UNIVERSIDADE FEDERAL DE SANTA CATARINA

ESTUDO DE MÉTODOS ITERATIVOS NÃO-ESTACIONÁRIOS DE RESOLUÇÃO DE GRANDES SISTEMAS LINEARES ESPARSOS

AUTOR: LUIZ FERNANDO SPILLERE DE SOUZA

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

TÍTULO: ESTUDO DE MÉTODOS ITERATIVOS NÃO-ESTACIONÁRIOS DE RESOLUÇÃO DE GRANDES SISTEMAS LINEARES ESPARSOS

AUTOR: LUIZ FERNANDO SPILLERE DE SOUZA

ORIENTADOR: PROF. JAÚBER C. DE OLIVEIRA

BANCA EXAMINADORA: PROF. DANIEL SANTANA DE FREITAS

PROF. JÚLIO FELIPE SZEREMETA

Florianópolis, 02 de maio de 2002

RESUMO

Este trabalho tem por objetivo comparar vários precondicionadores para os sistemas lineares esparsos resultantes da aplicação do método dos elementos espectrais a equações diferenciais parciais. Após introduzir os métodos iterativos estacionários, exploramos os métodos iterativos não estacionários, com ênfase nos gradientes conjugados.

ABSTRACT

We present comparisons among several preconditioning techniques for conjugate gradients used for solving sparse linear systems obtained from spectral element PDE solvers. After introducing the stationary iterative methods, we explore the nonstationary iterative methods, with emphasis in the conjugate gradients.

SUMÁRIO

OBJETIVOS GERAIS

O objetivo deste trabalho é estudar os **métodos numéricos** que contemplem fundamentalmente a **resolução de sistemas lineares**, focalizando no aspecto da **implementação computacional**.

Para isso, após uma breve apresentação dos **métodos diretos**, pretendemos **inicialmente** estudar os **métodos iterativos clássicos**, com a finalidade de familiarizar-nos com a aplicação de métodos iterativos para a resolução de sistemas lineares **esparsos.**

Após este estudo **inicial**, pretendemos estudar os denominados **métodos não-estacionários** a fim de investigar as diferentes alternativas dos **métodos baseados em gradientes** visando resolver eficientemente **grandes sistemas lineares esparsos** que resultam da solução numérica de **equações diferenciais.**

Nesse contexto, devemos também nos preocupar com as formas de **armazenamento de matrizes** e o **pré-condicionamento** das mesmas, para que possamos aplicar estes métodos com a garantia de sua convergência.

1. RESOLUÇÃO DE SISTEMAS LINEARES POR MÉTODOS DIRETOS

Nos métodos diretos, a solução é obtida por uma seqüência finita de operações, que produziria a solução exata se não houvesse os erros de arredondamento decorrentes da aritmética de ponto flutuante. A exigência de obtermos uma solução correta traz à luz a questão dos erros de representação finita dos números reais e de como estes se propagam através dos cálculos. A minimização do número de operações é desejável tanto a nível de reduzir o tempo computacional necessário para obter-se a solução do sistema, quanto no sentido de evitar maiores propagações de erros de representação. Neste aspecto, tirar vantagem da estrutura da matriz é o ponto central no desenvolvimento dos algoritmos.

Vamos então, apresentar rapidamente os mais usuais métodos diretos para solução numérica de sistemas lineares. Esses métodos são conhecidos como métodos de eliminação e métodos de fatoração.

1.1. Método de Eliminação de Gauss

Dado um sistema linear qualquer da forma:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

o método de Gauss consiste em fazer operações entre linhas deste sistema até chegarmos a um novo sistema (que terá a mesma solução que o inicial) com a forma triangular:

$$\begin{cases} a'_{11}x_1 + a'_{12}x_2 + \cdots + a'_{1n}x_n = b'_1 \\ \quad\quad + a'_{22}x_2 + \cdots + a'_{2n}x_n = b'_2 \\ \quad\quad\quad\quad\quad\quad\quad\quad \vdots \\ \quad\quad\quad\quad\quad\quad\quad\quad a'_{nn}x_n = b'_n \end{cases}$$

Claro que a determinação da solução do sistema linear triangular é óbvia e direta: o valor de x_n já está determinado na última equação. O valor de x_{n-1} será determinado na penúltima, substituindo-se o valor de x_n; o valor de x_{n-2} será obtido na penúltima linha com os valores já determinados de x_n e x_{n-1}, e assim por diante até determinamos x_1. Então, desde que se consiga levar o sistema linear inicial na forma triangular sem alterar suas soluções, o nosso problema está resolvido.

1.2. Método de Eliminação de Gauss-Jordan

Este método é uma complementação ao método de Gauss. Ele transforma o sistema dado em um outro diagonal, isto é, onde todos os elementos fora da diagonal são nulos. O método de Gauss exigia apenas que se chegasse à forma triangular.

Sabemos que o um sistema linear pode ser escrito na forma de uma matriz aumentada:

$$\begin{pmatrix} a_{11} & a_{12}\cdots a_{1n} & \vdots & b_1 \\ a_{21} & a_{22}\cdots a_{2n} & \vdots & b_2 \\ & & & \\ a_{n1} & a_{n2}\cdots a_{nn} & \vdots & b_n \end{pmatrix}$$

Então procuraremos chegar a outra matriz, com os passos do método de eliminação de Gauss, com a forma diagonal:

$$\begin{pmatrix} a_1 & 0\cdots 0 & \vdots & \beta_1 \\ 0 & a_2\cdots 0 & \vdots & \beta_2 \\ & & & \\ 0 & 0\cdots a_n & \vdots & \beta_n \end{pmatrix},$$

que será associada a um sistema na forma:

$$x_1 = \frac{\beta_1}{\alpha_1} \qquad x_2 = \frac{\beta_2}{\alpha_2} \qquad x_1 = \frac{\beta_3}{\alpha_3}$$

Isto quer dizer que se conseguirmos chegar da matriz aumentada para a forma diagonal, a solução do sistema é imediata. Podemos, ainda, dividindo cada linha pelo elemento da diagonal, chegar à matriz identidade.

1.3. Método de Gauss-Jordan na Inversão de Matrizes

Uma análise rápida do método de eliminação de Gauss nos mostra que, ao invés de resolvermos um sistema linear completo, resolvemos, na realidade, um sistema triangular. Isso quer dizer que, a partir do sistema linear dado, geramos outro, que conserva suas soluções originais.

De um modo mais formal, podemos dizer que saímos de um sistema geral

$$Ax = b$$

e chegamos a outro sistema:

$$A'x = b'$$

cuja matriz A' é triangular, mas as soluções são as mesmas que do sistema linear inicial.

Naturalmente a passagem de uma forma à outra não pode ser arbitrária. Para termos a garantia de que as raízes são preservadas só permitimos operações elementares sobre as linhas do sistema. Cada uma dessas operações, equivale à multiplicação da matriz do sistema por uma matriz elementar.

1.4. Método de Crout

Observamos que no caso do método de Gauss, podemos transformar uma dada matriz A em outra A', triangular superior. Vamos agora, explicitar o que acontece com as matrizes definidas anteriormente, para obter um novo método direto, chamado método compacto de Banachievicz ou Doolittle. Ele representa a fatoração da matriz A no produto $L.U$ onde L é triangular inferior e U triangular superior, isto é:

$$A = \begin{array}{|cc|}\hline & U \\ L & \\ \hline \end{array}$$

Da mesma forma que fatoramos A em um produto $L.U$, , podemos escrever $A = L.U$ onde, agora fixamos os elementos $u_{ii} = 1$, fazendo com sejam calculados no processo. Essa fatoração recebe o nome de método de Crout .

1.5. Considerações sobre Métodos Diretos

Apresentamos os métodos diretos, e estes, após um certo número de operações fornecem a solução exata de um sistema, pelo menos teoricamente. Isto porque, quando fazemos muitas divisões e multiplicações, introduzimos erros de arredondamento produzidos pelo computador. Por isso, os métodos diretos só são indicados para o caso em que temos um sistema pequeno, de modo que, em sua solução tenhamos de fazer um número reduzido de operações; senão a solução que vamos obter muitas vezes se afasta da solução real.

Para os sistemas maiores, esse efeito diminui com outros métodos que garantem em certos casos, qualquer precisão desejada. Estes métodos são chamados métodos iterativos, onde usamos uma aproximação inicial da solução e a melhoramos quantas vezes sejam necessárias para chegarmos a uma precisão satisfatória.

2. RESOLUÇÃO DE SISTEMAS LINEARES POR MÉTODOS ITERATIVOS

Um método é iterativo quando fornece uma seqüência de aproximantes da solução, onde cada um dos quais é obtido dos anteriores pela repetição do mesmo tipo de processo. Um método iterativo é estacionário se cada aproximante é obtido do anterior sempre pelo mesmo processo. Quando os processos variam de passo para passo mas se repetem ciclicamente de *n* em *n* passos dizemos que o processo é *n*-cíclico. Agrupando-se os s passos de cada ciclo num único passo composto, obtemos um método estacionário.

No caso de métodos iterativos precisamos sempre saber se a seqüência que estamos obtendo está convergindo ou não para a solução desejada. Além disso, precisamos sempre ter em mente o significado de convergência, como será visto posteriormente.

Os métodos iterativos devem ser aplicados quando a matriz dos coeficientes é esparsa. Matrizes esparsas são aquelas que possuem muitos elementos iguais a zero. Podem ser usados também para reduzir os erros de arredondamento na solução obtida por métodos exatos e também podem ser aplicados para resolver equações diferenciais.

2.1. Introdução aos Métodos Iterativos

Vamos estudar métodos iterativos para a solução de sistemas de equações lineares do tipo:

$$Ax = b, \tag{1}$$

onde A é uma matriz quadrada sendo que $A \in \Re^{n\,x\,n}$ e b são vetores sendo que $b \in \Re^{n}$:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \quad\vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

Um método iterativo necessita de um número infinito de operações para fornecer a solução de um sistema. Assim, a partir de uma estimativa inicial $x^{(0)}$ são geradas sucessivos vetores $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ onde:

$$\lim_{k \to \infty} x^{(k)} = x \tag{2}$$

Como na realidade não é possível efetuar um número infinito de operações, os métodos iterativos são interrompidos, o que implica em termos uma solução aproximada do sistema, porém, perfeitamente boa para a finalidade pretendida.

As principal vantagem dos métodos iterativos esta no fatos destes produzirem boas aproximações com relativamente poucas iterações e a matriz . Outra vantagem que temos é que a matriz *A* nunca é alterada no decorrer do processo iterativo, permitindo economia de memória e tempo de cálculo. Esta característica permite que estes sejam particularmente adaptados para resolver sistemas de grandes dimensões com matrizes esparsas.

De forma geral, os métodos iterativos podem ser escritos como:

$$x^{(k+1)} = G_k x^{(k)} + c_k, \qquad k = 0,1, ..., \tag{3}$$

onde G_k é uma matriz conhecida (matriz de iteração) e c_k é um vetor.

A equação (3) também pode ser vista como:

$$x^{(k+1)} = x^{(k)} + G_k x^{(k)} - x^{(k)} + c_k$$

$$x^{(k+1)} = x^{(k)} + (G_k - I)x^{(k)} + c_k$$

$$x^{(k+1)} = x^{(k)} + h^{(k)}, \tag{4}$$

onde $h^{(k)} = (G_k - I)x^{(k)} + c_k$.

2.1.1. Definições

Definição: Um método iterativo é dito **linear** se $x^{(k+1)}$ depende de $x^{(j)}$, onde $0 \le j \le k$. Já um método iterativo é dito **não linear** se $x^{(k+1)}$ independe de $x^{(j)}$, onde $0 \le j \le k$.

Definição: Um método iterativo é dito **estacionário** se G_k e c_k dependem de *k*. Já um método iterativo é dito **não estacionário** se G_k e c_k independem de *k*.

A expressão geral dos **métodos estacionários** é dada por:

$$x^{(k+1)} = Gx^{(k)} + c, \qquad k = 0,1, ..., \qquad (5)$$

considerando assim que G e c dependem do contador de iterações *k*.

Já a expressão gral dos **métodos não-estacionários** é dada por:

$$x^{(k+1)} = G_k x^{(k)} + c_k \qquad k = 0,1, ..., \qquad (6)$$

considerando assim que G_k e c_k independem do contador de iterações *k*.

Definição: Um método iterativo é **convergente** se $\forall b$ e qualquer $x^{(0)}$, a sucessão $x^{(k)}$ convergir para um limite, independente de $x^{(0)}$.

Definição: Um método iterativo é **consistente** se $x^{(k+1)} = Gx^{(k)} + c_k$ e o sistema: $x = Gx + c \qquad (7)$

tenham a mesma solução.

A consistência obriga que os sistemas (1) e (7) tenham a mesma solução.

Como $Ax = b$ podemos escrever:

$$x = A^{-1}b$$

Temos também de $x = Gx + c$ que:

$$x - Gx = c$$
$$(I - G)x = c$$
$$c = (I - G)A^{-1}b$$

2.1.2. Decomposição da Matriz A

Vamos agora nos preocupar em obter a matriz *G*. Uma forma usual de construir esta matriz é partir de uma *decomposição aditiva* da matriz *A* em duas matrizes *M* e *N* tais que:

$$A = M - N \tag{8}$$

Como $Ax = b$, temos:

$$(M - N)x = b$$

$$Mx = b + Nx \tag{9}$$

Então, o método iterativo sugerido é:

$$Mx^{(k+1)} = b + Nx^{(k)} \tag{10}$$

Este sistema eqüivale a:

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b,$$

que comparado ao sistema (5), podemos fazer a seguinte identificação:

$$G = M^{-1}N \quad \text{e} \quad c = M^{-1}b \tag{11}$$

Fazendo algumas modificações em G, veremos que:

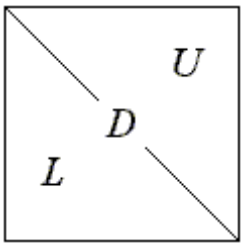
$$N = M - A$$
$$G = M^{-1}(M - A)$$
$$G = M^{-1}M - M^{-1}A \qquad ,I = M^{-1}M$$

e finalmente:

$$G = I - M^{-1}A \quad \text{e} \quad c = M^{-1}b \tag{12}$$

2.2. Método de Jacobi

Definimos as matrizes *L, D e U* onde *L* é formada pela parte triangular inferior, *D* é formada pela diagonal e *U* é formada pela parte triangular superior da matriz A, isto é:



Podemos então escrever:

$$A = D + L + U$$

O método de Jacobi consiste em escolher para matriz *M* a diagonal de *A*:

$$M = D = diag\ A$$

Assim, de (8) podemos verificar que:

$$N = -(L + U)$$

A matriz de iteração do método de Jacobi é dada por:

$$G_J = M^{-1}.N$$
$$G_J = D^{-1}.-(L+U)$$

$$G_J = -D^{-1}.(L+U) \qquad (13)$$

O processo iterativo do método de Jacobi é definido por:

$$x^{(k+1)} = -D^{-1}(L+U)x^{(k)} + D^{-1}b, \qquad (14)$$

Da expressão geral dos métodos estacionários (5) podemos escrever:

$$x_i^{(k+1)} = x_i^{(k)} + (b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k)}) \Big/ a_{ii} \qquad (15)$$

Com base em (15), teremos então o algoritmo do método de Jacobi:

```
Escolha do valor inicial x(0) para a solução x;
para k = 1,2,3 ...
     para i = 1,2,...,n
          σ = 0
          para j = 1,2,..., n
               se i ≠ j então  σ = σ + a_{i,j} x_j^{(k-1)}
          fim para
          σ = (b_i - σ) / a_{i,i}
     fim para
     x^(k) = σ
     calcular erro;
fim
```

Este algoritmo foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.3. Método de Gauss-Seidel

O método de Gauss-Seidel é uma modificação do método de Jacobi, que utiliza para o cálculo de uma componente de $x_i^{(k+1)}$ o valor mais recente de x_i.

Então, a matriz *M* é o triângulo inferior da matriz *A*, justamente para aproveitar os valores mais recentes e já calculados de x_i:

$$M = (L + D)$$

Assim, de (8) podemos verificar que:

$$N = -U$$

A matriz de iteração do método de Gauss-Seidel é dada por:

$$G_{GS} = I - M^{-1}.A$$

$$G_{GS} = I - (L + D)^{-1}.A \tag{16}$$

O processo iterativo do método de Gauss-Seidel definido por:

$$x^{(k+1)} = -(L + D)^{-1} Rx^{(k)} + (L + D)^{-1} b \tag{17}$$

Da expressão geral dos métodos estacionários (5) podemos escrever:

$$x_i^{(k+1)} = x_i^{(k)} + (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)}) \Big/ a_{ii} \tag{18}$$

Com base em (18), teremos então o algoritmo do método de Gauss-Seidel:

```
Escolha do valor inicial x(0) para a solução x;
para k = 1,2,3 ...
     para i = 1,2,...,n
          σ = 0
          para j = 1,2,..., n
               se j < i então  σ = σ + a_{i,j} x_j^(k)
               se j > i então  σ = σ + a_{i,j} x_j^(k-1)
          fim para
          x_i^(k) = (b_i - σ) / a_{i,i}
     fim para
     calcular erro;
fim
```

Este algoritmo foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.4. Métodos de Relaxação

Aplicando os métodos até aqui estudados, descobriu-se que era possível acelerar ou retardar a convergência de um método, somando aos valores de $x^{(k)}$ um certo fator de correção, ao qual chamamos de w. Modificando conveniente esta

correção, podemos então antecipar ou retardar a evolução das iterações, modificando com isto a convergência do método.

2.4.1. Método de Jacobi com Relaxação

Se afetarmos a correção aplicada a $x_i^{(k)}$ de um fator w, teremos:

$$M = \frac{1}{w} D \qquad \text{e} \qquad N = \frac{1-w}{w} D - L - U \tag{19}$$

Modificando então o parâmetro de relaxação w, teremos:

$$\rightarrow w < 1 : \textit{Sub} - \textit{relaxação}$$
$$\rightarrow w > 1 : \textit{Sobre} - \textit{relaxação}$$

A Matriz de iteração de Jacobi com relaxação é dada por:

$$G_{J(w)} = I - M^{-1}.A$$
$$G_{J(w)} = I - \left(\frac{1}{w}.D\right)^{-1}.A$$
$$G_{J(w)} = I - wD^{-1}.A$$

Como:

$$G_J = I - D^{-1}.A$$
$$D^{-1}.A = I - G_J \ ,$$

então podemos dizer que:

$$G_{J(w)} = I - w(I - G_J)$$
$$G_{J(w)} = I - wI + wG_J$$

$$G_{J(w)} = (1-w)I + wG_J \tag{20}$$

Aplicando a correção ao método de Jacobi, teremos então a expressão:

$$x_i^{(k+1)} = x_i^{(k)} + w(b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k)}) \Big/ a_{ii} \tag{21}$$

Mostraremos então o algoritmo do método de Jacobi com relaxação:

```
Escolha do valor inicial x(0) para a solução x;
Escolha o valor da relaxação w;
para k = 1,2,3 ...
     para i = 1,2,...,n
          σ = 0
          para j = 1,2,..., n
               se i ≠ j então   σ = σ + a_{i,j} x_j^{(k-1)}
          fim para
          σ = (b_i − σ) / a_{i,i}
     fim para
     x^{(k)} = x^{(k-1)} + w(σ − x^{(k-1)})
     calcular erro;
fim
```

Este algoritmo foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.4.2. Método de Gauss-Seidel com Relaxação

Aplicando a mesma idéia ao método de Gauss-Seidel, teremos:

$$M = \frac{1}{w} D + L \qquad \text{e} \qquad N = \frac{1-w}{w} D - U \tag{22}$$

Modificando então o parâmetro de relaxação w, teremos:

$$\rightarrow w < 1 : \textit{Sub} - \textit{relaxação}$$
$$\rightarrow w > 1 : \textit{Sobre} - \textit{relaxação}$$

A Matriz de iteração de Gauss-Seidel com relaxação é dada por:

$$G_{GS(w)} = M^{-1}.N$$

$$G_{GS(w)} = \left(\frac{1}{w}.D + L\right)^{-1}.\left(\frac{1-w}{w}.D - U\right)$$

$$G_{GS(w)} = \frac{\left(\frac{1-w}{w}.D - U\right)}{\left(\frac{1}{w}.D + L\right)} \quad ,que\ multiplicados\ por\ \frac{w}{D}:$$

$$G_{GS(w)} = \frac{(1-w)I - UwD^{-1}}{I + LwD^{-1}}$$

$$G_{GS(w)} = \left(I + wD^{-1}L\right)^{-1}.\left((1-w)I - wD^{-1}U\right) \qquad (23)$$

Aplicando a correção ao método de Gauss-Seidel, teremos então a expressão:

$$x_i^{(k+1)} = x_i^{(k)} + w(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)}) \Big/ a_{ii} \quad (24)$$

Mostraremos então o algoritmo do método de Gauss-Seidel com relaxação:

```
Escolha do valor inicial x(0) para a solução x;
Escolha o valor da relaxação w;
para k = 1,2,3 ...
     para i = 1,2,...,n
          σ = 0
          para j = 1,2,..., n
               se j < i então  σ = σ + a(i,j) x(j)^(k)
               se j > i então  σ = σ + a(i,j) x(j)^(k-1)
          fim para
          σ = (b(i) - σ) / a(i,i)
          x^(k) = x^(k-1) + w(σ - x^(k-1))
     fim para
     calcular erro;
fim
```

Este algoritmo foi implementado e testado em Pascal, e seu código será mostrado em anexo.

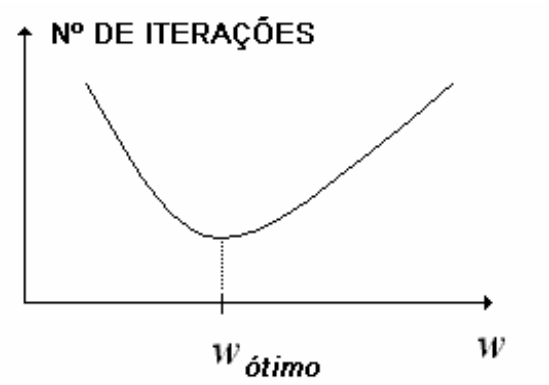
2.4.3.Determinação do Fator de Relaxação

A rapidez da convergência dos métodos iterativos depende agora do fator de relaxação w, de forma que o valor deste parâmetro torne $\|G\|$ mínima. Este seria então, um valor de w ótimo.

Contudo, este processo revela-se, na maioria dos casos, muito difícil ou até mesmo impossível. Embora existam estudos teóricos que permitem obter valores ótimos de relaxação, para certas classes de matrizes, somos obrigados a determinar estes valores de uma maneira um pouco mais experimental, para sermos assim mais abrangentes.

Uma maneira possível de se fazer isto é determinar o fator de relaxação ótimo por experimentação numérica. Neste processo, determina-se o número de iterações necessários à obtenção de uma dada precisão, para vários valores de w. Então, calcula-se aproximadamente o valor deste parâmetro w que produz o menor número de iterações. O custo da otimização efetuada pode assim, ser diminuído para a solução de muitos sistemas.

A figura a seguir esclarece melhor esta idéia:



2.5. Métodos Descendentes

Os métodos descendentes baseiam-se em acatar uma solução x de um sistema do tipo $Ax = b$, pela propriedade da minimização de uma norma apropriada do erro.

Por isso, são chamados de métodos de descendente, pois a partir de uma solução inicial x_0, estes procuram a solução correta pela diminuição do erro.

Seja a norma vetorial dada por:

$$\|x\|_S^2 = (x, Sx),$$

onde: $S \in \Re^{nxn}$ é uma matriz simétrica positivo definida, $x^{(k)}$ é uma solução aproximada e $e^{(k)}$ é o erro.

As normas vetoriais são utilizadas para se medir comprimentos de vetores. Estamos acostumados a utilizar esta noção ao fazermos uso do valor absoluto, estando num domínio unidimensional. No caso de métodos iterativos de sistemas lineares, a nossa preocupação consiste em obter uma estimativa da proximidade de um ponto situado num espaço vetorial multidimensional. Podemos estender a noção de proximidade através de normas vetoriais que medem comprimentos em espaços multidimensionais. É possível estender este conceito para matrizes.

A norma-1 definida por:

$$\|x\|^1 = \sum_{i=1}^{n} |x_i|$$

A norma-2 ou norma euclidiana (no corpo dos reais) definida por:

$$\|x\|^2 = \left[\sum_{i=1}^{n} (x_i)^2 \right]^{\frac{1}{2}}$$

Sabemos, evidentemente que $\|e^{(k)}\|_S^2 \geq 0$, e sendo que essa igualdade é verificada quando $e^{(k)} = 0$, ou seja, quando $x^{(k)} = x$.

Por outro lado, podemos ter que:

$$Ae^{(k)} = r^{(k)}, \qquad (25)$$

onde $r^{(k)}$ é chamado de resíduo, podemos deduzir que:

$$\left\|e^{(k)}\right\|_S^2 = (e^{(k)}, Se^{(k)}) = (r^{(k)}, Rr^{(k)})$$

em que pusemos:

$$R = A^{-T} S A^{-1} \tag{26}$$

Isto significa que fazendo a minimização do erro medido na norma $\|\bullet\|_S$, é equivalente a minimização do resíduo medido na norma $\|\bullet\|_R$.

Nos métodos descendentes parte-se de uma estimativa inicial $x^{(0)}$ da solução e gera-se uma sucessão de vetores $x^{(1)},...,x^{(k)},...$, caminhando sucessivamente ao longo da diminuição do erro $\left\|e^{(k)}\right\|_S^2$, de modo que essa sucessão convirja para *x*.

Para mostrar o que acabamos de dizer, suponhamos que tenha-se obtido um valor $x^{(k)}$ e que pretendamos avançar para um valor seguinte $x^{(k+1)}$, caminhando numa direção de pesquisa, na qual chamaremos de p_k, já conhecida. A equação paramétrica da reta que passa por $x^{(k)}$ e tem a direção de p_k é:

$$y = x^{(k)} + \alpha p^{(k)} \tag{27}$$

onde a iteração seguinte e o respectivo erro são dados por:

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} \tag{28}$$

$$e^{(k+1)} = e^{(k)} + \alpha_k p^{(k)} \tag{29}$$

Também é fácil concluir que o resíduo evolui ao longo das iterações de acordo com a equação:

$$\begin{aligned} r^{(k+1)} &= b - Ax^{(k+1)} \\ r^{(k+1)} &= b - A\left(x^{(k)} + \alpha_k p^{(k)}\right) \\ r^{(k+1)} &= \left(b - Ax^{(k)}\right) - \alpha_k Ap^{(k)} \end{aligned}$$

$$r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)} \tag{30}$$

Por conseguinte, as iterações são calculadas através de:

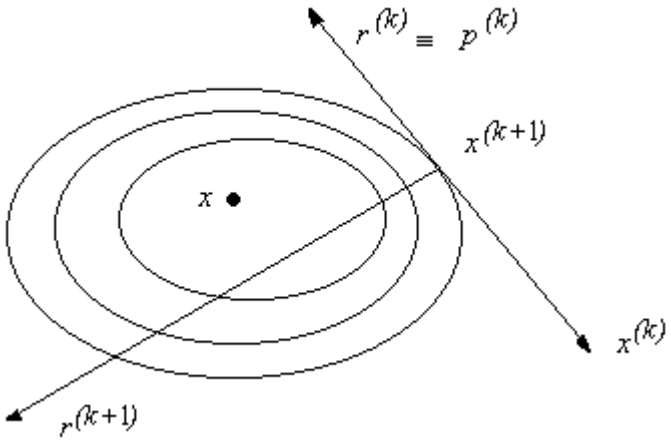
$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}, \qquad k = 0,1,... \quad (31)$$

em que o valor de α_k é dado por:

$$\alpha_k = \frac{\left(r^{(k)}, RAp^{(k)}\right)}{\left(p^{(k)}, \left(A^T RA\right)p^{(k)}\right)} \quad (32)$$

2.5.1. Método da Descendente mais Inclinada

Uma das formas de se achar a melhor direção de pesquisa seria escolher que em cada ponto $x^{(k)}$, a direção de $p^{(k)}$ que desce mais, aquela que esta alinhada com a direção do gradiente da função $\left\|e^{(k)}\right\|_S^2$, mas no seu sentido oposto. Por isso estes métodos são chamados da descencente mais inclinada, ou simplesmente, métodos de



gradiente.

Desta forma, a expressão iterativa dos métodos de gradiente é dada por:

$$x^{(k+1)} = G_k x^{(k)} + c_k \quad (33)$$

em que:

$$G_k = I - \alpha_k S$$

$$c_k = \alpha_k S A^{-1} b$$

verificando assim, que estes são métodos não-estacionários, isto é, G_k e c_k independem de *k*.

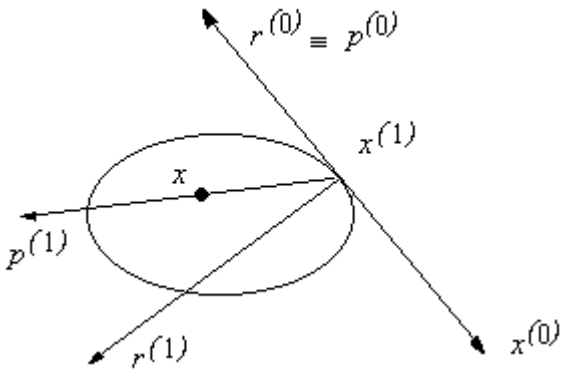
Apresentaremos agora o algoritmo do método de Gradientes:

Inicializa:
    Estimar $x^{(0)}$ *;*
    Fixar uma tolerância γ
    $r^{(0)} = b - Ax^{(0)}$
enquanto $\left\| r^{(k+1)} \right\| \geq r\|b\|_2$ *:*
    para *k = 0,1,2, ... faça*
        $\alpha_k = (r^{(k)}, r^{(k)}) / (r^{(k)}, Ar^{(k)})$
        $x^{(k+1)} = x^{(k)} + \alpha_k r^{(k)}$
        $r^{(k+1)} = r^{(k)} - \alpha_k r^{(k)}$
    fim para
fim

2.5.2. Método dos Gradientes Conjugados

O método dos gradientes conjugados é um método iterativo descendente, aplicado aos casos onde a matriz de iteração A é simétrica positivo definida. Este método prevê uma escolha mais criteriosa das direções de pesquisa $p^{(k)}$.

Para exemplificar, tomemos um problema bidimensional *n=(2)*. As linhas $\left\| e^{(k)} \right\|_A^2 = \left\| r^{(k)} \right\|_{A^{-1}}^2$ = constante, são elipses concêntricas com centro em *x*.



Suponhamos que dispomos de uma estimativa inicial $x^{(0)}$ e que de acordo com o método da descida mais inclinada, partiríamos deste ponto e pesquisaríamos o mínimo de $\left\|e^{(k)}\right\|_A^2$, ao longo da direção $r^{(0)}$, chegando a $x^{(1)}$. Observando a figura acima, verificamos que a melhor progressão de $x^{(1)}$ não seria a direção mais inclinada, mas sim uma direção $p^{(1)}$ que apontasse para o centro da elipse.

De fato, o novo ponto minimizador $x^{(2)}$ coincidiria com a solução exata de *x*, e assim o processo iterativo terminaria com apenas duas iterações. Podemos, então, desta forma, escolher a posição $\rho^{(1)}$ como:

$$p^{(1)} = x - x^{(1)} \tag{34}$$

Como se pode ver, atendendo a ortogonalidade dos resíduos sucessivos verificados anteriormente, temos:

$$\begin{aligned} 0 &= (r^{(1)}, r^{(0)}) \\ &= (r^{(1)}, p^{(0)}) \\ &= (b - Ax^{(1)}, p^{(0)}) \\ &= (A(x - x^{(1)}), p^{(0)}) \end{aligned}$$

de tal forma que:

$$(p^{(1)}, Ap^{(0)}) = 0 \tag{35}$$

As direções de busca são dadas por:

$$p^{(k+1)} = r^{(k+1)} + \beta_k p_k \tag{36}$$

onde β é dado por:

$$\beta_k = \frac{\left(r^{(k+1)}, r^{(k+1)}\right)}{\left(r^{(k)}, r^{(k)}\right)} \tag{37}$$

As iterações do método dos gradientes conjugados satisfazem a seguinte estimativa de erro:

$$\left\| e^{(k)} \right\|_A \leq \left(\frac{(cond_2 A)^{1/2} - 1}{(cond_2 A)^{1/2} + 1} \right) \left\| e^{(0)} \right\|_A \qquad (38)$$

Com isso, podemos ver que o método dos gradientes conjugados calcula a solução exata em *n* iterações. Sendo assim, por que então considerar este método como iterativo e não como direto? A resposta a esta questão está no fato que a utilização deste método em aritmética de precisão finita, introduz erros de arredondamento que destroem a exata conjugacidade das direções $p^{(k)}$ e a exata ortogonalidade dos resíduos $r^{(k)}$.

Nestas circunstâncias, não é certo que se obtenha a solução exata em um número finito de iterações. Neste caso, é recomendável efetuar-se as iterações necessárias que satisfaçam a critérios de convergência adequados. Desta forma, este método que é, em principio direto, adquire uma forma tipicamente iterativa.

Apresentaremos agora o algoritmo do método de Gradientes Conjugados:

Ler A, b
Estimar $x^{(0)}$*;*
Fixar uma tolerância γ
k = 0
$r^{(0)} = b - Ax^{(0)}$
$p^{(0)} = r^{(0)}$
enquanto $\left\| r^{(k+1)} \right\|_2 < \gamma \|b\|_2$ **faça:**
    k=k+1
    $\alpha_k = \left(p^{(k)}, r^{(k)} \right) / \left(p^{(k)}, Ap^{(k)} \right)$
    $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$
    $r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$
    $\beta_k = \left\| r^{(k+1)} \right\|^2 / \left\| r^{(k)} \right\|^2$
    $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$
fim enquanto

Este algoritmo foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.5.3. Método dos Gradientes Biconjugados

O método dos Gradientes Biconjugados é uma extensão do método dos gradientes conjugados, que é capaz de resolver problemas onde as matrizes *A* não tem que ser necessariamente simétricas positivo definidas. Embora a teoria desenvolvida para esta classe de métodos esteja ainda em fase de construção, podemos simplificadamente mostrar como o método funciona.

Como o próprio nome indica, este método vai possuir duas direções de pesquisa, sendo aqui denotadas por p_k e $\tilde{p}_k$, que podem ser calculadas respectivamente como:

$$p^{(k+1)} = r^{(k+1)} + \beta_k p_k \tag{39}$$

$$\tilde{p}^{(k+1)} = \tilde{r}^{(k+1)} + \beta_{k+1} \tilde{p}_k \tag{40}$$

Também teremos dois resíduos, que serão calculados como:

$$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)} \tag{41}$$

$$\tilde{r}^{(k+1)} = \tilde{r}^{(k)} - \alpha_k A^T \tilde{p}^{(k)} \tag{42}$$

sendo que os coeficientes α e β, podem ser obtidos:

$$\alpha_k = \frac{\left(\tilde{r}^{(k+1)}, r^{(k+1)}\right)}{\left(\tilde{p}^{(k)}, A p^{(k)}\right)} \tag{43}$$

$$\beta_k = \frac{\left(\tilde{r}^{(k+1)}, r^{(k+1)}\right)}{\left(\tilde{r}^{(k)}, r^{(k)}\right)} \tag{44}$$

Notemos que, se a matriz A é simétrica e positivo definida, então $\tilde{p}_k = p_k$ e $\tilde{r}_k = r_k$, e o método dos gradiente biconjugados se torna igual ao método dos gradientes conjugados.

Apresentaremos agora o algoritmo do método de Gradientes Biconjugados:

Ler A, b
Estimar $x^{(0)}$ *;*
Fixar uma tolerância γ
k = 0
$r^{(1)} = p^{(1)} = b$
$\tilde{r}^{(1)} = \tilde{p}^{(1)} = b$
$\rho_1 = \tilde{r}^{T(1)} r^{(1)}$
enquanto $\left\| r^{(k+1)} \right\|_2 < \gamma \|b\|_2$ **faça:**
k=k+1
$\sigma^{(k)} = \tilde{p}^{T(k)} A p^{(k)}$
$\alpha_k = \left(\rho^{(k)}\right) / \left(\sigma^{(k)}\right)$
$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$
$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$
$\tilde{r}^{(k+1)} = \tilde{r}^{(k)} - \alpha_k A^T \tilde{p}^{(k)}$
$\rho^{(k+1)} = \tilde{r}^{T(k+1)} r^{(k+1)}$
$\beta_k = \rho^{(k+1)} / \rho^{(k)}$
$p^{(k+1)} = r^{(k+1)} + \beta_{k+1} p^{(k)}$
$\tilde{p}^{(k+1)} = \tilde{r}^{(k+1)} + \beta_{k+1} \tilde{p}^{(k)}$
fim enquanto

Este algoritmo foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.6. Armazenamento de Matrizes

Os métodos de resolução de sistemas lineares vistos até o presente momento foram desenvolvidos num contexto de matrizes cheias, onde a questão da eventual esparsidade da matriz não foi explorada. Neste tópico vamos desenvolver uma metodologia que permita adaptar algoritmos escritos para matrizes cheias a matrizes esparsas, que são potencialmente as que encontramos nos problemas de interesse.

As matrizes podem ser catalogadas pela forma que seus coeficientes não-nulos se distribuem. Dividiremos as matrizes quanto à forma em três classes que cobrem todas as formas que foram mencionadas e que estarão relacionadas diretamente com o tipo de método utilizado na resolução do sistema linear associado. São elas:

a) matrizes tipo banda variável monótona: usadas no contexto de métodos diretos de resolução pois armazenam todos os coeficientes situados no interior da banda o que já prevê a transformação de coeficientes inicialmente nulos por valores não-nulos durante o processo de eliminação/fatoração. São utilizadas também no contexto de métodos iterativos, especialmente quando associados ao método do Gradiente Conjugado Pré-condicionado;

b) matrizes tipo morse: usadas no contexto de métodos iterativos que, como veremos, não alteram os coeficientes originais da matriz durante o processo de resolução. Armazenam apenas os coeficientes não-nulos mas devem também armazenar a sua localização individualmente;

c) matrizes tipo bloco: usadas no contexto tanto de métodos diretos quanto de iterativos e exploram a divisão da matriz em submatrizes;

Todas essas formas de armazenamento são diferentes maneiras de alocar os coeficientes da matriz de modo mais eficiente, explorando as suas características próprias de esparsidade. O resultado desta operação constitui-se para os algoritmos de resolução em duas vantagens: a de reduzir a memória necessária, eliminando os coeficientes nulos que forem inertes para o processo de resolução e por conseguinte, a de reduzir o número de operações necessárias para obter a solução do sistema. Para alcançar um gerenciamento da alocação dos coeficientes ativos da matriz sem entretanto

descaracterizar o algoritmo de resolução, a idéia fundamental consiste em estabelecer para cada forma de armazenamento um mapeamento do tipo:

$$A_{i,j} \rightarrow vA_{i,j}$$

onde $A_{i,j}$ representa a matriz A sob a forma de estrutura de matriz computacional e aloca por conseguinte todos os coeficientes ativos ou inertes sem distinção enquanto que $vA_{i,j}$ representa a matriz e sua esparsidade sob a forma de estrutura de vetor computacional alocando apenas os coeficientes ativos.

Nos temas subseqüentes iremos detalhar o mapeamento **matrizes tipo morse**, pelo fato de estarmos estudando no contexto de métodos iterativos, com a finalidade de armazenam apenas os coeficientes não-nulos, que é nossa finalidade principal, na resolução de equações diferenciais.

2.6.1. Armazenamento Morse

Existem situações, onde considerar que todos os coeficientes da matriz sejam armazenados leva a um desperdício de memória. Veremos que nos métodos iterativos, não ocorre o processo de preenchimento total da matriz e os seus coeficientes. Desta forma, apenas os valores não-nulos precisariam ser de fato armazenados. O armazenamento morse descrito nesta seção atinge exatamente este objetivo.

No método de armazenamento morse, considera-se que podem existir vazios (coeficientes nulos) e só seriam armazenados coeficientes não nulos. Em cada linha *i*, vão existir grupos de coeficientes não-nulos intercalados por vazios. Os limites de cada grupo *g* serão indicados por *min[g]* e *max[g]*. Um vetor auxiliar *loc[g]* acumula os coeficientes armazenados até o grupo *g*, fazendo o papel do vetor *loc[i]* do armazenamento para a linha *i*. Para poder localizar os grupos ativos em cada linha *i*, usaremos o vetor *grp* que acumula os grupos ativos em cada linha. O exemplo abaixo ajuda a compreender as estruturas utilizadas para o armazenamento morse.

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	
01	01	02	03		04		05	06	07			08	09			01
02		10	11	12	13	14		15		16		17	18			02
03																03
04	19		20	21	22											04
05			23	24	25			26	27	28	29					05
06				30	31	32			33	34						06
07				35	36											07
08							37	38	39			40	41	42	43	08
09			44	45	46	47										09
10	48	49	50							51	52	53	54			10
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	

	grp
01	04
02	08
03	08
04	10
05	12
06	14
07	15
08	17
09	18
10	20

	min	max	loc		min	max	loc
01	01	03	00	11	03	05	20
02	06	06	-2	12	09	12	17
03	09	11	-4	13	04	06	26
04	14	15	-6	14	10	11	23
05	02	06	08	15	04	05	31
06	09	09	06	16	07	09	30
07	11	11	05	17	12	15	28
08	13	14	04	18	03	06	41
09	01	01	18	19	01	03	47
10	03	05	16	20	10	13	41

A localização de um elemento *ij* da matriz *A* no vetor *va* pode ser obtido pela afirmação:

"Elementos acumulados até o grupo anterior ao atual (aquele que contém a coluna *j* na linha *i*) + Elementos acumulados no grupo atual até a coluna *j*"

e expresso pela relação:

$$ij = loc\left[Grupo(i,j)\right] + j$$

Para determinarmos Grupo(i,j), pode-se usar o algoritmo:

```
funcao grupo(i,j)
    para g de grp[i-1]+1 até grp[i]
        se j menor_igual max[g] então retorne g
    fim_para
fim_funcao
```

Obs:

- Define-se grp[0]=0;
- O vetor *loc* pode ser obtido de uma vez por todas, conhecidos os vetores *grp*, *min* e *max*;
- Exemplo: na matriz exemplificada acima A[5,10] = va[loc[12]+10] = va[17+10] = va[27].

2.7. Precondicionamento de Matrizes

Uma forma de tornar um método iterativo mais rápido consiste em transformar o sistema $A\,x = b$ num sistema equivalente, onde a matriz tenha um número de condição mais favorável.

Para ilustrar o que acabamos de dizer, podemos utilizar o método dos gradientes conjugados, que de acordo com a expressão (38), as iterações satisfazem a seguinte estimativa de erro:

$$\left\| e^{(k)} \right\|_A \leq \left(\frac{(cond_2 A)^{1/2} - 1}{(cond_2 A)^{1/2} + 1} \right) \left\| e^{(0)} \right\|_A$$

Logo, se pudermos diminuir $(cond_2 A)$ consequentemente o erro $\left\| e^{(0)} \right\|_A$ também irá diminuir, proporcionando uma rápida convergência.

Esta transformação é conseguida premultiplicando e posmultiplicando a matriz A por duas matrizes P e Q inversíveis do seguinte modo:

$$(PAQ)\,(Q)^{-1}\,x = P\,b \qquad (45)$$

O sistema transformado corresponde, portanto a :

$$\overline{A}\overline{x} = \overline{b}$$
$$com\ \ \overline{A} = PAQ, \qquad \overline{x} = Q^{-1}x, \qquad \overline{b} = Pb$$

As matrizes P e Q devem ser escolhidas de modo a proporcionar à matriz $\overline{A}$ melhores propriedades de convergência em relação a matriz original A.

Este procedimento é conhecido como precondicionamento de sistema de equações original, sendo P e Q conhecidas pelo nome de matrizes de precondicionamento. No caso onde $Q = I$, chamamos de precondicionamento esquerdo, já se $P = I$, chamamos de precondicionamento direito.

Por razões de economia computacional, é conveniente que o algoritmo de precondicionamento deve evitar a formação explícita de $\bar{A}$, e trabalhar apenas com a matriz original A.

Considerando que no método dos gradientes conjugados, a matriz transformada tem de continuar a ser simétrica e positivo definida, o precondicionamento também deverá preservar esta propriedade. De fato, podemos obter isso, fazendo com que:

$$P = S$$

$$Q = P^T = S^T$$

e desta forma, podemos chegar a:

$$\bar{A}\bar{x} = \bar{b}$$
$$com \ \ \bar{A} = SAS^T, \qquad \bar{x} = S^{-T}x, \qquad \bar{b} = Sb$$

Tendo em vista que $\left(cond_2 \bar{A}\right) << \left(cond_2 A\right)$, é fácil verificar que:

$$\bar{p}^{(0)} = \bar{r}^{(0)} = \bar{b} - \bar{A}\bar{x}^{(0)} = Sr^{(0)}$$

Por outro lado, se escolhermos

$$p^{(0)} = S^T \bar{p}^{(0)} = S^T \bar{r}^{(0)} = S^T S r^{(0)}$$

o que é sempre possível, temos que:

$$\left(\bar{p}^{(0)}, \bar{A}\bar{p}^{(0)}\right) = \left(p^{(0)}, Ap^{(0)}\right)$$

Também é válida a relação

$$\left(\bar{r}^{(0)}, \bar{r}^{(0)}\right) = \left(\bar{r}^{(0)}, r^{(0)}\right)$$

em que pusemos

$$W\bar{r}^{(0)} = r^{(0)} \qquad com \quad W = \left(S^T S\right)^{-1}$$

A partir daí, podemos obter sem dificuldade

$$\alpha_0 = \left(\bar{r}^{(0)}, r^{(0)}\right) / \left(p^{(0)}, Ap^{(0)}\right)$$

$$x^{(1)} = x^{(0)} + \alpha_0 p^{(0)}$$

$$r^{(1)} = r^{(0)} - \alpha_0 A p^{(0)}$$

$$\beta_0 = \left(\bar{r}^{(1)}, r^{(1)}\right) / \left(\bar{r}^{(0)}, r^{(0)}\right)$$

$$p^{(1)} = \bar{r}^{(1)} + \beta_0 p^{(0)}$$

mantendo estas expressões para as iterações seguintes.

Mostraremos então, o método dos gradientes conjugados precondicionado:

Ler A, b
Estimar $x^{(0)}$*;*
Fixar uma tolerância γ
k = 0
$r^{(0)} = b - Ax^{(0)}$
Resolver $W\tilde{r}^{(0)} = r^{(0)}$
$p^{(0)} = \tilde{r}^{(0)}$
enquanto $\left\|r^{(k+1)}\right\|_2 < \gamma \|b\|_2$ **faça:**
    k=k+1
    $\alpha_k = \left(\tilde{r}^{(k)}, r^{(k)}\right) / \left(p^{(k)}, Ap^{(k)}\right)$
    $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$
    $r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$
    Resolver $W\tilde{r}^{(k+1)} = r^{(k)}$
    $\beta_k = \left(\tilde{r}^{(k+1)}, r^{(k+1)}\right) \Big/ \left(\tilde{r}^{(k)}, r^{(k)}\right)$
    $p^{(k+1)} = \tilde{r}^{(k+1)} + \beta_k p^{(k)}$
fim enquanto

Nota-se que em cada iteração, é necessário resolvermos um sistema auxiliar:

$$W\tilde{r}^{(k+1)} = r^{(k)}$$

sendo por isso desejável que seja sistema 'fácil' de ser resolvido.

As situações extremas correspondem a:

- $W = I$: recuperamos o método dos gradientes conjugados
- $W = A$: obtemos a solução em uma única iteração, porém com o custo de ter que inverter a matriz A, o que não nos interessa.

O objetivo é ficar entre este dois extremos, sem recorrer a custos computacionais elevados.

Convém reparar que

$$A = SAS^T = S^{-T}(W^{-1}A)S^T \tag{46}$$

e, portanto

$$\overline{A} = W^{-1}A$$

Logo, podemos concluir que:

$$cond_2\overline{A} = cond_2\left(W^{-1}A\right) \tag{47}$$

Apresentaremos a seguir, algumas formas de escolhermos esta matriz W, de modo a tornar $cond_2\left(W^{-1}A\right)$ tão pequeno quanto possível, sem recorrer a custos computacionais demasiadamente elevados.

2.7.1. Precondicionamento pela Diagonal Principal de A

Pode-se dizer que este é o mais simples dos precondicionadores , onde a matriz de precondicionamento W escolhida, é a diagonal principal de A, ou como também chamada, matriz D.

Este tipo de precondicionamento é indicado para resolver sistemas onde a matriz a possui um número de condição baixo, porém não sendo ideal para ser calculado apenas pelo método dos gradientes conjugados. Nestes casos, podemos usar como precondicionador, a matriz diagonal principal de *A*. Assim, a inversa aproximada de *A*, denotada por W^{-1}, é calculada pela simples inversão da matriz diagonal *D*.

Para implementar este precondicionador, devemos resolver o sistema auxiliar $W\tilde{r}^{(k+1)} = r^{(k)}$ com a sub-rotina mostrada a seguir:

```
sub-rotina Resolver W r~(k+1) = r(k) ;
     se i=j então D = Aij ;
     W = D;
     r~(k+1) = W^-1 r(k) ;
fim da sub-rotina;
```

Este algoritmo de precondicionamento foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.7.2. Precondicionamento pelo Método S.S.O.R.

Vamos considerar agora o método S.S.O.R. (Simetric Sucessive Over Relaxation) que aplica os conceitos de sobre-relaxação para matrizes simétricas.

Para este caso, vamos considerar que a matriz *A* pode ser escrita como:

$$A = D + L + U$$

onde:

D = é a matriz diagonal

L = é a matriz triangular inferior

U = é a matriz triangular superior

Quando resolvemos sistemas onde a matriz *A* é simétrica, temos que *L = U*, logo, podemos escrever:

$$A = D + L + L^T$$

No precondicionamento por S.S.O.R., a inversa aproximada de A , denotada por W^{-1}, é:

$$W^{-1} = M^{-1} D M^T,$$

onde:

$$M = (D - w L) . (w (2 - w))^{(-1/2)}$$

em que w é o coeficiente de relaxação.

Para simplificar os cálculos de $W \tilde{r}^{(k+1)} = r^{(k)}$, vamos dividi-lo em três etapas:

(1) Resolve-se $(D - w L)\, x^* = w\, r^{(k)}$

(2) Calcula-se $x^{**} = D\, x^*$

(3) Resolve-se $(D - w L^T)\, \tilde{r}^{(k+1)} = w (2 - w)\, x^{**}$;para obter $\tilde{r}^{(k+1)}$.

A sub-rotina do método de precondicionamento S.S.O.R. será mostrada a seguir:

```
sub-rotina Resolver W r~(k+1) = r(k);
     estabelecer w;
     x* =(D - w L)^-1 .w.r(k); {subst. direta}
     x** = D.x*;
     r~(k+1) =(D-wL^T)^-1 .w.(2-w).x**; {subst. inversa}
fim da sub-rotina;
```

Este algoritmo de precondicionamento foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.7.3. Precondicionamento pelo Método da Decomposição LU Incompleta.

No precondicionamento pela decomposição *LU* incompleta, a inversa aproximada de A , denotada por W^{-1}, é uma decomposição *LU* de Choleski, não da matriz *A*, mas sim, de uma matriz ligeiramente diferente de *A*. Esta matriz é obtida preservando-se a esparsidade da matriz *A*, ou seja, não modificando seus elementos nulos.

Como estamos trabalhando com uma matriz *A* simétrica, a decomposição *LU* pode ser escrita como sendo uma decomposição LL^T .Para este caso, vamos considerar que a matriz *A* pode ser escrita como:

$$A = L\,L^T + E$$

em que *E* representa a diferença entre a matriz original *A*.

A matriz W é escolhida como sendo:

$$W = L\,L^T$$

A esparsidade deve ser respeitada, os coeficientes nulos da matriz *A* são mantidos e a matriz *W* é chamada de decomposição *LU* incompleta:

Desta forma, resolvemos $W\tilde{r}^{(k+1)} = r^{(k)}$ em duas etapas:

(1) $L\,z^{(k+1)} = r^{(k+1)}$;para obter $z^{(k+1)}$

(2) $L^T\,\tilde{r}^{(k+1)} = z^{(k+1)}$;para obter $\tilde{r}^{(k+1)}$

A sub-rotina do método de precondicionamento pela decomposição *LU* incompleta será mostrada a seguir:

```
sub-rotina Resolver W r̃^(k+1) = r^(k);
     fazer a decomposição LL^T incompleta;
      z^(k+1)=L^-1.r^(k+1); {subst. direta}
     r̃^(k+1) =L^-T.z^(k+1); {subst. inversa}
fim da sub-rotina;
```

Este algoritmo de precondicionamento foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.7.4. Precondicionamento pela minimização de I - AW na norma de Frobenius.

Um outro precondicionamento é obtido pela minimização de $I - AW$ pela norma de Frobenuis. O objetivo é encontrar uma matriz W em que o resíduo $I - AW$ calculado seja pequeno:

$$F(M) = || I - A\ W ||^2_F$$

Uma forma de minimizar esta norma de Frobenius é pelo algoritmo dos resíduos mínimos. Neste caso, escolhemos arbitrariamente uma matriz inicial W e calculamos o resíduo G da seguinte maneira:

$$G = I - A\ W$$

Em seguida, calculamos o coeficiente α, que auxilia a encontrar a nova matriz W:

$$\alpha = \frac{(G^T.A.G)}{\|A.W\|^2_F}$$

A nova matriz W é dada por:

$$W^{(k+1)} = W^{(k)} + \alpha.G$$

A sub-rotina do método de precondicionamento pela minimização de $I - AW$ pela norma de Frobenuis , utilizando o método dos resíduos mínimos é mostrada a seguir:

```
sub-rotina Resolver W r~(k+1) = r(k) ;
     escolha de W inicial;
     escolha de γ ;
     enquanto ||G||²F > γ faça
          C = A . W;
          G = I - C ;
```

$\alpha = (G^T.A.G) \big/ \|C\|_F^2$;

W = W + α . G;

fim enquanto;

$\tilde{r}^{(k+1)} = W^{-1} r^{(k)}$;

fim da sub-rotina;

Este algoritmo de precondicionamento foi implementado e testado em Pascal, e seu código será mostrado em anexo.

Outra segunda forma de minimizar esta norma de Frobenius é pelo algoritmo da descida mais íngreme. Também escolhemos uma matriz inicial *W* e calculamos o resíduo *G* da seguinte maneira:

$$G = A^T . (I - A W)$$

Em seguida, calculamos o coeficiente α, que auxilia a encontrar a nova matriz *W*:

$$\alpha = \frac{\|G\|_F^2}{\|A.G\|_F^2}$$

A nova matriz W é dada por:

$$W^{(k+1)} = W^{(k)} + \alpha . G$$

A sub-rotina do método de precondicionamento pela minimização de *I - AW* pela norma de Frobenuis , utilizando o método da descida mais íngreme é mostrada a seguir:

sub-rotina Resolver $W \tilde{r}^{(k+1)} = r^{(k)}$;

escolha de W inicial;

escolha de γ ;

enquanto $\|G\|_F^2 > \gamma$ **faça**

C = A . W;

R = I - C;

$G = A^T$. R ;

$\alpha = \|G\|_F^2 \big/ \|AG\|_F^2$;

$W = W + \alpha . G;$

fim enquanto;

$\tilde{r}^{(k+1)} = W^{-1} r^{(k)};$

fim da sub-rotina;

Este algoritmo de precondicionamento foi implementado e testado em Pascal, e seu código será mostrado em anexo.

2.8. Considerações sobre Métodos Iterativos

Quando comparamos métodos diretos com iterativos, não se pode garantir que método é o mais eficiente. É necessário o estabelecimento de certos critérios. Usando critérios bem gerais, pode-se afirmar que os métodos diretos se prestam aos sistemas de pequeno porte com matrizes de coeficientes densas; também resolvem satisfatoriamente sistemas lineares com a mesma matriz de coeficientes.

Já os métodos iterativos, quando há convergência garantida, são bastante vantajosos na resolução de sistemas de grande porte, com a matriz de coeficientes do tipo 'esparso' (grande número de zeros entre seus elementos). Os sistemas oriundos da discretização de equações diferenciais parciais são um caso típico. Neles, os zeros da matriz original são preservados e as iterações são conduzidas com a preservação da matriz original, tornando os cálculos autocorrigíveis, o que tende a minimizar os erros de arredondamento e torna-los muito mais rápidos e eficientes.

3. APLICAÇÃO PRÁTICA DE MÉTODOS ITERATIVOS

Os testes práticos dos métodos iterativos serão divididos em três partes.

A primeira parte consiste em um simples teste de funcionamento dos algoritmos implementados, onde foram usadas apenas matrizes muito pequenas, com a finalidade de verificar se o programa estava executando sem problemas.

Na segunda parte, utilizamos os métodos iterativos implementados para resolver equações diferenciais, pelo método das diferenças finitas de segunda ordem. Neste caso, a ordem das matrizes utilizadas eram de no máximo 60.

Finalmente, na terceira parte, utilizamos os métodos iterativos não estacionários para resolver grandes sistemas lineares esparsos. A matriz usada neste teste é "uma matriz representativa" de matrizes resultantes da aplicação do método dos elementos espectrais a equações diferenciais parciais (EDPs).

3.1. Teste de Funcionamento

Primeiramente, testamos os métodos iterativos estacionários: Jacobi, Jacobi com relaxação, Gauss-Seidel e Gauss-Seidel com relaxação. Para os métodos de relaxação, foram testados coeficientes compreendidos entre 0,2 e 1,2. A solução inicial $x^{(0)}$ encolhida para os métodos foi 0.

Estes métodos foram implementados na linguagem Pascal estruturada, utilizando o compilador Turbo Pascal 7.0. Para facilitar a execução dos algoritmos, estes foram juntos em quatro programas, distintos por sua ordem:

A:\TurboPascal\Estacionários\Iterativos_Estacionários_Ordem2.exe

A:\TurboPascal\Estacionários\Iterativos_Estacionários_Ordem3.exe

A:\TurboPascal\Estacionários\Iterativos_Estacionários_Ordem4.exe

A:\TurboPascal\Estacionários\Iterativos_Estacionários_Ordem5.exe

Neste mesmo diretório encontram seu código com a extensão .pas:

A:\TurboPascal\Estacionários\Iterativos_Estacionários.pas

Nos testes, foram utilizadas matrizes *A* e coeficientes *b* para resolução do sistema $Ax = b$, sendo que estes estão armazenados em forma de um arquivo de texto, que é solicitado pelo programa quando executado.

As matrizes *A* e vetores *b* utilizados foram:

A:\TurboPascal\Estacionários\m2a.txt

```
A =  2   1        b =   1
    -1   3             -4
```

A:\TurboPascal\Estacionários\m2b.txt

```
A =  5   4        b =  23
    -3   9             9
```

A:\TurboPascal\Estacionários\m3a.txt

```
A =  4   2   3    b =   8
     1   5   2          0
     3   2   6         13
```

A:\TurboPascal\Estacionários\m3b.txt

```
A =  9   3   2    b =  19
     5   8   1         17
     3   5  12         -1
```

A:\TurboPascal\Estacionários\m4a.txt

```
A =  2  -2   1  -1    b =   8
     2   3   1  -2          5
     1  -2   2  -1          9
    -3  -1   1   2         -4
```

A:\TurboPascal\Estacionários\m4b.txt

```
A =  4  -1   3   2     b =  11
    -2   2  -1   3         -12
     1  -3   3  -1          12
     2  -2   1   4           5
```

A:\TurboPascal\Estacionários\m5a.txt

```
A = 5  -1   3   2  -2     b =  7
   -2   3   1  -3   2          0
    1   3   4  -1   2          7
    2  -2   1   4   3         14
    1  -2   3  -3   5         27
```

A:\TurboPascal\Estacionários\m5b.txt

```
A = 5   1  -2   3   4     b =  5
    1   4  -3   3   2        -13
    1  -2   5   3  -4         -3
   -3   2   1   5   3         -6
    1   2   3   4   5         10
```

O número de iterações que cada método apresentou com os testes, foram organizados na tabela a seguir:

Metodo	Jacobi	Gauss_S	Jacobi_W					Gauss_S_W				
Sistema			0,2	0,4	0,6	0,8	1,2	0,2	0,4	0,6	0,8	1,2
m2a.txt	40	22	156	72	46	38	54	160	72	42	24	80
m2b.txt	56	30	162	80	54	48	84	158	70	40	24	144
m3a.txt	399	48	525	252	162	117	109	327	153	96	63	39
m3b.txt	120	36	381	177	111	75	2340	285	132	81	54	48
m4a.txt	1404	316	932	452	292	212	1054	700	332	224	164	207
m4b.txt	652	96	1040	588	468	556	2064	336	152	96	88	304
m5a.txt	1480	320	3635	1805	1195	890	1201	1740	855	555	410	330
m5b.txt	2110	536	5418	1205	1070	1230	4520	1005	720	1020	632	254

Agora, testamos os métodos iterativos não estacionários: Gradientes Conjugados, gradientes Conjugados Precondicionados (Diagonal Principal de A, SSOR, Decomposição LU incompleta, minimização de Frobenius pelos Resíduos mínimos e pela descida mais inclinada) e Gradientes Biconjugados. Para os método SSOR, utilizamos o coeficiente de relaxação sendo 1,3. Nos métodos de Frobenius, a matriz W inicial escolhida foi a diagonal principal de A modificada (foi utilizado 1 no lugar de coeficientes nulos). A solução inicial $x^{(0)}$ encolhida para os métodos foi 0.

Estes métodos foram implementados na linguagem Pascal estruturada, utilizando o compilador Turbo Pascal 7.0. Para facilitar a execução dos algoritmos, estes foram juntos em quatro programas, distintos por sua ordem:

A:\TurboPascal\NãoEstacionários\Iterativos_Nao_Estacionários_Ordem2.exe

A:\TurboPascal\NãoEstacionários\Iterativos_Nao_Estacionários_Ordem3.exe

A:\TurboPascal\NãoEstacionários\Iterativos_Nao_Estacionários_Ordem4.exe

A:\TurboPascal\NãoEstacionários\Iterativos_Nao_Estacionários_Ordem5.exe

Neste mesmo diretório encontram seu código com a extensão .pas:

A:\TurboPascal\NãoEstacionários\Iterativos_Nao_Estacionários.pas

Nos testes, foram utilizadas matrizes A e coeficientes b para resolução do sistema $Ax = b$, sendo que estes estão armazenados em forma de um arquivo de texto, que é solicitado pelo programa quando executado.

As matrizes A e vetores b utilizados foram:

A:\TurboPascal\Estacionários\m2c.txt

```
A =  2  -3        b =   5
    -3   1             -4
```

A:\TurboPascal\Estacionários\m2d.txt

```
A = -5   4        b = -23
     4   2              8
```

A:\TurboPascal\Estacionários\m3c.txt

```
A =  1   5  -2     b =   5
     5   3  -1           8
    -2  -1   4           8
```

A:\TurboPascal\Estacionários\m3d.txt

```
A = -2   1   3     b = -11
     1  -4   5          -4
     3   5  -3           7
```

A:\TurboPascal\Estacionários\m4c.txt

```
A =  1  -1   2   2     b =   2
    -1   3  -2   1         -10
     2  -2  -1   0           2
     2   1   0   2          -3
```

A:\TurboPascal\Estacionários\m4d.txt

```
A =  3  -2   0  -1     b = -16
    -2   4  -3   2          17
     0  -3   5   1           2
    -1   2   1   2          14
```

A:\TurboPascal\Estacionários\m5c.txt

```
A = 2   0  -1   3  -2     b = 15
    0   2   1  -1   0          -1
   -1   1   4  -2   3         -20
    3  -1  -2  -2  -1          -4
   -2   0   3  -1   5         -22
```

A:\TurboPascal\Estacionários\m5d.txt

```
A = 3  -1   2  -2  -3     b = -2
   -1   2   0  -3   1          12
    2   0   5   1   2          -5
   -2  -3   1   4   3         -10
   -3   1   2   3   2           5
```

O número de iterações que cada método apresentou com os testes, foram organizados na tabela a seguir:

Metodo	Gradiente	Gradiente Conjugado Precondicionado					Gradiente
Sistema	Conjug.	diag_A	SSOR	LU_incomp.	FRB_rm	FRB_di	Biconjug.
m2c.txt	2	2	2	2	2	2	3
m2d.txt	2	2	2	1	2	2	3
m3c.txt	3	3	3	14	overflow	3	4
m3d.txt	3	3	3	2	overflow	3	3
m4c.txt	4	4	3	erro	overflow	4	5
m4d.txt	4	4	4	2	overflow	4	5
m5c.txt	5	5	4	erro	overflow	5	6
m5d.txt	5	5	4	4	overflow	5	6

3.2. Aplicação dos Métodos Iterativos na Resolução de Equações Diferenciais

O objetivo destes testes é aplicar os métodos iterativos na resolução de equações diferenciais, pelo método das diferenças finitas de segunda ordem. Neste método, é gerado um sistema linear, no qual pode ser resolvido por métodos iterativos. Para isto, foram escolhidas duas equações diferenciais, e variamos o intervalo de pontos aos quais serão calculados seus coeficientes. Esse número de intervalos é será a ordem da matriz.

As equações utilizadas foram:

$$(1) \; -u'' - u' + 2u - 2 = 0$$
$$-1 < x < 1$$
$$u(-1) = u(1) = 0$$

(2) $- u'' + 2 \cos x - \text{sen } x\,(x+1) = 0$

$0 < x < 1$

$u(0) = 0 , u(1) = 2\text{sen}1$

Primeiramente, comparamos os métodos estacionários com os métodos de gradientes Conjugado e Biconjugado. Para o isto, utilizamos matrizes de ordem 10 até 60, com passos de 10.

Este programa foi implementado na linguagem Pascal estruturada, utilizando o compilador Turbo Pascal 7.0.

Os arquivos que contem os algoritmos que resolvem a equação diferencial (1) estão em:

A:\TurboPascal\EqDiferenciais\Eqdif_1_ordem10.exe

A:\TurboPascal\EqDiferenciais\Eqdif_1_ordem20.exe

A:\TurboPascal\EqDiferenciais\Eqdif_1_ordem30.exe

A:\TurboPascal\EqDiferenciais\Eqdif_1_ordem40.exe

A:\TurboPascal\EqDiferenciais\Eqdif_1_ordem50.exe

A:\TurboPascal\EqDiferenciais\Eqdif_1_ordem60.exe

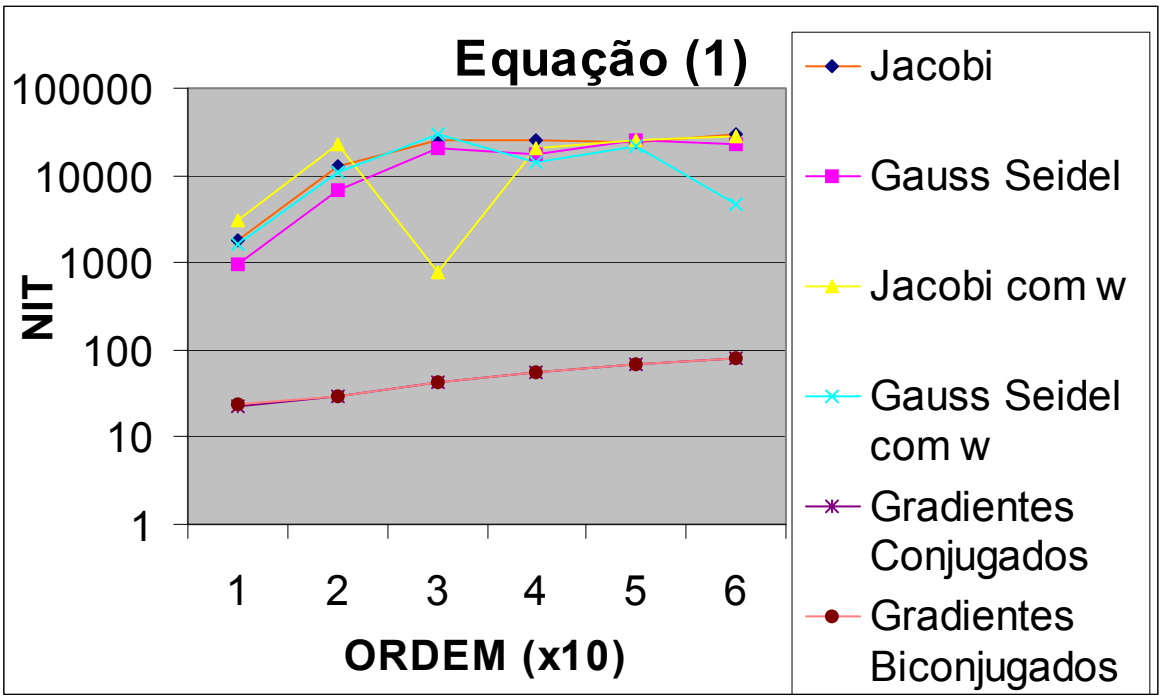
Neste mesmo diretório encontram seu código com a extensão .pas:

A:\TurboPascal\EqDiferenciais\Eqdif_1.pas

O número de iterações que cada método apresentou com os testes, foram organizados na tabela e gráfico a seguir:

MÉTODO / ORDEM	10	20	30	40	50	60
Jacobi	1810	12540	25786	25144	24258	29516
Gauss Seidel	970	6640	21030	17576	25644	23308
Jacobi com w = 0,6	3050	22960	794	20208	25256	27488

Gauss Seidel com w = 0,6	1650	11140	30376	14544	21078	4684
Gradientes Conjugados	23	29	42	55	67	80
Gradientes Biconjugados	24	30	43	56	68	81



Os arquivos que contem os algoritmos que resolvem a equação diferencial (2) estão em:

A:\TurboPascal\EqDiferenciais\Eqdif_2_ordem10.exe

A:\TurboPascal\EqDiferenciais\Eqdif_2_ordem20.exe

A:\TurboPascal\EqDiferenciais\Eqdif_2_ordem30.exe

A:\TurboPascal\EqDiferenciais\Eqdif_2_ordem40.exe

A:\TurboPascal\EqDiferenciais\Eqdif_2_ordem50.exe

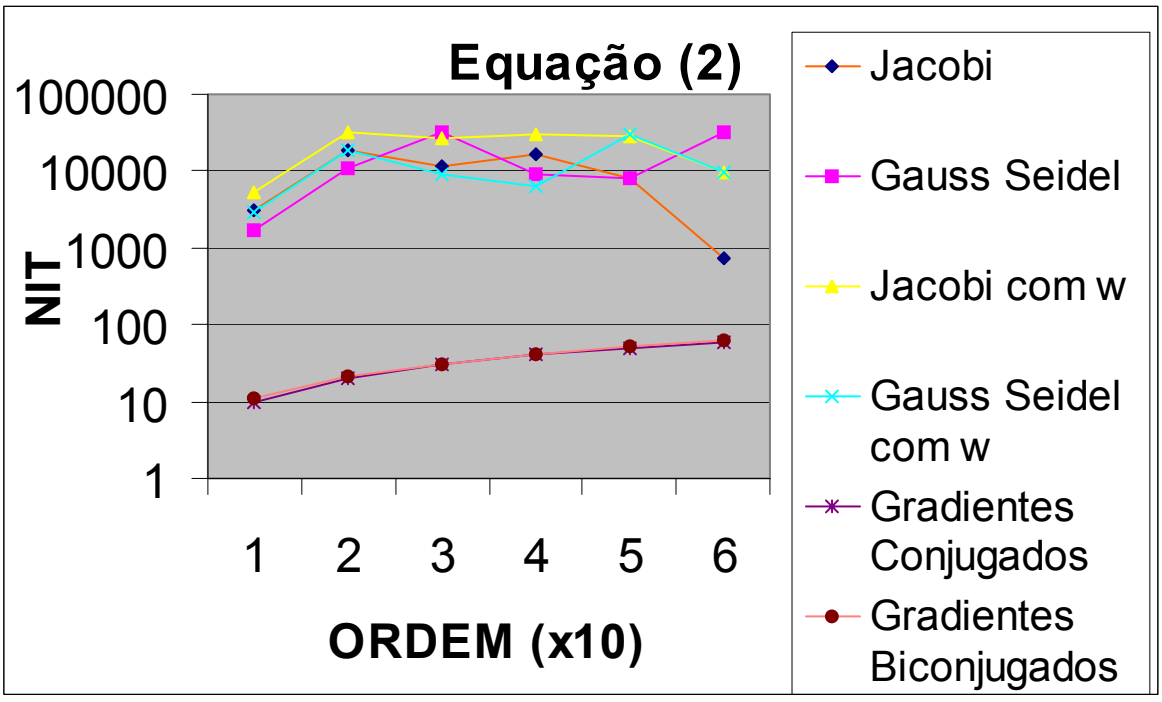
A:\TurboPascal\EqDiferenciais\Eqdif_2_ordem60.exe

Neste mesmo diretório encontram seu código com a extensão .pas:

A:\TurboPascal\EqDiferenciais\Eqdif_2.pas

O número de iterações que cada método apresentou com os testes, foram organizados na tabela e gráfico a seguir:

MÉTODO / ORDEM	10	20	30	40	50	60
Jacobi	3010	18580	11506	16392	7942	740
Gauss Seidel	1680	11060	31756	9344	7828	31984
Jacobi com w	5230	31116	26072	29864	28602	9416
Gauss Seidel com w	2840	18500	9136	6192	30494	9456
Gradientes Conjugados	10	20	30	40	50	60
Gradientes Biconjugados	11	21	31	41	51	61



Após estes testes iniciais, comparamos então os métodos não estacionários dos gradientes Conjugado com e sem precondicionadores e gradiente Biconjugado. Para o isto, utilizamos matrizes um pouco maiores, de ordem 100 até 300, com passos de 100.

Este programa foi implementado na linguagem Pascal estruturada, utilizando o compilador Delphi 3.0 , devido a limitações no tamanho da matriz encontradas no Turbo Pascal.

Os arquivos que contem os algoritmos que resolvem a equação diferencial (1) estão em:

A:\Delphi\Eqdif1\ProgEqdif_1_Ordem100.exe

A:\Delphi\Eqdif1\ProgEqdif_1_Ordem200.exe

A:\Delphi\Eqdif1\ProgEqdif_1_Ordem300.exe

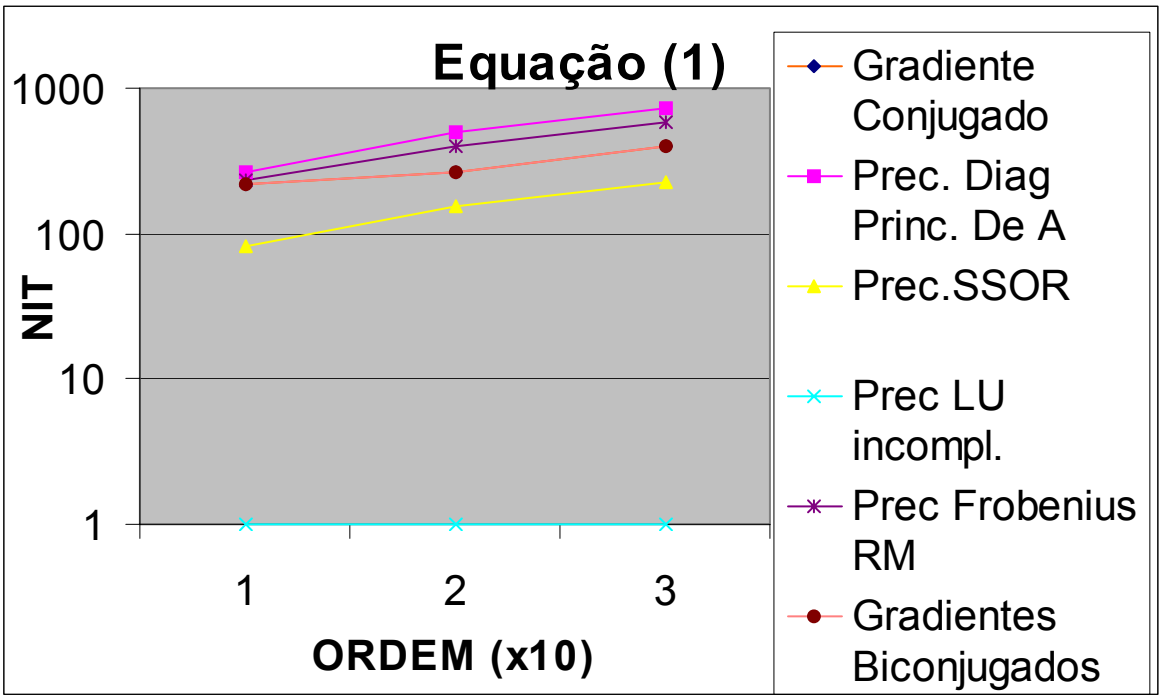
Neste mesmo diretório encontram seu código com a extensão .dpr e .pas:

A:\Delphi\Eqdif1\ProgEqdif_1.dpr

A:\Delphi\Eqdif1\UprEqdif_1.pas

O número de iterações que cada método apresentou com os testes, foram organizados na tabela e gráfico a seguir:

MÉTODO / ORDEM	100	200	300
Gradiente Conjugado	218	265	394
Prec. Diag Princ. De A	263	501	730
Prec.SSOR	83	152	222
Prec LU incompl.	1	1	1
Prec Frobenius RM	231	402	580
Prec Frobenius DI	201	336	425
Gradientes Biconjugados	219	266	395



Os arquivos que contem os algoritmos que resolvem a equação diferencial (2) estão em:

A:\Delphi\Eqdif1\ProgEqdif_2_Ordem100.exe

A:\Delphi\Eqdif1\ProgEqdif_2_Ordem200.exe

A:\Delphi\Eqdif1\ProgEqdif_2_Ordem300.exe

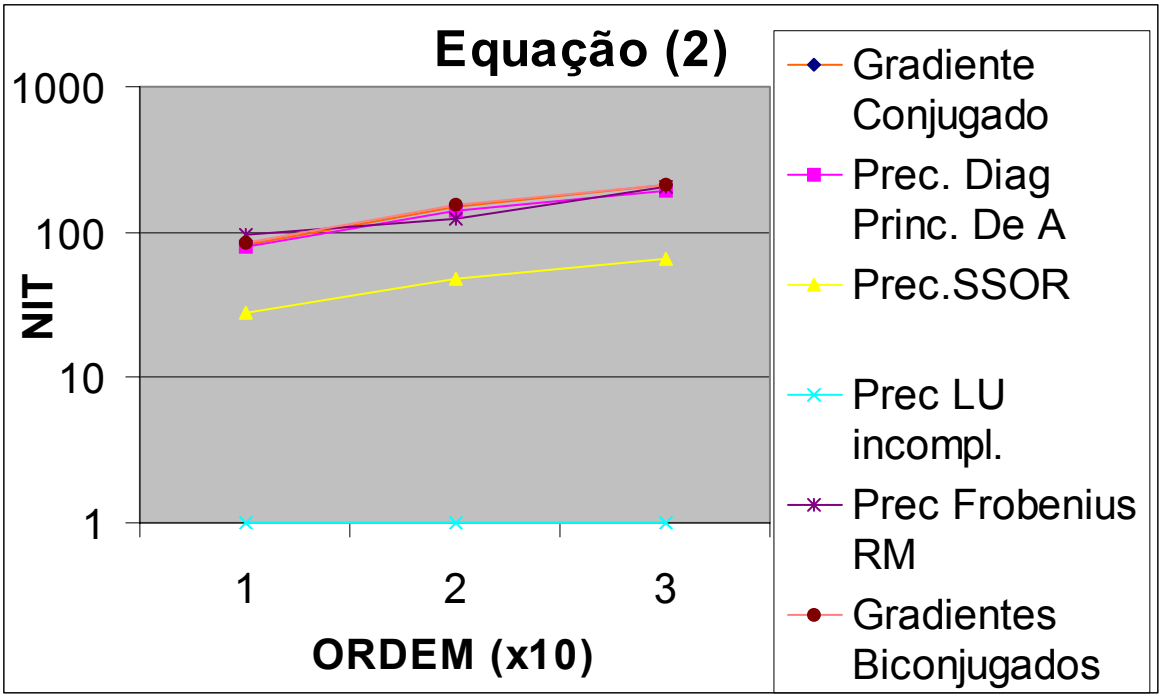
Neste mesmo diretório encontram seu código com a extensão .dpr e .pas:

A:\Delphi\Eqdif1\ProgEqdif_2.dpr

A:\Delphi\Eqdif1\UprEqdif_2.pas

O número de iterações que cada método apresentou com os testes, foram organizados na tabela e gráfico a seguir:

MÉTODO / ORDEM	100	200	300
Gradiente Conjugado	83	151	211
Prec. Diag Princ. De A	80	142	195
Prec.SSOR	28	48	66
Prec LU incompl.	1	1	1
Prec Frobenius RM	95	123	204
Prec Frobenius DI	78	103	189
Gradientes Biconjugados	84	152	212



3.3. Aplicação dos Métodos Iterativos na resolução de matrizes geradas por Métodos Espectrais

Vamos agora, resolver uma matriz grande, gerada pela aplicação de métodos espectrais. A matriz grande usada nos testes é "uma matriz representativa" de matrizes resultantes da aplicação do método dos elementos espectrais a equações diferenciais parciais (EDPs). Os métodos dos elementos espectrais é uma combinação

dos métodos espectrais com o método dos elementos finitos. Esta combinação via aproveitar a flexibilidade dos elementos finitos e uni-la a elevada acuracia dos métodos espectrais.

Para isto, é necessário fornecer ao programa esta matriz *A* externa, juntamente com os seus coeficientes *b*. Em nosso teste, utilizamos uma matriz de ordem 961x961, armazenada em um arquivo chamado mat_a.log e os vetores em outro arquivo chamado vet_b.log. No programa, temos uma rotina que abre automaticamente estes 2 arquivos e os lê. Por razão de seu tamanho ser muito grande, este arquivo será compactado e armazenado em:

A:\Delphi\IterativosNEstac\dados.zip

O programa foi implementado na linguagem Pascal estruturada, utilizando o compilador Delphi 3.0.

Os arquivos que contem os algoritmos que resolvem a matriz *A* resultante da aplicação de métodos espectrais, estão em:

A:\Delphi\IterativosNEstac\ProgIterativosNEst.exe

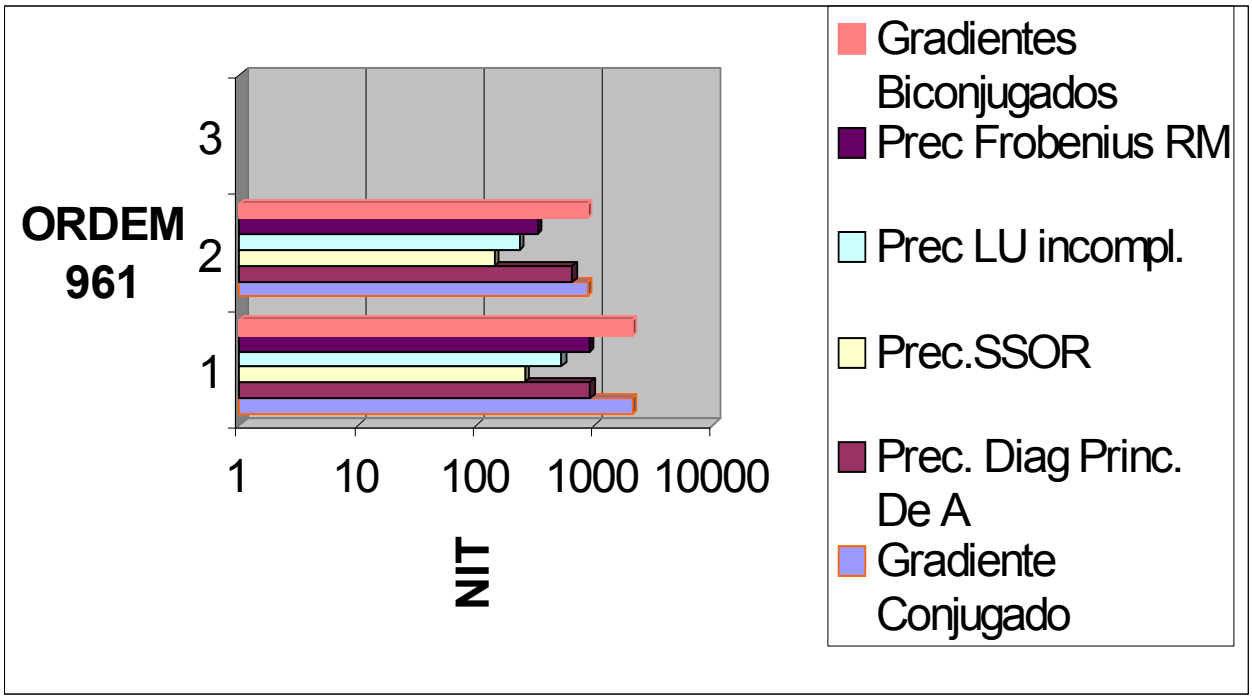
Neste mesmo diretório encontram seu código com a extensão .dpr e .pas:

A:\Delphi\IterativosNEstac\ProgIterativosNEst.dpr

A:\Delphi\IterativosNEstac\UprIterativosNEst.pas

O número de iterações que cada método apresentou com os testes, foram organizados na tabela e gráfico a seguir:

	vetor b = vet_b.log	vetor b = 1
MÉTODO / ORDEM	961	961
Gradiente Conjugado	2083	865
Prec. Diag Princ. De A	912	651
Prec.SSOR	256	142
Prec LU incompl.	526	231
Prec Frobenius RM	894	327
Prec Frobenius DI	761	287
Gradientes Biconjugados	2084	866



CONCLUSÕES

Neste trabalho, adquirimos uma grande experiência com o estudo dos métodos iterativos. Pudemos colocar em prática vários conhecimentos teóricos adquiridos, tanto na parte de análise e cálculo numérico, como também programação.

Com base nos testes realizados, vimos com clareza a superioridade dos métodos não estacionários na resolução de sistemas lineares grandes e esparsos, em relação aos métodos estacionários.

Quanto aos precondicionadores implementados, que eram o objetivo central de nosso trabalho, podemos fazer algumas considerações baseados nos testes realizados:

a) O método da diagonal principal de A apresentou eficiência para sistemas onde a matriz A apresentava um número de condição não muito elevado;

b) O método SSOR apresentou resultados muito bons, para todos os testes;

c) O método da decomposição LU incompleta comportou-se como um método direto nas equações diferenciais, resolvendo os problemas em uma única iteração. Teve também um bom desempenho na resolução do sistema de grande porte;

d) O método de minimização de Frobenuis por resíduos mínimos possui uma desvantagem: é mais lento que os outros e as vezes gera overflow, quando a matriz é cheia;

e) O método da minimização de Frobenius pela descida mais inclinada não apresenta o problema do overflow, porém, também é lento.

Pode-se dizer que todos os objetivos formam alcançados, pois pudemos verificar os pontos positivos e negativos de cada método iterativo.

ANEXOS

1) Implementação do Método de Jacobi

```
procedure metodo_Jacob;
var
 k,i,j    :integer;
begin
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if i<>j then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
  end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= resp[i];
    resp[i]:=0;
    nit:=nit+1;
   end;
 end;{while}
end;
```

2) Implementação do Método de Gauss-Seidel

```
procedure metodo_GaussSeidel;
var
 k,i,j    :integer;
begin
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if j<i then resp[i]:= (coef_a[i,j] * resp[j]) + resp[i];
     if j>i then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
   end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= resp[i];
    resp[i]:=0;
    nit:=nit+1;
```

```
   end;
 end;{while}
end;
```

3) Implementação do Método de Jacobi com Relaxação

```
procedure metodo_RJacob;
var
 k,i,j    :integer;
begin
 write('Digite o coeficiente de relaxacao w:');
 readln(w);
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if i<>j then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
  end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= x[i] + w*(resp[i] - x[i]) ;
    resp[i]:=0;
    nit:=nit+1;
   end;
 end;{while}
end;
```

4) Implementação do Método de Gauss-Seidel com Relaxação

```
procedure metodo_RGaussSeidel;
var
 k,i,j    :integer;
begin
 write('Digite o coeficiente de relaxacao w:');
 readln(w);
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if j<i then resp[i]:= (coef_a[i,j] * resp[j]) + resp[i];
     if j>i then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
   end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= x[i] + w*(resp[i] - x[i]) ;
```

```
    resp[i]:=0;
    nit:=nit+1;
   end;
 end;{while}
end;
```

5) Implementação dos Métodos Iterativos Estacionários

```
Program Iterativos_Estacionarios;

{Este programa resolve um problema  A x = b , onde:
 A eh uma matriz, x sao as icognitas e b eh um vetor
 de coeficientes, pelo metodos iterativos Estacionarios:
 Jacob, Gauss Seidel, com e sem relaxaçao}

uses
  crt;

const
  ordem = 3;

type
 vetor_real = array[1..ordem] of real;
 vetor_inteiro = array[1..ordem] of integer;
 matriz = array[1..ordem,1..ordem] of integer;

var
 i,j,k  :integer;
 coef_a :matriz;
 resp   :vetor_real;
 x      :vetor_real;
 coef_b :vetor_inteiro;
 erro   :real;
 nit    :integer;
 w      :real;
 Arquivo:TEXT;
 nome   :string;
 sair   :boolean;
 opsair :integer;
 contr  :boolean;


procedure le_coeficientes_arquivo;
begin
 write('Digite o nome do arquivo e extensao:');
 readln(nome);
 assign(Arquivo, nome);
 reset(Arquivo);
 while not EOF(Arquivo) do
 begin
  for i:=1 to ordem do
   begin
    for j:=1 to ordem do readln(Arquivo, coef_a[i,j]);
    readln(Arquivo, coef_b[i]);
   end;
 end;
 close(Arquivo);
end;

procedure le_coeficientes_manual;
```

```
begin
 for i:=1 to ordem do
 begin
  for j:=1 to ordem do
   begin
    write('Digite a',i,j ,':');
    readln(coef_a[i,j]);
   end;
  write('Digite o Coeficiente b',i,':');
  readln(coef_b[i]);
 end;
end;

procedure exibe_matriz;
begin
 write('MATRIZ DE COEFICIENTES A');
 for i:=1 to ordem do
  begin
   writeln('');
   for j:=1 to ordem do
    begin
     write('  ',coef_a[i,j]);
    end;
 end;
 writeln('');
end;

procedure exibe_resultado;
begin
 write('VETOR DE RESULTADOS');
 for i:=1 to ordem do
 begin
   writeln('');
   writeln('Variavel x',i, ': ',x[i]:10:2);
 end;
 writeln('Numero de Iteracoes necessarias: ',nit);
end;


procedure inicializa;
begin
 erro:=0;
 for i:= 1 to ordem do
  begin
   resp[i]:=0;
   x[i]:=0;
  end;
end;

procedure calcule_erro;
var v_erros: array[1..ordem] of real;
    cont:integer;
begin
 for cont:=1 to ordem do
  v_erros[cont]:= abs (x[cont] - resp[cont]);
 erro:= v_erros[1];
 for cont:=2 to ordem do
  begin
   if v_erros[cont] > v_erros[cont-1] then
    erro:=v_erros[cont];
  end;
```

```
end;


procedure metodo_Jacob;
var
 k,i,j    :integer;
begin
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if i<>j then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
  end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= resp[i];
    resp[i]:=0;
    nit:=nit+1;
   end;
 end;{while}
end;

procedure metodo_GaussSeidel;
var
 k,i,j    :integer;
begin
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if j<i then resp[i]:= (coef_a[i,j] * resp[j]) + resp[i];
     if j>i then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
   end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= resp[i];
    resp[i]:=0;
    nit:=nit+1;
   end;
 end;{while}
end;

procedure metodo_RJacob;
var
 k,i,j    :integer;
begin
 write('Digite o coeficiente de relaxacao w:');
```

```
 readln(w);
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if i<>j then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
  end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= x[i] + w*(resp[i] - x[i]) ;
    resp[i]:=0;
    nit:=nit+1;
   end;
 end;{while}
end;

procedure metodo_RGaussSeidel;
var
 k,i,j    :integer;
begin
 write('Digite o coeficiente de relaxacao w:');
 readln(w);
 nit:=0;
 erro:=5;
 while erro>0.0000001 do
 begin
  for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do
    begin
     if j<i then resp[i]:= (coef_a[i,j] * resp[j]) + resp[i];
     if j>i then resp[i]:= (coef_a[i,j] * x[j]) + resp[i];
    end;
    resp[i]:= (coef_b[i] - resp[i] ) / coef_a[i,i];
   end;
  calcule_erro;
  for i:=1 to ordem do
   begin
    x[i]:= x[i] + w*(resp[i] - x[i]) ;
    resp[i]:=0;
    nit:=nit+1;
   end;
 end;{while}
end;

procedure menu_inicial;
var
 opcao_a: integer;
 contr_a: boolean;
begin
 clrscr;

writeln('************************************************************');
```

```
 writeln('*
*');
 writeln('*            METODOS ITERATIVOS ESTACIONARIOS
*');
 writeln('*           PARA RESOLUCAO DE SISTEMAS LINEARES
*');
 writeln('*
*');
 writeln('* ORDEM DA MATRIZ   ',ordem,'
*');
 writeln('*
*');

writeln('************************************************************');
 writeln('');
 writeln('');
 writeln('');
 writeln('ENTRADA DE DADOS  :');
 writeln('');
 writeln('1 - DIGITACAO MANUAL');
 writeln('2 - ARMAZENADO EM ARQUIVO');
 writeln('');
 writeln('');
 contr_a:=FALSE;
 while contr_a=FALSE do
 begin
  write('ENTRADA DE DADOS -> ');
  readln(opcao_a);
  case opcao_a of
   1: begin
       le_coeficientes_manual;
       contr_a:=TRUE;
      end;
   2: begin
       le_coeficientes_arquivo;
       contr_a:=TRUE;
      end;
  Else
   contr_a:=FALSE;
  end;{case}
 end;{while}
 clrscr;
end;

procedure menu_metodos;
var
 opcao_b: integer;
 contr_b: boolean;
begin
 clrscr;

writeln('************************************************************');
 writeln('*
*');
 writeln('*            METODOS ITERATIVOS ESTACIONARIOS
*');
 writeln('*           PARA RESOLUCAO DE SISTEMAS LINEARES
*');
 writeln('*
*');
```

```
writeln('*******************************************************');
 writeln('');
 writeln('METODOS : ');
 writeln('');
 writeln('1 - METODO DE JACOBI');
 writeln('2 - METODO DE JACOBI COM RELAXACAO');
 writeln('3 - METODO DE GAUSS SEIDEL');
 writeln('4 - METODO DE GAUSS SEIDEL COM RELAXACAO');
 writeln('');
 writeln('');

writeln('*******************************************************');
 writeln('');
 contr_b:=FALSE;
 while contr_b=FALSE do
 begin
  write('METODO ESCOLHIDO ->');
  readln(opcao_b);
  case opcao_b of
   1: begin
       metodo_Jacob;
       contr_b:=TRUE;
      end;
   2: begin
       metodo_RJacob;
       contr_b:=TRUE;
      end;
   3: begin
        metodo_GaussSeidel;
        contr_b:=TRUE;
      end;
   4: begin
        metodo_RGaussSeidel;
        contr_b:=TRUE;
      end;
  Else
      contr_b:=FALSE;
  end;{case}
 end;{while}
clrscr;
end;

BEGIN
 menu_inicial;
 sair:=FALSE;
 repeat
  inicializa;
  menu_metodos;
  exibe_resultado;
  contr:=false;
  while contr=FALSE do
  begin
  writeln('');
  writeln('');
  writeln('');
  writeln('');

writeln('*******************************************************');
  writeln('1 - Executar novamente');
  writeln('2 - Sair');
```

```
  write('OPCAO ->');
  readln(opsair);
   case opsair of
    1: begin
         sair:= TRUE;
         contr:=TRUE;
       end;
    2: begin
         sair:= FALSE;
         contr:=TRUE;
       end;
    Else
     contr:=FALSE;
   end;{case}
  end;{while}
 until sair=FALSE;



END.
```

6) Implementação do Método de Gradientes Conjugados

```
procedure metodo_GradConjPre;
var
 i,j    :integer;
 Ax     :vetor_real;
 o_aux  :vetor_real;
 o      :real;
 rtilTr :real;
begin
 nit:=0;
 for i:= 1 to ordem do Ax[i]:=0;
 for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do Ax[i]:=(coef_a[i,j] * x[j]) + Ax[i];
   r[i]:=coef_b[i] - Ax[i];
  end;

 case prec of
   0:  calcule_rtil_0(coef_a,r);
   1:  calcule_rtil_1(coef_a,r);
   2:  calcule_rtil_2(coef_a,r);
   3:  calcule_rtil_3(coef_a,r);
   4:  calcule_rtil_4(coef_a,r);
   5:  calcule_rtil_5(coef_a,r);
 end;

 for i:=1 to ordem do p[i]:=rtil[i];
 pp[0]:=0;
 for i:=1 to ordem do pp[0]:= (rtil[i]*r[i]) + pp[0];
 while (abs(pp[nit])> tol )  do if (nit<nit_max) then
 begin
  nit:=nit+1;
  o:=0;
  for i:=1 to ordem do o_aux[i]:=0;
  for i:=1 to ordem do
  begin
   for j:=1 to ordem do
    o_aux[i]:= (p[j] * coef_a[j,i]) + o_aux[i];
```

```
   o:= (o_aux[i] * p[i]) + o;
  end;
  rtilTr:=0;
  for i:=1 to ordem do
   rtilTr:= (rtil[i]*r[i]) + rtilTr;
  aa:=rtilTr/o;
  for i:=1 to ordem do ww[i]:=0;
  for i:=1 to ordem do
   for j:=1 to ordem do ww[i]:= (coef_a[i,j] * p[j]) + ww[i];
  for i:=1 to ordem do
   begin
    x[i]:=x[i] + (aa * p[i]);
    r[i]:=r[i] - (aa * ww[i]);
   end;

  case prec of
   0:  calcule_rtil_0(coef_a,r);
   1:  calcule_rtil_1(coef_a,r);
   2:  calcule_rtil_2(coef_a,r);
   3:  calcule_rtil_3(coef_a,r);
   4:  calcule_rtil_4(coef_a,r);
   5:  calcule_rtil_5(coef_a,r);
  end;

  pp[nit]:=0;
  for i:=1 to ordem do pp[nit]:= (rtil[i]*r[i]) + pp[nit];
  bb:=pp[nit] / pp[nit-1];
  for i:=1 to ordem do p[i]:= rtil[i] + (bb * p[i]);
 end;{while}
end;
```

7) Implementação do Método de Gradientes Biconjugados

```
procedure metodo_GradBiconj;
var
 i,j       :integer;
 Ap        :vetor_real;
 o_aux     :vetor_real;
 rrT       :real;
begin
 nit:=0;
 for i:=1 to ordem do
  begin
   r[i]:= coef_b[i];
   p[i]:= coef_b[i];
   rtil[i]:= coef_b[i];
   ptil[i]:= coef_b[i];
  end;
 pp[1]:=0;
 for i:=1 to ordem do pp[1]:= (r[i]*r[i]) + pp[1];
 pp[0]:=pp[1];
 while (pp[nit]> tol ) AND (nit<nit_max) do
  begin
   nit:=nit+1;
   o[nit]:=0;
   for i:=1 to ordem do o_aux[i]:=0;
   for i:=1 to ordem do
   begin
    for j:=1 to ordem do
     o_aux[i]:= (p[j] * coef_a[j,i]) + o_aux[i];
```

```
    o[nit]:= (o_aux[i] * p[i]) + o[nit];
   end; {end for}
   aa:=pp[nit]/o[nit];
   for i:= 1 to ordem do Ap[i]:=0;
   for i:=1 to ordem do
    begin
     for j:=1 to ordem do
      Ap[i]:=(coef_a[i,j] * p[j]) + Ap[i];
     r[i]:= r[i] - aa * Ap[i];
    end;
   for i:=1 to ordem do
    x[i]:=  x[i] + (aa * p[i]);
   for i:=1 to ordem do
    begin
     Ap[i]:=0;
     for j:=1 to ordem do
      Ap[i]:=(coef_a[j,i] * ptil[j]) + Ap[i];
     rtil[i]:= rtil[i] - aa * Ap[i];
    end;
   for i:=1 to ordem do rrT:=0;
   for i:=1 to ordem do rrT:= (r[i] * r[i]) + rrT;
   pp[nit+1]:= rrT;
   bb:=pp[nit+1] / pp[nit];
   for i:= 1 to ordem do
    begin
     p[i]:= r[i] + (bb * p[i]);
     ptil[i]:= rtil[i] + (bb * ptil[i]);
    end;
   end;{while}
end;
```

8) Implementação dos Precondicionadores

```
{SEM PRECONDICIONAMENTO}
procedure calcule_rtil_0(coef_a_linha:matriz;
coef_b_linha:vetor_real);
begin
 for i:=1 to ordem do rtil[i]:=0;
 for i:=1 to ordem do rtil[i]:=r[i];
end;

{PRECONDICIONAMENTO PELA DIAGONAL PRINCIPAL DA MATRIZ A}
procedure calcule_rtil_1(coef_a_linha:matriz;
coef_b_linha:vetor_real);
begin
 for i:=1 to ordem do rtil[i]:=0;
 for i:=1 to ordem do
  rtil[i]:=coef_b_linha[i]/coef_a_linha[i,i];
end;

{PRECONDICIONAMENTO PELO METODO SSOR}
procedure calcule_rtil_2(coef_a_linha:
matriz;coef_b_linha:vetor_real);
var  x_aster : vetor_real;
begin
 for i:=1 to ordem do rtil[i]:=0;
 {calcula - w L}
 for i:=1 to ordem  do
   for j:=1 to ordem do
    coef_a_linha[i,j]:= -w * coef_a_linha[i,j];
```

```
 {calcula w r_(k+1)}
 for k:=1 to ordem do coef_b_linha[k]:= w * coef_b_linha[k];
 {Resolve-se (D - w L) x* = w r_(k+1)}
  for k:= 1 to ordem-1 do
   begin
    x_aster[k]:=coef_b_linha[k] / coef_a_linha[k,k];
    for i:= k+1 to ordem do
     coef_b_linha[i]:= coef_b_linha[i] - x_aster[k] *
coef_a_linha[i,k];
   end;
  x_aster[ordem]:=coef_b_linha[ordem] / coef_a_linha[ordem,ordem];
  {Calcula-se x** = D x*}
  for i:=1 to ordem do x_aster[i]:= coef_a[i,i] * x_aster[i];
  {Calcula-se w (2 - w) x** }
  for k:=1 to ordem do coef_b_linha[k]:= w * (2-w) * x_aster[k];
  {Resolve-se (D - w L') rtil = w (2 - w) x** }
  for k:= ordem downto 2 do
   begin
    rtil[k]:=coef_b_linha[k] / coef_a_linha[k,k];
    for i:= k-1  downto 1 do
     coef_b_linha[i]:= coef_b_linha[i] - rtil[k] * coef_a_linha[i,k];
   end;
  rtil[1]:=coef_b_linha[1] / coef_a_linha[1,1];
end;

{PRECONDICIONAMENTO PELO METODO DA DECOMPOSICAO LU INCOMPLETA}
procedure calcule_rtil_3(coef_a_linha:matriz;
coef_b_linha:vetor_real);
var
 l: vetor_inteiro;
 smax, r , rmax : real;
 lk,h : integer;
 s : vetor_real;
 xmult,soma: real;
begin
for i:=1 to ordem do rtil[i]:=0;
for i:= 1 to ordem do
 begin
  j:=1;
  l[i]:= i ;
  smax:=coef_a_linha[i,j];
  for j:= 2 to ordem do
   if smax<coef_a_linha[i,j] then smax:=coef_a_linha[i,j];
  s[i]:= smax;
 end;
for k:=1 to ordem-1 do
begin
 rmax:= 0;
 for i:= k to ordem do
  begin
   r:= abs(coef_a_linha[l[i], k])/s[l[i]];
   if r > rmax then
    begin
     j:= i;
     rmax:= r;
    end;
  end;
 lk:= l[j];
 l[j]:= l[k];
 l[k]:= lk;
 for i:= k + 1 to ordem do
```

```
  begin
   xmult:= coef_a_linha[l[i], j]/coef_a_linha[lk,k];
   for h:= k+1 to  ordem do
    if coef_a_linha[l[i],h] <> 0 then  {preservar os coeficientes
nulos}
     coef_a_linha[l[i], h]:= coef_a_linha[l[i], h] - xmult *
coef_a_linha[lk, h];
   coef_a_linha[l[i], k]:= xmult;
  end;
end;
for k:= 1 to ordem-1 do
 for i:= k+1 to ordem do
   coef_b_linha[l[i]]:= coef_b_linha[l[i]] - coef_b_linha[l[k]] *
coef_a_linha[l[i],k];
rtil[ordem]:= coef_b_linha[l[ordem]]/coef_a_linha[l[ordem],ordem];
for i:= ordem-1 downto 1 do
 begin
  soma:= coef_b_linha[l[i]];
  for j:= i+1 to ordem do  soma:= soma - rtil[j] *
coef_a_linha[l[i],j];
  rtil[i]:= soma/coef_a_linha[l[i],i];
 end;
end;

{PRECONDICIONAMENTO PELO MINIMIZACAO DE FROBENIUS USANDO RESIDUOS
MINIMOS}
procedure calcule_rtil_4(coef_a_linha:matriz;
coef_b_linha:vetor_real);
var
 W,C,G,GT,GTA,GTAG : matriz;
 stp: integer;
 coef_alfa: real;
begin
for i:=1 to ordem do rtil[i]:=0;
stp:= 5;
 {escolha de W inicial}
 for i:=1 to ordem do
  for j:=1 to ordem do
   if i=j then W[i,j]:=coef_a_linha[i,j]
          else W[i,j]:=1;
 {repeticao ate convergencia}
 for l:=1 to stp do
  begin
   {C = A W}
   for i:=1 to ordem do
    for j:=1 to ordem do C[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      C[i,j]:= (coef_a_linha[i,k] * W[k,j]) + C[i,j];
   {G = I - C}
   for i:=1 to ordem do
    for j:=1 to ordem do
     if i=j then G[i,j]:= 1 - C[i,j]
            else G[i,j]:= - C[i,j];
   {GTAG}
   for i:=1 to ordem do
    for j:=1 to ordem do GTAG[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do GT[i,j]:=G[j,i];
   for i:=1 to ordem do
```

```
    for j:=1 to ordem do
     for k:=1 to ordem do
      GTA[i,j]:= (GT[i,k] * coef_a_linha[k,j]) + GTA[i,j];
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      GTAG[i,j]:= (GTA[i,k] * G[k,j]) + GTAG[i,j];
   {alfa}
   coef_alfa:= calcule_traco_matriz(GTAG) / (calcule_norma_matriz(C) *
calcule_norma_matriz(C));
   {W novo}
   for i:=1 to ordem do
    for j:=1 to ordem do
     W[i,j]:= coef_alfa * W[i,j];
  end;
  for i:=1 to ordem do rtil[i]:=0;
  for i:=1 to ordem do
   for j:=1 to ordem do
    rtil[i]:= (W[i,j] * coef_b_linha[j]) + rtil[i];
end;

{PRECONDICIONAMENTO PELO MINIMIZACAO DE FROBENIUS USANDO DESCIDA MAIS
INGREME}
procedure calcule_rtil_5(coef_a_linha:matriz;
coef_b_linha:vetor_real);
var
 W,C,R,G,AT,AG : matriz;
 stp: integer;
 coef_alfa: real;
begin
for i:=1 to ordem do rtil[i]:=0;
stp:= 5;
 {escolha de W inicial}
 for i:=1 to ordem do
  for j:=1 to ordem do
   if i=j then W[i,j]:=coef_a_linha[i,j]
          else W[i,j]:=1;
 {repeticao ate convergencia}
 for l:=1 to stp do
  begin
   {C = A W}
   for i:=1 to ordem do
    for j:=1 to ordem do C[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      C[i,j]:= (coef_a_linha[i,k] * W[k,j]) + C[i,j];
   {R = I - C}
   for i:=1 to ordem do
    for j:=1 to ordem do
     if i=j then R[i,j]:= 1 - C[i,j]
            else R[i,j]:= - C[i,j];
   {G}
   for i:=1 to ordem do
    for j:=1 to ordem do G[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do AT[i,j]:=coef_a_linha[j,i];
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      G[i,j]:= (AT[i,k] * R[k,j]) + G[i,j];
```

```
   {A * G}
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      AG[i,j]:= (coef_a_linha[i,k] * G[k,j]) + G[i,j];
   {alfa}
   coef_alfa:= (calcule_norma_matriz(G) * calcule_norma_matriz(G)) /
(calcule_norma_matriz(AG) * calcule_norma_matriz(AG));
   {W novo}
   for i:=1 to ordem do
    for j:=1 to ordem do
     W[i,j]:= coef_alfa * W[i,j];
  end;
  for i:=1 to ordem do rtil[i]:=0;
  for i:=1 to ordem do
   for j:=1 to ordem do
    rtil[i]:= (W[i,j] * coef_b_linha[j]) + rtil[i];
end;
```

9) Implementação dos Métodos Iterativos Estacionários

```
Program Iterativos_NaoEstacionarios;

{Este programa resolve um problema  A x = b , onde:
 A eh uma matriz, x sao as icognitas e b eh um vetor
 de coeficientes, pelo metodos iterativos Nao Estacionarios:
 Gradientes Conjugados, Gradientes Conjugados Precondicionados e
 Gradientes Biconjugados}

uses
  crt;

const
  ordem = 3;
  nit_max = 1000;
  tol = 0.0000001;
  w = 1.3;

type
 vetor_real = array[1..ordem] of real;
 vetor_inteiro = array[1..ordem] of integer;
 matriz = array[1..ordem,1..ordem] of real;

var
 i,j,k,l  :integer;
 coef_a :matriz;
 resp   :vetor_real;
 x      :vetor_real;
 coef_b :vetor_inteiro;
 nit    :integer;
 ww      :vetor_real;
 r,rtil :vetor_real;
 p,ptil :vetor_real;
 aa,bb  :real;
 o   :array[0..nit_max] of real;
 pp     :array[0..nit_max] of real;
 Arquivo:TEXT;
 nome   :string;
 sair   :boolean;
```

```
 opsair :integer;
 contr  :boolean;
 prec   :integer;

procedure le_coeficientes_arquivo;
begin
 write('Digite o nome do arquivo e extensao:');
 readln(nome);
 assign(Arquivo, nome);
 reset(Arquivo);
 while not EOF(Arquivo) do
 begin
  for i:=1 to ordem do
   begin
    for j:=1 to ordem do readln(Arquivo, coef_a[i,j]);
    readln(Arquivo, coef_b[i]);
   end;
 end;
 close(Arquivo);
end;

procedure le_coeficientes_manual;
begin
 for i:=1 to ordem do
 begin
  for j:=1 to ordem do
   begin
    write('Digite a',i,j ,':');
    readln(coef_a[i,j]);
   end;
  write('Digite o Coeficiente b',i,':');
  readln(coef_b[i]);
 end;
end;

procedure exibe_matriz;
begin
 write('MATRIZ DE COEFICIENTES A');
 for i:=1 to ordem do
  begin
   writeln('');
   for j:=1 to ordem do
    begin
     write('  ',coef_a[i,j]);
    end;
 end;
 writeln('');
end;

procedure exibe_resultado;
begin
 write('VETOR DE RESULTADOS');
 for i:=1 to ordem do
 begin
   writeln('');
   writeln('Variavel x',i, ': ',x[i]:10:2);
 end;
 writeln('Numero de Iteracoes necessarias: ',nit);
end;
```

```
procedure inicializa;
begin
 for i:= 1 to ordem do
  begin
   resp[i]:=0;
   x[i]:=0;
  end;
end;


function calcule_norma_vetor(var vetor:array of real): real;
var calc: real;
begin
 calc:=0;
 for i:=0 to ordem-1 do
  begin
   calc:= (vetor[i]*vetor[i]) + calc;
  end;
 calcule_norma_vetor:=sqrt(calc);
end;

function calcule_norma_matriz (var mtz: matriz): real;
var calc:real;
begin
 calc:=0;
 for i:=1 to ordem do
  for j:=1 to ordem do
   begin
    calc:= (mtz[i,j]* mtz[i,j]) + calc;
   end;
  calcule_norma_matriz:=sqrt(calc);
end;

function calcule_traco_matriz (var mtz: matriz): real;
var calc:real;
begin
 calc:=0;
 for i:=1 to ordem do
  calc:= mtz[i,i] + calc;
 calcule_traco_matriz:=calc;
end;

procedure escolha_precondicionador;
var
 opcao_a: integer;
 contr_a: boolean;
begin
 writeln('');
 writeln('');
 writeln('PRECONDICIONADORES DISPONIVEIS  :');
 writeln('');
 writeln('1 - DIAGONAL PRINCIPAL DA MATRIZ A');
 writeln('2 - METODO SSOR');
 writeln('3 - METODO DA DECOMPOSICAO LU INCOMPLETA');
 writeln('4 - MINIMIZACAO DE FROBENIUS USANDO RESIDUOS MINIMOS');
 writeln('5 - MINIMIZACAO DE FROBENIUS USANDO DESCIDA MAIS INGREME');
 writeln('');
 writeln('');
 contr_a:=FALSE;
 while contr_a=FALSE do
 begin
```

```
  write('ENTRADA DE DADOS -> ');
  readln(opcao_a);
  case opcao_a of
   1: begin
       prec:=1;
       contr_a:=TRUE;
      end;
   2: begin
       prec:=2;
       contr_a:=TRUE;
      end;
   3: begin
       prec:=3;
       contr_a:=TRUE;
      end;
   4: begin
       prec:=4;
       contr_a:=TRUE;
      end;
   5: begin
       prec:=5;
       contr_a:=TRUE;
      end;
  Else
   contr_a:=FALSE;
  end;{case}
 end;{while}
end;

{SEM PRECONDICIONAMENTO}
procedure calcule_rtil_0(coef_a_linha:matriz;
coef_b_linha:vetor_real);
begin
 for i:=1 to ordem do rtil[i]:=0;
 for i:=1 to ordem do rtil[i]:=r[i];
end;

{PRECONDICIONAMENTO PELA DIAGONAL PRINCIPAL DA MATRIZ A}
procedure calcule_rtil_1(coef_a_linha:matriz;
coef_b_linha:vetor_real);
begin
 for i:=1 to ordem do rtil[i]:=0;
 for i:=1 to ordem do
  rtil[i]:=coef_b_linha[i]/coef_a_linha[i,i];
end;

{PRECONDICIONAMENTO PELO METODO SSOR}
procedure calcule_rtil_2(coef_a_linha:
matriz;coef_b_linha:vetor_real);
var  x_aster : vetor_real;
begin
 for i:=1 to ordem do rtil[i]:=0;
 {calcula - w L}
 for i:=1 to ordem  do
   for j:=1 to ordem do
    coef_a_linha[i,j]:= -w * coef_a_linha[i,j];
 {calcula w r_(k+1)}
 for k:=1 to ordem do coef_b_linha[k]:= w * coef_b_linha[k];
 {Resolve-se (D - w L) x* = w r_(k+1)}
  for k:= 1 to ordem-1 do
   begin
```

```
    x_aster[k]:=coef_b_linha[k] / coef_a_linha[k,k];
    for i:= k+1 to ordem do
     coef_b_linha[i]:= coef_b_linha[i] - x_aster[k] *
coef_a_linha[i,k];
   end;
  x_aster[ordem]:=coef_b_linha[ordem] / coef_a_linha[ordem,ordem];
  {Calcula-se x** = D x*}
  for i:=1 to ordem do x_aster[i]:= coef_a[i,i] * x_aster[i];
  {Calcula-se w (2 - w) x** }
  for k:=1 to ordem do coef_b_linha[k]:= w * (2-w) * x_aster[k];
  {Resolve-se (D - w L') rtil = w (2 - w) x** }
  for k:= ordem downto 2 do
   begin
    rtil[k]:=coef_b_linha[k] / coef_a_linha[k,k];
    for i:= k-1  downto 1 do
     coef_b_linha[i]:= coef_b_linha[i] - rtil[k] * coef_a_linha[i,k];
   end;
  rtil[1]:=coef_b_linha[1] / coef_a_linha[1,1];
end;

{PRECONDICIONAMENTO PELO METODO DA DECOMPOSICAO LU INCOMPLETA}
procedure calcule_rtil_3(coef_a_linha:matriz;
coef_b_linha:vetor_real);
var
 l: vetor_inteiro;
 smax, r , rmax : real;
 lk,h : integer;
 s : vetor_real;
 xmult,soma: real;
begin
for i:=1 to ordem do rtil[i]:=0;
for i:= 1 to ordem do
 begin
  j:=1;
  l[i]:= i ;
  smax:=coef_a_linha[i,j];
  for j:= 2 to ordem do
   if smax<coef_a_linha[i,j] then smax:=coef_a_linha[i,j];
  s[i]:= smax;
 end;
for k:=1 to ordem-1 do
begin
 rmax:= 0;
 for i:= k to ordem do
  begin
   r:= abs(coef_a_linha[l[i], k])/s[l[i]];
   if r > rmax then
    begin
     j:= i;
     rmax:= r;
    end;
  end;
 lk:= l[j];
 l[j]:= l[k];
 l[k]:= lk;
 for i:= k + 1 to ordem do
  begin
   xmult:= coef_a_linha[l[i], j]/coef_a_linha[lk,k];
   for h:= k+1 to  ordem do
    if coef_a_linha[l[i],h] <> 0 then  {preservar os coeficientes
nulos}
```

```
     coef_a_linha[l[i], h]:= coef_a_linha[l[i], h] - xmult *
coef_a_linha[lk, h];
   coef_a_linha[l[i], k]:= xmult;
  end;
end;
for k:= 1 to ordem-1 do
 for i:= k+1 to ordem do
   coef_b_linha[l[i]]:= coef_b_linha[l[i]] - coef_b_linha[l[k]] *
coef_a_linha[l[i],k];
rtil[ordem]:= coef_b_linha[l[ordem]]/coef_a_linha[l[ordem],ordem];
for i:= ordem-1 downto 1 do
 begin
  soma:= coef_b_linha[l[i]];
  for j:= i+1 to ordem do  soma:= soma - rtil[j] *
coef_a_linha[l[i],j];
  rtil[i]:= soma/coef_a_linha[l[i],i];
 end;
end;

{PRECONDICIONAMENTO PELO MINIMIZACAO DE FROBENIUS USANDO RESIDUOS
MINIMOS}
procedure calcule_rtil_4(coef_a_linha:matriz;
coef_b_linha:vetor_real);
var
 W,C,G,GT,GTA,GTAG : matriz;
 stp: integer;
 coef_alfa: real;
begin
for i:=1 to ordem do rtil[i]:=0;
stp:= 5;
 {escolha de W inicial}
 for i:=1 to ordem do
  for j:=1 to ordem do
   if i=j then W[i,j]:=coef_a_linha[i,j]
          else W[i,j]:=1;
 {repeticao ate convergencia}
 for l:=1 to stp do
  begin
   {C = A W}
   for i:=1 to ordem do
    for j:=1 to ordem do C[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      C[i,j]:= (coef_a_linha[i,k] * W[k,j]) + C[i,j];
   {G = I - C}
   for i:=1 to ordem do
    for j:=1 to ordem do
     if i=j then G[i,j]:= 1 - C[i,j]
            else G[i,j]:= - C[i,j];
   {GTAG}
   for i:=1 to ordem do
    for j:=1 to ordem do GTAG[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do GT[i,j]:=G[j,i];
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      GTA[i,j]:= (GT[i,k] * coef_a_linha[k,j]) + GTA[i,j];
   for i:=1 to ordem do
    for j:=1 to ordem do
```

```
     for k:=1 to ordem do
      GTAG[i,j]:= (GTA[i,k] * G[k,j]) + GTAG[i,j];
   {alfa}
   coef_alfa:= calcule_traco_matriz(GTAG) / (calcule_norma_matriz(C) *
calcule_norma_matriz(C));
   {W novo}
   for i:=1 to ordem do
    for j:=1 to ordem do
     W[i,j]:= coef_alfa * W[i,j];
  end;
  for i:=1 to ordem do rtil[i]:=0;
  for i:=1 to ordem do
   for j:=1 to ordem do
    rtil[i]:= (W[i,j] * coef_b_linha[j]) + rtil[i];
end;

{PRECONDICIONAMENTO PELO MINIMIZACAO DE FROBENIUS USANDO DESCIDA MAIS
INGREME}
procedure calcule_rtil_5(coef_a_linha:matriz;
coef_b_linha:vetor_real);
var
 W,C,R,G,AT,AG : matriz;
 stp: integer;
 coef_alfa: real;
begin
for i:=1 to ordem do rtil[i]:=0;
stp:= 5;
 {escolha de W inicial}
 for i:=1 to ordem do
  for j:=1 to ordem do
   if i=j then W[i,j]:=coef_a_linha[i,j]
          else W[i,j]:=1;
 {repeticao ate convergencia}
 for l:=1 to stp do
  begin
   {C = A W}
   for i:=1 to ordem do
    for j:=1 to ordem do C[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      C[i,j]:= (coef_a_linha[i,k] * W[k,j]) + C[i,j];
   {R = I - C}
   for i:=1 to ordem do
    for j:=1 to ordem do
     if i=j then R[i,j]:= 1 - C[i,j]
            else R[i,j]:= - C[i,j];
   {G}
   for i:=1 to ordem do
    for j:=1 to ordem do G[i,j]:=0;
   for i:=1 to ordem do
    for j:=1 to ordem do AT[i,j]:=coef_a_linha[j,i];
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      G[i,j]:= (AT[i,k] * R[k,j]) + G[i,j];
   {A * G}
   for i:=1 to ordem do
    for j:=1 to ordem do
     for k:=1 to ordem do
      AG[i,j]:= (coef_a_linha[i,k] * G[k,j]) + G[i,j];
```

```
   {alfa}
   coef_alfa:= (calcule_norma_matriz(G) * calcule_norma_matriz(G)) /
(calcule_norma_matriz(AG) * calcule_norma_matriz(AG));
   {W novo}
   for i:=1 to ordem do
    for j:=1 to ordem do
     W[i,j]:= coef_alfa * W[i,j];
  end;
  for i:=1 to ordem do rtil[i]:=0;
  for i:=1 to ordem do
   for j:=1 to ordem do
    rtil[i]:= (W[i,j] * coef_b_linha[j]) + rtil[i];
end;



procedure metodo_GradConjPre;
var
 i,j    :integer;
 Ax     :vetor_real;
 o_aux  :vetor_real;
 o      :real;
 rtilTr :real;
begin
 nit:=0;
 for i:= 1 to ordem do Ax[i]:=0;
 for i:= 1 to ordem do
  begin
   for j:= 1 to ordem do Ax[i]:=(coef_a[i,j] * x[j]) + Ax[i];
   r[i]:=coef_b[i] - Ax[i];
  end;

 case prec of
   0:  calcule_rtil_0(coef_a,r);
   1:  calcule_rtil_1(coef_a,r);
   2:  calcule_rtil_2(coef_a,r);
   3:  calcule_rtil_3(coef_a,r);
   4:  calcule_rtil_4(coef_a,r);
   5:  calcule_rtil_5(coef_a,r);
 end;

 for i:=1 to ordem do p[i]:=rtil[i];
 pp[0]:=0;
 for i:=1 to ordem do pp[0]:= (rtil[i]*r[i]) + pp[0];
 while (abs(pp[nit])> tol )  do if (nit<nit_max) then
 begin
  nit:=nit+1;
  o:=0;
  for i:=1 to ordem do o_aux[i]:=0;
  for i:=1 to ordem do
  begin
   for j:=1 to ordem do
    o_aux[i]:= (p[j] * coef_a[j,i]) + o_aux[i];
   o:= (o_aux[i] * p[i]) + o;
  end;
  rtilTr:=0;
  for i:=1 to ordem do
   rtilTr:= (rtil[i]*r[i]) + rtilTr;
  aa:=rtilTr/o;
  for i:=1 to ordem do ww[i]:=0;
  for i:=1 to ordem do
```

```
   for j:=1 to ordem do ww[i]:= (coef_a[i,j] * p[j]) + ww[i];
  for i:=1 to ordem do
   begin
    x[i]:=x[i] + (aa * p[i]);
    r[i]:=r[i] - (aa * ww[i]);
   end;

  case prec of
   0:  calcule_rtil_0(coef_a,r);
   1:  calcule_rtil_1(coef_a,r);
   2:  calcule_rtil_2(coef_a,r);
   3:  calcule_rtil_3(coef_a,r);
   4:  calcule_rtil_4(coef_a,r);
   5:  calcule_rtil_5(coef_a,r);
  end;

  pp[nit]:=0;
  for i:=1 to ordem do pp[nit]:= (rtil[i]*r[i]) + pp[nit];
  bb:=pp[nit] / pp[nit-1];
  for i:=1 to ordem do p[i]:= rtil[i] + (bb * p[i]);
 end;{while}
end;

procedure metodo_GradBiconj;
var
 i,j       :integer;
 Ap        :vetor_real;
 o_aux     :vetor_real;
 rrT       :real;
begin
 nit:=0;
 for i:=1 to ordem do
  begin
   r[i]:= coef_b[i];
   p[i]:= coef_b[i];
   rtil[i]:= coef_b[i];
   ptil[i]:= coef_b[i];
  end;
 pp[1]:=0;
 for i:=1 to ordem do pp[1]:= (r[i]*r[i]) + pp[1];
 pp[0]:=pp[1];
 while (pp[nit]> tol ) AND (nit<nit_max) do
  begin
   nit:=nit+1;
   o[nit]:=0;
   for i:=1 to ordem do o_aux[i]:=0;
   for i:=1 to ordem do
   begin
    for j:=1 to ordem do
     o_aux[i]:= (p[j] * coef_a[j,i]) + o_aux[i];
    o[nit]:= (o_aux[i] * p[i]) + o[nit];
   end; {end for}
   aa:=pp[nit]/o[nit];
   for i:= 1 to ordem do Ap[i]:=0;
   for i:=1 to ordem do
    begin
     for j:=1 to ordem do
      Ap[i]:=(coef_a[i,j] * p[j]) + Ap[i];
     r[i]:= r[i] - aa * Ap[i];
    end;
   for i:=1 to ordem do
```

```
    x[i]:=  x[i] + (aa * p[i]);
   for i:=1 to ordem do
    begin
     Ap[i]:=0;
     for j:=1 to ordem do
      Ap[i]:=(coef_a[j,i] * ptil[j]) + Ap[i];
     rtil[i]:= rtil[i] - aa * Ap[i];
    end;
   for i:=1 to ordem do rrT:=0;
   for i:=1 to ordem do rrT:= (r[i] * r[i]) + rrT;
   pp[nit+1]:= rrT;
   bb:=pp[nit+1] / pp[nit];
   for i:= 1 to ordem do
    begin
     p[i]:= r[i] + (bb * p[i]);
     ptil[i]:= rtil[i] + (bb * ptil[i]);
    end;
   end;{while}
end;


procedure menu_inicial;
var
 opcao_a: integer;
 contr_a: boolean;
begin
 clrscr;

writeln('************************************************************');
 writeln('*
*');
 writeln('*          METODOS ITERATIVOS NAO ESTACIONARIOS
*');
 writeln('*           PARA RESOLUCAO DE SISTEMAS LINEARES
*');
 writeln('*
*');
 writeln('* ORDEM DA MATRIZ   ',ordem,'
*');
 writeln('*
*');

writeln('************************************************************');
 writeln('');
 writeln('');
 writeln('');
 writeln('ENTRADA DE DADOS  :');
 writeln('');
 writeln('1 - DIGITACAO MANUAL');
 writeln('2 - ARMAZENADO EM ARQUIVO');
 writeln('');
 writeln('');
 contr_a:=FALSE;
 while contr_a=FALSE do
 begin
  write('ENTRADA DE DADOS -> ');
  readln(opcao_a);
  case opcao_a of
   1: begin
       le_coeficientes_manual;
       contr_a:=TRUE;
```

```
      end;
   2: begin
       le_coeficientes_arquivo;
       contr_a:=TRUE;
      end;
  Else
   contr_a:=FALSE;
  end;{case}
 end;{while}
 clrscr;
end;

procedure menu_metodos;
var
 opcao_b: integer;
 contr_b: boolean;
begin
 clrscr;

writeln('************************************************************');
 writeln('*
*');
 writeln('*          METODOS ITERATIVOS NAO ESTACIONARIOS
*');
 writeln('*           PARA RESOLUCAO DE SISTEMAS LINEARES
*');
 writeln('*
*');

writeln('************************************************************');
 writeln('');
 writeln('METODOS : ');
 writeln('');
 writeln('1 - METODO DOS GRADIENTES CONJUGADOS');
 writeln('2 - METODO DOS GRADIENTES CONJUGADOS PRECONDICIONADO');
 writeln('3 - METODO DOS GRADIENTES BICONJUGADOS');
 writeln('');
 writeln('');

writeln('************************************************************');
 writeln('');
 contr_b:=FALSE;
 while contr_b=FALSE do
 begin
  write('METODO ESCOLHIDO ->');
  readln(opcao_b);
  case opcao_b of
   1: begin
       prec:=0;
       metodo_GradConjPre;
       contr_b:=TRUE;
      end;
   2: begin
       escolha_precondicionador;
       metodo_GradConjPre;
       contr_b:=TRUE;
      end;
   3: begin
        metodo_GradBiconj;
        contr_b:=TRUE;
      end;
```

```
  Else
      contr_b:=FALSE;
  end;{case}
 end;{while}
clrscr;
end;

BEGIN
 menu_inicial;
 sair:=FALSE;
 repeat
  inicializa;
  menu_metodos;
  exibe_resultado;
  contr:=false;
  while contr=FALSE do
  begin
  writeln('');
  writeln('');
  writeln('');
  writeln('');

writeln('*******************************************************');
  writeln('1 - Executar novamente');
  writeln('2 - Sair');
  write('OPCAO ->');
  readln(opsair);
   case opsair of
    1: begin
         sair:= TRUE;
         contr:=TRUE;
       end;
    2: begin
         sair:= FALSE;
         contr:=TRUE;
       end;
    Else
     contr:=FALSE;
   end;{case}
  end;{while}
 until sair=FALSE;



END.
```

REFERÊNCIAS BIBLIOGRÁFICAS

Oliveira, J.C.(2001). Comunicação pessoal.

Santos, Votoriano R. de Barros.(1976).Curso de Cálculo Numérico . Livros Técnicos e Científicos, 3ªed, Rio de Janeiro.

Canuto, C., M. Y. Hussaini, et al. (1988). Spectral Methods in Fluid Dynamics, Springer-Verlag Berlin Heidelberg.

Gottlieb, D. and S. A. Orszag (1977). Numerical Analysis of Spectral Methods: Theory and Applications, SIAM (Society of Industrial and Applied Mathematics).

R. Barret, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst (1994).Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM. Philadelphia, PA.

Y. Saad (1996), Iterative Methods for Sparse Linear Systems, PWS Publishing Co., Boston.

L. Hageman, D. Young (1981). Applied Iterative Methods. Academic Press, NY.

D. M. Young (1971). Iterative Solution of Large Linear Systems. Academic Press, NY.

G. H. Golub, C. F. van Loan (1996) . Matrix Computations. The John Hopkins University Press, Baltimore, Maryland.