolvedor SAMG Testado

Na procura por encontrar um método eficiente para resolver os sistemas lineares na implementação do otimizador, neste trabalho tentou-se usar o resolvedor comercial SAMG(Algebraic Multigrid Methods for System) desenvolvido pelo Institute of Algorithms and Scientific Computation (Klaus & Tonja, 2005). Este programa usa o Método Multigrid para resolver sistema lineares.

A tentativa de resolver o problema expresso pelas Equações 3.53 e 3.54 usando este programa não foi bem sucedida.

3.3. Melhora do desempenho

Como o desempenho do otimizador implementado depende do desempenho dos algoritmos que servem para resolver o sistema de equações lineares, neste trabalho foram pesquisadas, implementadas e testadas técnicas como tratamento da matriz esparsa e uso de pré-condicionadores. Estas técnicas são descritas a seguir.

3.3.1. Tratamento de Matriz Esparsa

Uma matriz esparsa é aquela que apresenta muitos elementos iguais a zero, e elementos diferentes de zero podem ser armazenados em uma estrutura de dados especial ou em vetores (Tsal, 1998).

Existem várias técnicas ou formatos para tratamento de uma matriz esparsa com vetores, neste trabalho usou-se o formato CSR (Compressed Sparse Row) para armazenar a matriz de coeficientes em vetores.

3.3.1.1. Formato CSR(Compressed Sparse Row)

Neste formato, uma matriz esparsa é representada por 3 vetores como é mostrado na Figura 3.23a (SMAILBEGOVIC et all, 2006), onde AN é um vetor com os valores diferentes de zero da matriz A , AJ é um vetor com os índices das

colunas da matriz A e AI são os limites dos índices de colunas do vetor AN para cada fila de A .

Neste trabalho, a matriz de coeficientes das Equações (3.53) e (3.54) são armazenados em vetores no formato CSR. A escolha de este formato foi porque é adequado para realizar o produto de uma matriz por um vetor e sua implementação é simples demais como mostrado pelo código na Figura 3.23b.

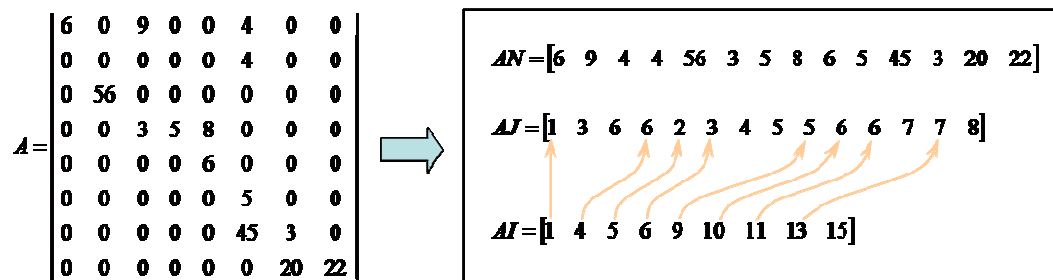


Figura 3.23a – Armazenamento da matriz esparsa (SMALBEGOVIC et all, 2006).

```
void Sparse::CSRxVect( CSR& A, Vector& d, Vector& Ad)
{  Nat i,j,j0,jf;
   for(i=0;i<d.getSize();i++)
   {  Ad[i]=0.0;j0 = A.r[i]; jf = A.r[i+1];
      for(j=j0;j<jf;j++) Ad[i] += A.val[j]*d[A.c[j]];
   }
}
```

Figura 3.23b – Produto de uma matriz esparsa A por um vetor d .

3.3.1.2.

Teste de desempenho de CG com tratamento da matriz esparsa

A Tabela 3.10 apresenta os resultados dos testes feitos com os métodos de Gradiente conjugado CG e o método Quase-Newton BFGS com tratamento da matriz esparsa pelo formato CSR. Observa-se que com o método Quase Newton BFGS obteve-se uma melhora na eficiência de 50% aproximadamente e com o método de Gradiente Conjugado CG a melhora de eficiência foi ainda maior.

Na Figura 3.24 são apresentados graficamente os resultados dos testes, onde se pode visualizar o ganho obtido pelo método de Gradiente Conjugado com tratamento da matriz esparsa CSR-CG. É de se esperar que este ganho influencie diretamente na eficiência do otimizador implementado.

Malha	N	Tempo (seg)					
		GJ	LU	QN_BFGS	CSR+QN_BFGS	CG	CSR_CG
a	47	0.000	0.000	0.0	0.00	0.0	0.000
b	151	0.063	0.032	1.0	0.00	0.0	0.000
c	385	0.829	0.547	5.0	3.00	5.0	0.600
d	901	10.219	9.656	55.0	27.00	9.0	1.400
e	2381	284.672	185.984			994.0	12.000
f	6460	4854.000	6492.000			5958.0	359.000

Tabela 3.10 – Desempenho dos métodos com o tratamento da matriz esparsa.

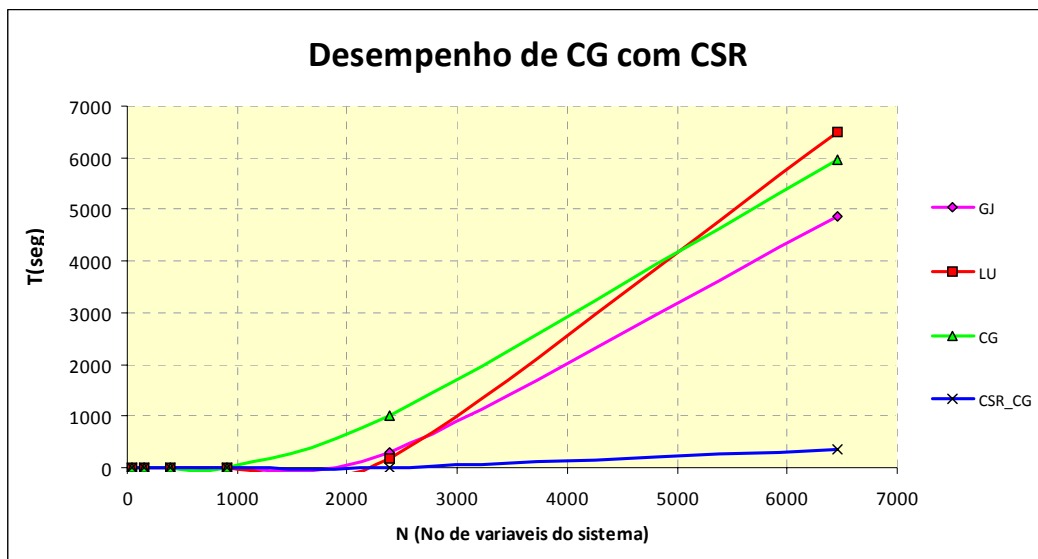


Figura 3.24 – Desempenho do método CG com tratamento da matriz esparsa CSR.

3.3.2. Precondicionamento

A idéia do precondicionamento é transformar um sistema linear $Ax = b$ em outro equivalente com condições espectrais mais favoráveis, onde o número de iterações requeridas para a convergência é reduzido (Cervantes & Mejía, 2004).

Pré-condicionadores são técnicas utilizadas para acelerar a convergência da solução de sistemas lineares (Equação 3.57) pelo método de Gradiente Conjugado (CG) e, portanto aumenta o desempenho deste método em termos de tempo.

$$Ax = b \quad (3.57)$$

Neste trabalho foram implementados e testados três pré-condicionadores encontrados na literatura e o uso de pré-condicionadores mistos proposto neste trabalho.

Os pré-condicionadores implementados e testados são Escala Diagonal(DS) (Pini e Gambolati, 1990), Escala Simétrica (SS) (Jennings e Malik, 1978) e Fatoração Incompleta de Cholesky (ICF) (Meijerink e Van der Vorst, 1977). Os seguintes pré-condicionadores mistos são propostos neste trabalho DS+SS e ICF+SS. É importante mencionar que os pré-condicionadores mistos não aparecem na literatura e foi uma tentativa deste trabalho de encontrar um pré-condicionador eficiente.

Campos (1999) indica que, “o condicionamento implica em alterar a matriz de coeficientes do sistema, fazendo com que os autovalores desta matriz sejam mais próximos e, logo, o sistema mais estável, reduzindo o número de iterações necessárias para a solução do mesmo.

3.3.2.1. Escala Diagonal (DS)

A escala diagonal é um pré-condicionador muito simples, fácil de ser obtido e implementado. A matriz pré-condicionadora M apresentada no algoritmo da Figura 3.20 é uma matriz diagonal cujos elementos são formados pela diagonal da matriz de coeficientes, como mostrado na Equação 3.58 (Pini e Gambolati, 1990).

$$M = \text{diag}(A) \quad (3.58)$$

3.3.2.2. Escala Simétrica (SS)

Este pré-condicionador foi apresentado por Jennings e Malik (1978) e consiste em escalonar o sistema linear apresentado pela Equação 3.57 como mostra a Equação 3.59. A matriz pré-condicionadora D é uma matriz diagonal cujos elementos são obtidos como indicado na Equação 3.60.

$$DADD^{-1}x = Db \quad (3.59)$$

$$d_{ii} = (a_{ii})^{-0.5} \quad (3.60)$$

Da Equação 3.59 para $B = DAD$, $y = D^{-1}x$ e $c = Db$ a Equação 3.52 é transformada para a Equação 3.61. A Equação 3.61 é resolvida pelo método do Gradiente Conjugado (Figura 3.20) com a matriz $M = I$.

$$By = c \quad (3.61)$$

Conhecido o vetor y , o vetor x é facilmente determinado pela relação $x = Dy$.

3.3.2.3. Fatoração Incompleta de Cholesky (ICF)

A fatoração incompleta de Cholesky foi proposta por Meijerink e Van der Vorst (1977). Este pré-condicionador baseia-se na decomposição de Cholesky para resolver sistemas lineares (Equação 3.57), onde a matriz de coeficientes A é decomposta como o indicado na Equação 3.62.

$$A = L^T L \quad (3.62)$$

onde L é uma matriz triangular inferior. Esta decomposição na sua forma original, requer do cálculo da raiz quadrada e dos elementos da diagonal, para evitar esse cálculo, Meijerink e Van der Vorst (1977) apresentam o algoritmo de Cholesky modificado como mostra a Equação 3.63.

$$A = L^T D L \quad (3.63)$$

onde D é uma matriz diagonal e as matrizes L e D são obtidas como mostrado nas expressões:

$$\begin{aligned} L_{ji} &= A_{ji} - \sum_k^{(i-1)} L_{jk} L_{ik} D_{kk} / j = i, N \\ D_{ii} &= \frac{1}{L_{ii}} \end{aligned} \quad (3.64)$$

No processo de decomposição, a matriz A perde o padrão de distribuição de zeros, ou seja, elementos nulos podem tornar-se não nulos após a decomposição. Para que a fatoração seja incompleta deve-se manter a mesma distribuição de zeros da matriz original. Isto é conseguido fazendo-se com que os elementos nulos na matriz original permaneçam nulos na matriz fatorizada.

Benzi & Tuma em 2001 indicam que uma fatoração incompleta pode falhar para uma matriz geral de tipo esparsa positiva e definida (SPD) devido à ocorrência de pivôs não positivos (Benzi & Tuma, 2001).

Na literatura pesquisada encontraram-se trabalhos indicando que o pré-condicionador ICF tem o melhor desempenho, entre eles Pinheiro (1998), Campos (1999).

3.3.2.4. Pré-condicionadores mistos - proposto

Neste trabalho propõe-se o uso de pré-condicionadores mistos para melhorar o desempenho do método do gradiente conjugado na solução de sistemas lineares. Na tentativa de encontrar um pré-condicionador eficiente para os tipos de problema a serem resolvidos neste trabalho, foram implementados e testados dois tipos de pré-condicionadores mistos. A escala diagonal misturada com escala simétrica (DS+SS) e a escala diagonal misturada com fatoração incompleta de Cholesky (DS+ICF).

Os pré-condicionadores mistos são fáceis de ser implementados. Para o pré-condicionador DS+SS, primeiro é aplicado o pré-condicionador SS e logo o DS. Similarmente para o pré-condicionador DS+ICF, primeiro aplica-se o ICF e logo o DS.

Para o tipo de problema deste trabalho, o tipo de pré-condicionador que mostrou o melhor desempenho é o pré-condicionador misto proposto DS+SS como indicam os resultados dos testes realizados (Figura 3.25).

3.3.2.5. Teste de Desempenho de Pré-condicionadores

A Tabela 3.11 apresenta os resultados obtidos com os pré-condicionadores implementados. Na Figura 3.25 estes resultados são ilustrados graficamente, onde

pode-se observar que o método do gradiente conjugado com pré-condicionador misto DS+SS teve melhor desempenho para nosso tipo de problema.

Malha	N	Temp(seg)					
		CSR+CG	DS+CSR+CG	SS+CSR+CG	ICF +CSR+CG	DS+SS+SCR+CG	ICF+SS+SCR+CG
a	47	0.0	0.000	0.000	0.000	0.000	0.000
b	151	0.0	0.016	0.016	0.046	0.015	0.047
c	385	0.6	0.063	0.063	0.437	0.063	0.375
d	901	1.4	0.188	0.109	0.297	0.157	0.281
e	2381	12.0	4.781	3.109	1.719	2.172	1.407
f	6460	359.0	101.607	104.297	444.734	82.266	409.437

Tabela 3.11 – Teste de pré-condicionadores implementados.

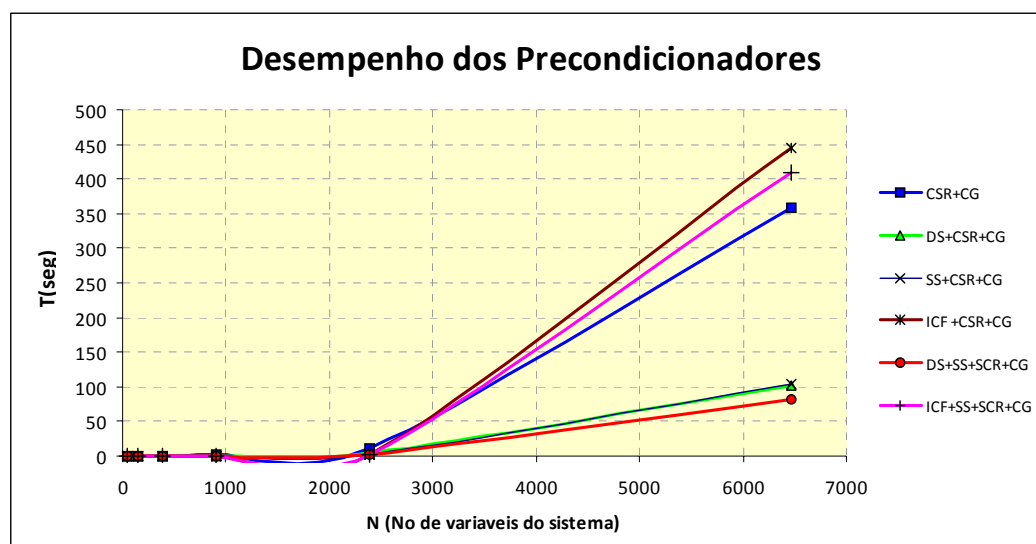


Figura 3.25 – Desempenho de CG com os pré-condicionadores implementados.

3.4. Teste de desempenho do Otimizador Implementado

Com o objetivo de ter uma idéia comparativa do desempenho do otimizador implementado, o mesmo problema em 2D da Figura 3.1 com malhas da Figura 3.2 foi analisado usando o otimizador implementado e os resultados são comparados com os otimizadores Lingo e Minos. Outro problema em 3D com resultados da análise já conhecidos (obtidos usando o otimizador MINOS 5.5) é analisado e a solução é obtida com o otimizador implementado.

3.4.1. Teste com problema em 2D

O problema da Figura 3.1 com malhas da Figura 3.2 que com o qual os programas Lingo e Minos foram testados, foi analisado utilizando o otimizador implementado.

Os resultados obtidos com o programa Geolima versão 2.0 cujo otimizador foi implementado neste trabalho são apresentados na Tabela 3.12. Para fim de comparação, estes resultados são apresentados graficamente nas Figuras 3.26, 3.27, 3.28 e 3.29, onde se observa que o programa Geolima com otimizador implementado é mais eficiente tanto no uso de memória quanto no tempo de otimização.

Malha		Lingo				Minos				Geolima			
Elem.	Nós	Mem	α	iter.	t(seg)	Mem	α	iter.	t(seg)	Mem	α	iter.	t(seg)
28	40	0.20	3.76732	93	0	253	3.767329	225	1	0.45	3.766621	23	2.422
64	81	0.45	3.58688	245	2	253	3.586883	599	4	0.60	3.584825	24	5.484
126	150	0.88	3.56612	482	13	253	3.566153	1602	6	0.90	3.560147	31	11.92
225	256	1.67	3.49650	857	110	253	3.497155	3556	16	1.40	3.492299	39	27.48
360	399	2.79	3.47713	1492	775	253	3.478615	6561	72	2.00	3.471841	27	27.97
500	546	3.91	3.46887	3524	2305	253	3.470797	10182	202	2.70	3.463164	28	62.67
750	806	5.67	0.027280	43	2453	295	3.453981	18067	671	4.00	3.447660	22	92.72

Tabela 3.12 – Resultados do teste em 2D do otimizador implementado.

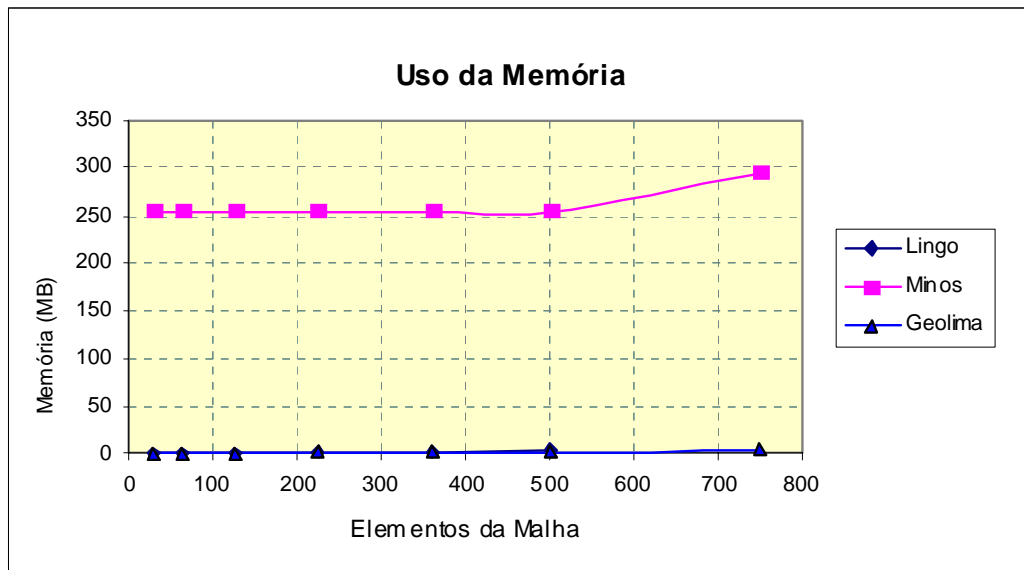


Figura 3.26 – Comparação de uso da memória pelos otimizadores.

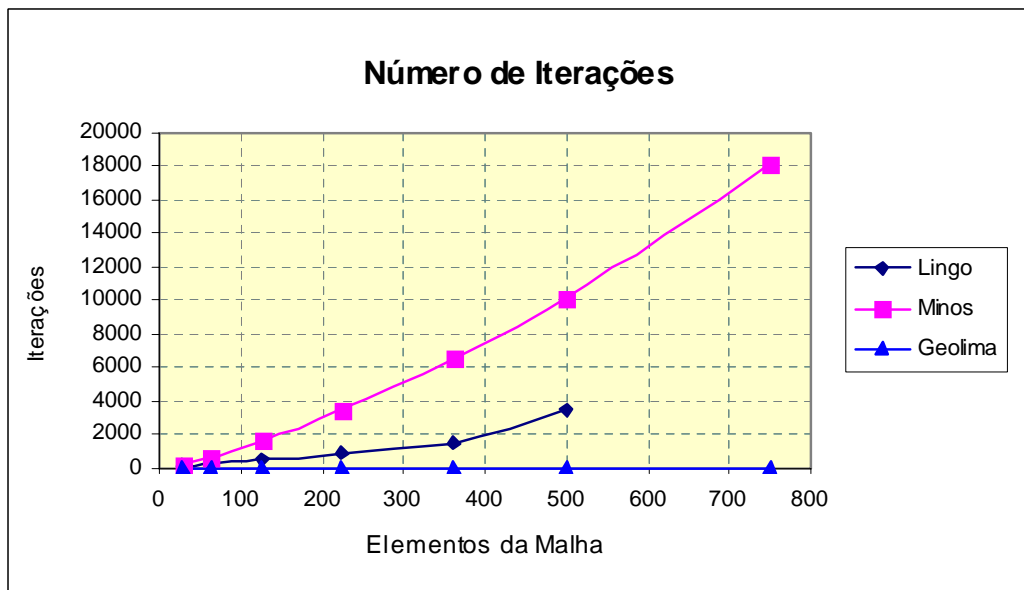


Figura 3.27 – Comparação de número de iterações.

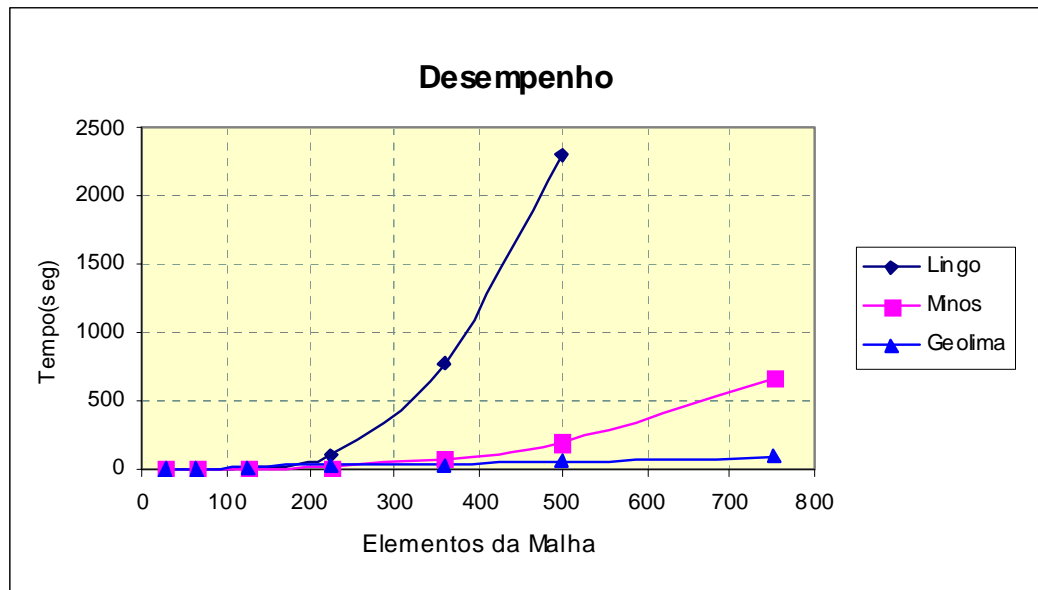


Figura 3.28 – Comparação do desempenho dos otimizadores.

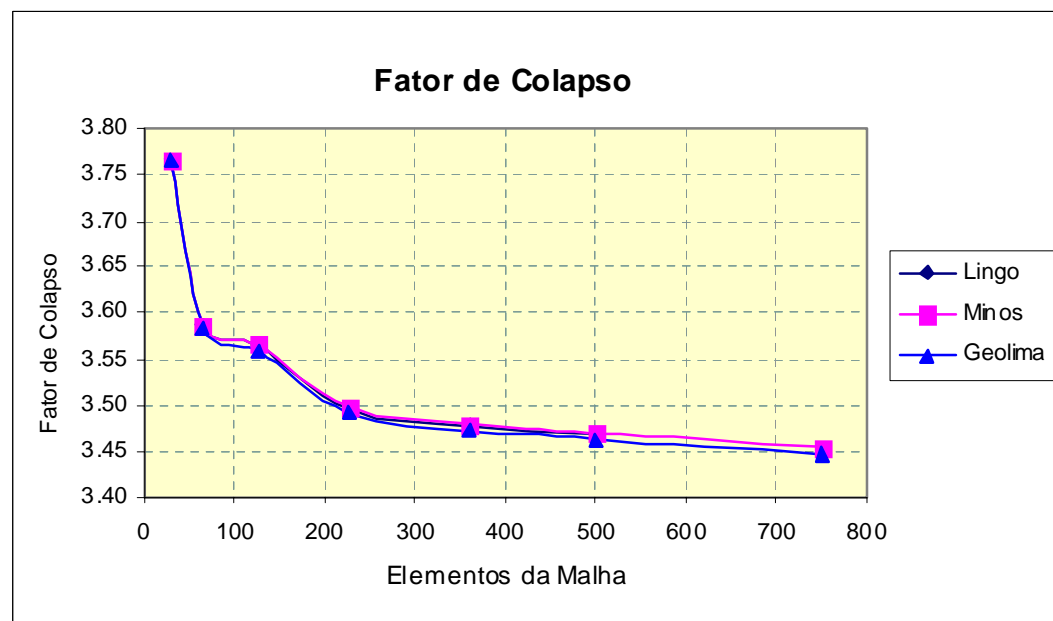


Figura 3.29 – Variação de fator de colapso.

3.4.2. Teste com problema em 3D

Com a finalidade de validar o otimizador implementado, um problema 3D de frente de escavação de túneis (Figura 3.30) é analisado pelo programa GEOLIMA 2.0 com seu próprio otimizador implementado. Este problema é interessante, porque se tem resultados da simulação física do comportamento de frente de escavação, feitas mediante ensaios de laboratório em modelos 3D em escala reduzida. A simulação física foi feita no Laboratório da Mitsubishi Heavy Industries Ltda., Japão (Sterpi et al., 1996). Este problema foi usado também para validar os resultados da Análise Limite com GEOLIMA 1.0 implementada na dissertação de mestrado e que usava o otimizador Minos 5.5.

Neste trabalho, a análise deste problema é feita com GEOLIMA 2.0 e os resultados são comparados com os obtidos com o otimizador MINOS.

A análise foi feita mantendo todas as condições, ou seja para a mesma geometria (Figura 3.30), a mesma malha (Figura 3.31) (676 elementos com 945 nós), as mesmas condições de contorno, as mesmas propriedades do material coesão ($C=50 \text{ kN/m}^2$), ângulo de atrito ($\phi = 5^\circ$) e peso específico ($\gamma = 19.5 \text{ kN/m}^3$); o mesmo critério de escoamento Drucker-Prager (parâmetros de aproximação do círculo superior); com um carregamento inicial de $\gamma_o = 1.0 \text{ kN/m}^3$ e usando o mesmo computador (Pentium IV com dois processadores de 3.07 GHz e memória RAM de 1.4 Gb).

O problema a ser resolvido pelo otimizador tem um total de 4057 variáveis, 1727 restrições lineares e 676 restrições não lineares. A ordem do sistema de equações lineares a ser resolvido em cada iteração é de 6460.

O tempo requerido pelo Minos foi de 16 horas 43 minutos e o fator de colapso obtido foi de $\alpha = 25.012$.

O tempo requerido pelo otimizador implementado foi de 17 minutos com 49 seg. e o fator de colapso obtido foi de $\alpha = 24.6685$. A Figura 3.32 apresenta o mecanismo de ruptura do problema obtido pela Análise Limite com GEOLIMA 2.0 e é muito similar à obtida com Minos e também à obtida ao modelo físico (Figura 3.33).

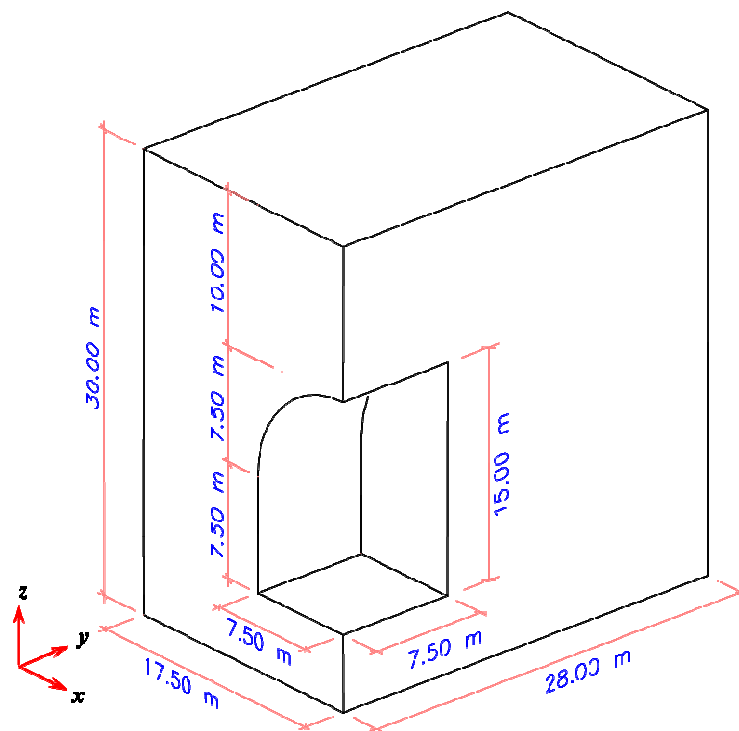


Figura 3.30 – Geometria da estrutura a ser analisada.

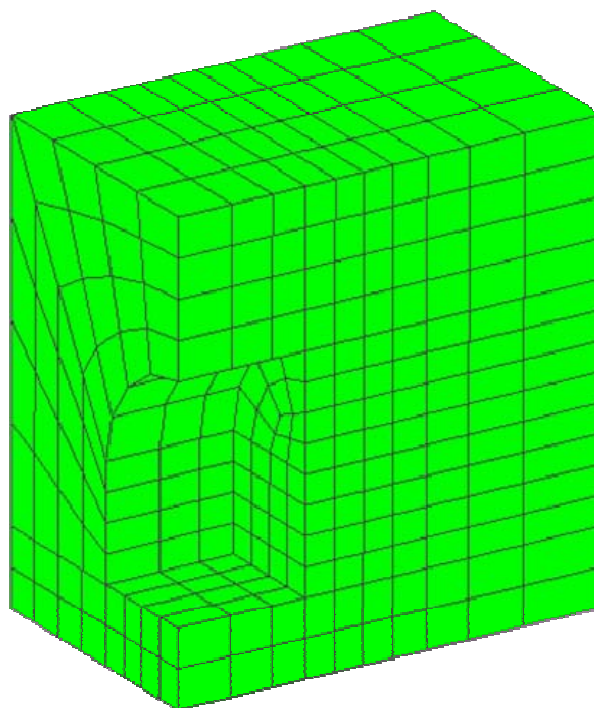


Figura 3.31 – Malha de elementos finitos (676 elementos e 945 nós).

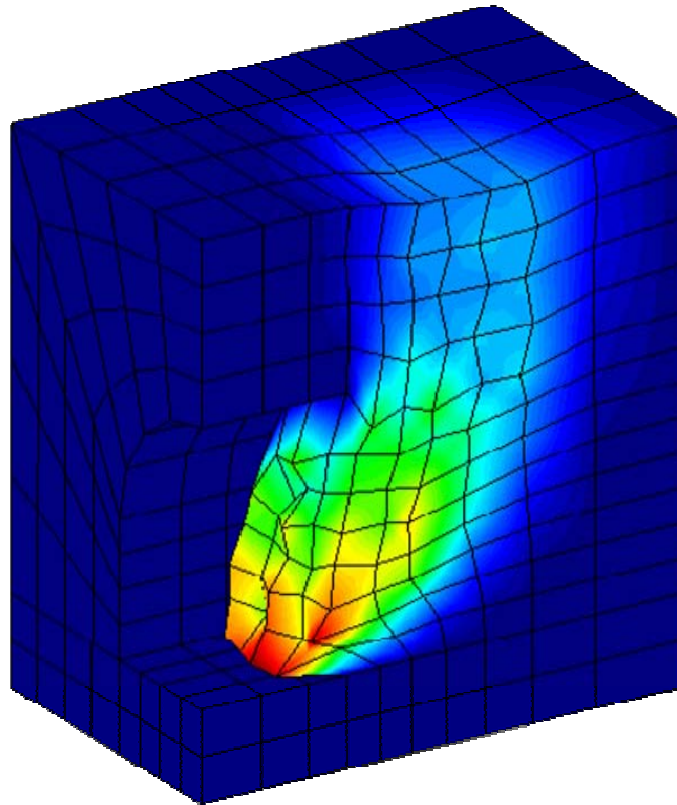


Figura 3.32 – Mecanismo de colapso da estrutura obtido pelo programa GEOLIMA 2.0.

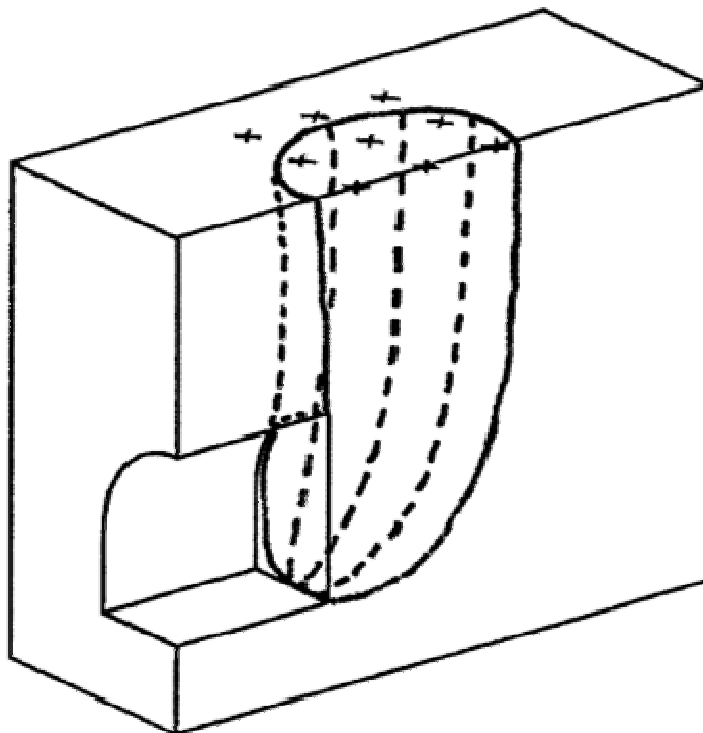


Figura 3.33 – Mecanismo de ruptura obtido a partir de ensaios em modelo físico em escala reduzida (Sterpi, 1996)