UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

CURSO DE GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

DISCIPLINA DE CÁLCULO NUMÉRICO

PROFESSOR DR. ALESSANDRO SANTANA

PROFESSOR DR. SANTOS ALBERTO ENRIQUEZ REMIGIO

RESOLUÇÃO DE EQUAÇÃO DIFERENCIAL ORDINÁRIA (EDO) DE SEGUNDA ORDEM, A PARTIR DO MÉTODO DE DIFERENÇAS FINITAS DE SEGUNDA ORDEM RESOLVIDOS DE DIFERENTES MÉTODOS DE RESOLUÇAO DE SISTEMAS LINEARES.

GUILHERME SALOMÃO AGOSTINI - 11721EMT003
LUIZ RENATO RODRIGUES CARNEIRO - 11721EMT004

UBERLÂNDIA

2019

Sumário

INTRODUÇÃO:

O estudo de métodos numéricos é de extrema importância na vida prática de um engenheiro, tendo em vista que a maior parte das integrais e equação diferencial não são possíveis de serem resolvidas analiticamente. Para tal, foram desenvolvidas series matemáticas que convergem ao resultado real e podem ser programadas.

Esse artigo tem por objetivo apresentar um código que resolve equações diferenciais ordinárias de segunda ordem no formato da equação (1), através do método das diferenças finitas (MDF)de segunda ordem.

$$p(x)\frac{d^2u}{dx^2} + q(x)\frac{du}{dx} + r(x)u = f(x), \quad a < x < b, \tag{1}$$

Como o método das diferenças finitas gera um output matricial (sistema linear), ainda, será preciso métodos de resolução de sistemas lineares, onde serão utilizados: Eliminação Gaussiana Sem Pivotamento (MEG), Eliminação Gaussiana com Pivotamento (MEGPP), Gauss-Jacob (MGJ), Gauss-Seidel (MGS) e Sobre Relaxação Sucessiva (SOR).

Para garantir a eficácia e confiabilidade do código, serão aplicados testes: “Teste0”, “Teste1” e “Teste2”.

GEREÇÃO DA MATRIZ DA MATRIZ DO PROBLEMA DE VALOR DE CONTORNO (PVC)

As equações propostas são todas equações diferenciais, e apresentam apenas uma variável independente, dessa forma são todas Equações Diferenciais Ordinárias (EDOs), para encontrar a solução de uma EDO qualquer é necessária a utilização de Métodos Numéricos, pois nem todas as EDOs apresentam solução analítica.

Assim, para encontrar a solução dos Problemas de Valor de Contorno (PVC) será utilizado o Método das Diferenças Finitas (MDF) centradas de segunda ordem, esse é um método que tem origem da expansão da série de Taylor, isolando as derivadas de primeira e segunda ordem, representadas pela equação (2), essa igualdade associa um erro, como esse erro depende do espaçamento h^2, logo esse método é de segunda ordem, um método com ordem maior apresentará um erro associado menor ainda, exigindo menor número de passos.

Substituindo a equação (2) e (1) e colocando os termos as funções *u* em evidência, temos a equação (3).

$$\frac{du_i}{dx} = \frac{u_{i+1} - u_{i-1}}{2(h)} + O(h^2) \qquad (2)$$

$$\frac{d^2u_i}{dx^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{(h)^2} + O(h^2)$$

$$[2p(x_i) - hq(x_i)]u_{i-1} + [2h^2r(x_i) - 4p(x_i)]u_i + [2p(x_i) + hq(x_i)]u_{i+1} = 2h^2f(x_i). \qquad (3)$$

Em que denominaremos alguns termos como:

$$d_i = 2p(x_i) - h * q(x_i) \quad (4)$$

$$a_i = 2 * h^2 * r(x_i) - 4 * p(x_i) \quad (5)$$

$$ci = 2 * p(x_i) + h * q(x_i) \quad (6)$$

$$b_i = 2 * h^2 * f(x_i) \quad (7)$$

Como esse método escolhido é implícito ele não apresenta a solução diretamente, assim ele resulta em um sistema linear, que nesse trabalho será expresso na forma de matriz, que está logo a baixo, que será resolvido depois pelos métodos de resolução de sistema linear, para assim encontrar a solução do PVC.

$$\begin{bmatrix} a_1 & c_1 & 0 & 0 & 0 & \dots & 0 \\ d_2 & a_2 & c_2 & 0 & 0 & \dots & 0 \\ 0 & d_3 & a_3 & c_3 & 0 & \dots & 0 \\ 0 & 0 & d_4 & a_4 & c_4 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_{n-2} & a_{n-2} & c_{n-2} \\ 0 & 0 & 0 & 0 & \dots & d_{n-1} & a_{n-1} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix} \quad (8)$$

Um dos maiores desafios do código é gerar essa matriz das expressões (8), podemos ver que entre a segunda e penúltima linha existe um padrão, as funções $d_i, a_i, c_{i,}, b_i$, com mostradas nas equações (4) ,(5) ,(6) ,(7), são bem definidas, e de fácil implementação, porém a primeira e a última linha são diferentes.

Para a primeira linha *i*=1, assim pela equação (3), temos que o termo que acompanha d_0 está multiplicado por u_0, e$u_0$depende das condições de contorno da função e é dado pela seguinte expressão.

$$u_0 = \frac{2hv_a - 4\alpha_2 u_1 + \alpha_2 u_2}{2h\alpha_1 - 3\alpha_2}$$

Como u_0 e $d_1$são constantes podemos passá-los para o lado direito da equação (3), e assim para a primeira linha do código o b_1 é dado pela expressão a baixo:

$$b_1 = 2 * h^2 * f(x_1) - \frac{2 * h * v_a * (2 * p(x_1) - h * q(x_1)}{2 * h * \alpha_1 - 3\alpha_2} \quad (9)$$

Em que α_1, β_1, v_a, α_2, β_2, v_b é dado pelas expressões da condição de contorno a baixo.

$$\begin{cases} \alpha_1 u(a) + \alpha_2 \frac{du}{dx}(a) = v_a \\ \beta_1 u(b) + \beta_2 \frac{du}{dx}(b) = v_b \end{cases} \quad (10)$$

Da mesma maneira que analisamos a primeira linha também podemos analisar a última linha da matriz,

Para a última linha *i*=n-1, assim pela equação (3), temos que o termo que acompanha c_{n-1} está multiplicado por u_n, e u_n depende das condições de contorno da função e é dado pela seguinte expressão.

$$u_n = \frac{2hv_b + 4\beta_2 u_{n-1} - \beta_2 u_{n-2}}{2h\beta_1 + 3\beta_2} \quad (11)$$

Como u_n e c_{n-1}são constantes podemos passá-los para o lado direito da equação (3), e assim para a última linha do código o b_{n-1}é dado pela expressão a baixo:

$$b_{n-1} = 2 * h^2 * f(x_{n-1}) - \frac{2 * h * v_b * (2 * p(x_{n-1}) + h * q(x_{n-1})}{2 * h * \beta_1 + 3 * \beta_2} \quad \textbf{(12)}$$

Dessa forma, para implementar as matrizes que serão utilizadas nos métodos de resolução de sistemas lineares utilizou-se um laço condicional que criava da linha dois até a linha n-2 da matriz seguindo as equações (4) ,(5) ,(6) ,(7), e a primeira e última linha foram criadas separadamente dadas pelas expressões (9) e (12).

METODOS DE RESOLUÇÕES DE SISTEMAS LINEARES:

Existem duas formas distintas de resolver sistemas lineares:

MÉTODOS DIRETOS

São métodos que geram solução exata a partir de finitas operações matemáticas em teoria, porém devido à representação binária IEEE724 computacional, que utiliza expoentes na base dois, parte do valor numérico real exato é perdido no processo de conversão de um número real decimal em real binário.

Iremos citar abaixo dois exemplos de métodos diretos:

ELIMINAÇÃO GAUSSIANA SEM PIVOTAMENTO (MEG)

A eliminação gaussiana sem pivotamento é um método de resolução baseada em operações elementares algébricas matriciais escalonamento, isto, sem permutar as linhas da matriz.

Escalonamento: é um método de resolução de sistemas ao qual uma das variáveis é isolada e através desta, obtém-se progressivamente todas as outras, como o exemplo abaixo:

$$a + b = 7$$

$$b = 2$$

Ou seja, a vale cinco.

Perceba, que o sistema acima é um sistema triangular superior.

Portanto, o método se resume a dois passos: converter a matriz dos coeficientes em triangular superior; substituir os valores obtidos nas equações subsequentes obtendo assim a solução.

Para converter uma matriz em triangular superior, fixa-se a linha mais superior, e então se zera os coeficientes abaixo desta:

$$a + b = 7$$

$$2a + 6b = 22$$

$$L2 = L2 - L1 * Coef_{mult}$$

Para este exemplo:

$$Coef_{mult} = \frac{2}{1}$$

Perceba: este coeficiente é mantido para toda operação da linha, porém, cada linha tem seu coeficiente multiplicador.

As desvantagens deste método: perceba que se o valor da linha superior ou inferior ser nulo temos o coeficiente tendendo a infinito, ou temos ele igual a zero, que nos leva a solução trivial nula. Outro problema: neste tipo de cálculo, temos a possibilidade de obter um coeficiente multiplicativo com valor alto, o que pode gerar diferenças altas entre os valores das linhas, propiciando a amplitude de erros associados a arredondamentos.

ELIMINAÇÃO GAUSSIANA COM PIVOTEAMENTO PARCIAL

Este método é a evolução do MEG: utilizando a operação de ‘’permutação’’, trocar a localização das linhas entre si.

Já que o problema avaliado no MEG estava associado ao coeficiente multiplicador, neste método buscamos posicionar os valores nulos de forma a o coeficiente multiplicador nulo, infinito ou tendendo a infinito.

Os passos são similares, a única alteração se trata: antes de efetuar a operação de zerar as constantes das linhas inferiores ao pivô, escolhe-se para ser a linha superior (ou pivô) a que possui maior valor na constante da coluna a ser zerada. Desta forma garantimos que o coeficiente é diferente de zero ou infinito, e também, limitamos o coeficiente multiplicativo em torno do módulo de um, evitando a disparidade entre os valores das linhas e amenizando o erro de arredondamento.

MÉTODOS ITERATIVOS

São métodos que dependem de certo número de repetições (iterações), e dependem de um método matemático convergente: um valor inicial é dado então através das múltiplas iterações chega-se ao resultado aproximado.

Como sua convergência não depende da representação fiel do valor, estes métodos geralmente não carregam erros de arredondamento, porém carregam consigo condições especiais de aplicação e utilizam mais processamento, causado pelas múltiplas iterações acompanhado de um critério de parada. Por esse motivo é recomendado para sistemas esparsos (sistemas com muitos coeficientes nulos).

Nesse trabalho serão usados dois métodos iterativos Método de Gauss-Jacobi e método de Gauss-Seidel, a base de funcionamento desses dois métodos é a mesma, e se baseiam em transformar o sistema linear (13)no sistema linear equivalente (14).

(13)

$$A * x = b$$

(14)

$$x = G * x + d$$

Em que *A* é a matriz dos coeficientes, *b* vetor dos termos independentes, *x* o vetor solução, *G* a matriz de iteração do método utilizado, e *d* é o termo independente do método utilizado.

Assim a através de um método iterativo temos a equação (15)

(15)

$$x^{(k+1)} = G * x^{(k)} + d$$

Em que *k* é a o número da iteração realizada, e inicializando em *k*=0, dessa forma, é necessário dar o vetor aproximação ($x^{(0)}$) que dará início ao método iterativo, quanto mais próximo esse vetor estiver da solução, menos iterações serão necessárias e assim mais rápidas será sua convergência.

MÉTODO DE GAUSS-JACOBI (MGJ)

Assim, para (MGJ) a equação (15)pode ser representada da seguinte forma:

(16)

$$\begin{cases} x_1^{(k+1)} = \dfrac{b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} - \cdots - a_{1n}x_n^{(k)}}{a_{11}} \\ x_2^{(k+1)} = \dfrac{b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} - \cdots - a_{2n}x_n^{(k)}}{a_{22}} \\ x_3^{(k+1)} = \dfrac{b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} - a_{34}x_4^{(k)} - \cdots - a_{3n}x_n^{(k)}}{a_{33}} \\ \vdots \quad \vdots \qquad \vdots \\ x_n^{(k+1)} = \dfrac{b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - a_{n3}x_3^{(k+1)} - \cdots - a_{nn-1}x_{n-1}^{(k+1)}}{a_{nn}} \end{cases}$$

Dessa forma, conforme o sistema linear (16) podemos ver que todos os valores utilizados em uma mesma iteração k para gerar os valores $x^{(k+1)}$ são apenas dos resultados obtidos na iteração anterior, ou seja, utilizam apenas os valores de $x^{(k)}$.

E no MGJ a equação (15)é dada pela seguinte forma:

(17)

$$x^{(k+1)} = G_{MGJ} * x^{(k)} + d_{MGJ}$$

Em que $G_{MGJ} = -D^{-1} * (L + U)$,$d_{MGJ} = D^{-1} * b$ e L+D+U=A, pois L é a matriz triangular inferior, U é a matriz triangular superior e D a matriz diagonal.

(18)

$$x_i{}^{(k+1)} = \frac{1}{a_{ii}} * \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} * x_j{}^{k} \right]$$

MÉTODO DE GAUSS-SEIDEL (MGS)

Semelhante ao método MGJ o MGS também gera um sistema linear na sua resolução por iterações, porém utilizam-se os valores recém calculados dentro da mesma iteração para encontrar o valore dos coeficientes subsequentes, conforme ilustrado no sistema linear (19). Por esse motivo é um método que proporciona uma convergência mais rápida, e assim exigindo menos iterações.

(19)

$$\begin{cases} x_1^{(k+1)} = \dfrac{b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} - \cdots - a_{1n}x_n^{(k)}}{a_{11}} \\ x_2^{(k+1)} = \dfrac{b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} - \cdots - a_{2n}x_n^{(k)}}{a_{22}} \\ x_3^{(k+1)} = \dfrac{b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} - a_{34}x_4^{(k)} - \cdots - a_{3n}x_n^{(k)}}{a_{33}} \\ \vdots \quad \vdots \qquad \vdots \\ x_n^{(k+1)} = \dfrac{b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - a_{n3}x_3^{(k+1)} - \cdots - a_{nn-1}x_{n-1}^{(k+1)}}{a_{nn}} \end{cases}$$

E no MGS a equação (15)é dada pela seguinte forma:

(20)

$$x^{(k+1)} = G_{MGS} * x^{(k)} + d_{MGS}$$

Em que $G_{MGS} = -(L + D)^{-1} * U$ e$d_{MGJ} = (L + D)^{-1} * b$

(21)

$$x_i{}^{(k+1)} = \frac{1}{a_{ii}} * \left[b_i - \sum_{j=1}^{i-1} a_{ij} * x_j{}^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} * x_j{}^{k} \right]$$

MÉTODO DE SOBRE RELAXAÇÃO SUCESSIVA (SOR)

O método de SOR é uma melhoria do MGS, pois a partir da equação (21)multiplicou a equação por ω eadicionou-se um termo a mais na equação, assim gerando um a variável ω para refinamento, caso a escolha dessa variável seja boa é possível acelerar a convergência para a solução do sistema linear, caso$\omega = 1$, a equação (22)se torna a mesma equação do MGS (equação (21)). Porém a escolha dessa variável é difícil, em virtude dos teoremas existentes só serem aplicáveis em um grupo específico de equação.

(22)

$$x_i{}^{(k+1)} = (1 - \omega) * x_i{}^{k} + \frac{\omega}{a_{ii}} * \left[b_i - \sum_{j=1}^{i-1} a_{ij} * x_j{}^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} * x_j{}^{k} \right]$$

CRITÉRIO DE CONVERGÊNCIA DOS METODOS ITERATIVOS

Como havia sido informado anteriormente um método iterativo exige um critério de convergência para saber se após diversas iterações o método irá convergir pra a solução real.

Para esses dois métodos há um critério que diz que caso a matriz dos coeficientes seja diagonal dominante então haverá garantia de convergência independente da aproximação inicial, porém nem todo sistema linear que converge é diagonal dominante.

Uma matriz é diagonal dominante se atender a condição da equação (23), ou se apresentar essa inequação válida pelo menos par uma linha e o somatório seja igual ao elemento da diagonal principal nos demais valores.

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \forall i = 1, 2, \ldots, n \tag{23}$$

Porém um critério necessário para a convergência de um sistema de equações é todos os autovalores da matriz de iteração G ser menores do que um, ou seja:

$$\max_{i \leq i \leq n} |\lambda_i| < 1 \tag{24}$$

Em que λ_i são os autovalores da matriz de iteração G do método utilizado.

ENTENDENDO O CÓDIGO E OS TESTES:

Como o objetivo do código era programar em um mesmo arquivo todos os métodos de resolução de sistemas lineares e testes, a equipe utilizou-se de um *menu* que perguntará ao usuário seu desejo, como na Figura 1.

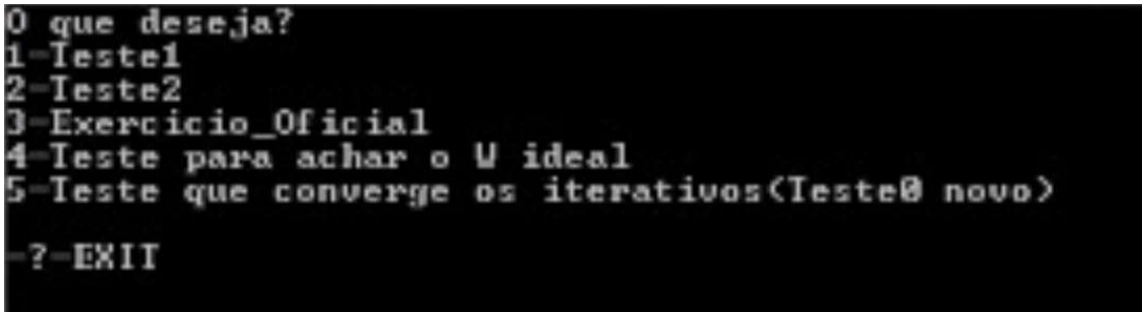

Figura 1– Menu, ao executar o código.

Ao visualizar o *menu*, o exercício proposto nos mostra quatro testes e uma resolução.

A resolução da EDO proposta, esta na opção “3” e printa as respostas no mesmo terminal de console. Caso o usuário necessite da matriz output do MDF, esta estará disponível em um arquivo “Matriz_exercicio_proposto.txt”. Caso precise das soluções, “Solucao_exercicio_proposto.txt”.

O “Teste1” e o “Teste2” têm objetivo de mostrar ao usuário a ordem de erro do Método de Diferenças Finitas_ que será mostrado direto no console do terminal do sistema e criará um arquivo com o sistema linear de nome “Matriz_teste1.txt” e “Matriz_teste2.txt”, respectivamente. O motivo da escolha está nas diferentes condições iniciais de contorno onde o teste1 é Dirichlet e o teste2 de Robin.

A importância da verificação da ordem do erro no método está na confiabilidade matemática do código.

O “Teste5” tem por objetivo mostrar que os métodos de resolução dos sistemas lineares estão em funcionamento completo. Ao executá-lo, mostrará no próprio console a solução da analítica e a solução obtida pelos diferentes métodos de resolução. Além de tudo, nos métodos iterativos, mostrará o tempo que foi necessário para sua utilização. Ao executar do teste 5 o executável criará um

arquivo de nome “Matriz_teste5.txt” que apresentará a matriz. Caso o usuário precise dos resultados, no arquivo “Solucao_teste5.txt”.

O “Teste4” é uma otimização de um parâmetro do método de Sobre Relaxação Sucessivas (SOR) para o problema proposto.

Ao final de qualquer “modo” escolhido no *menu*, o programa chegará ao fim e precisará ser reaberto.

Do ponto de vista de lógica construtiva do programa, os “testes” são variáveis de controle que habilitam ou desabilitam funções ou loops dentro do código principal. Este tipo de lógica precisou ser implementada uma vez que não era possível realizar a separação dos métodos e funções em outros arquivos executáveis.

OBSERVAÇÕES DO TESTE 1 E DO TESTE 2

O teste 1 e o teste 2 são EDOs de segunda ordem e estão apresentadas pelas equações (25) e (26), respectivamente.

$$\begin{cases} \dfrac{d^2u}{dx^2} + u = x^2e^{-x} \quad , \quad 0 < x < 10 \\ u(0) = 0 \quad , \quad u(10) = 0 \end{cases} \tag{25}$$

$$\begin{cases} -\dfrac{d^2u}{dx^2} + u = 2cos(x) \quad , \quad \dfrac{\pi}{2} < x < \pi \\ u'\left(\dfrac{\pi}{2}\right) + 3u\left(\dfrac{\pi}{2}\right) = -1 \quad , \quad u'(\pi) + 4u(\pi) = -4 \end{cases} \tag{26}$$

Estes testes possuem solução analítica representados pelas equações (27) e (28), respectivamente.

$$u(x) = \frac{1}{2}e^{-x}\left[1 + 2x + x^2 - e^x cos(x) - 2e^x\left(-\frac{cot(10)}{2} + \frac{121cosec(10)}{2e^{10}}\right)sin(x)\right] \tag{27}$$

$$u(x) = cos(x). \tag{28}$$

Estas EDOs tem por objetivo central mostrar a validade do método matemático envolvido verificando a ordem do decaimento do erro em relação ao número de passos *np*, neste caso, de segunda ordem.

A segunda ordem no decaimento do erro significa que a diferença modular entre a resposta analítica e a obtida será reduzida pela metade, se o numero de passos for dobrado (uma vez que o numero de passos é inversamente proporcional ao h - distância de ponto a ponto calculado).

Pois então, no código, utilizando o Método de Gauss Jacob com Pivotamento Parcial (MEGPP) foi inserido um loop, representado pela Figura 2, controlado por um iterante *cy*. Este loop tem por função repetir toda a resolução por cinco vezes

consecutivas, sendo que, ao final de cada vez o numero de passos é dobrado, um novo sistema linear é montado e calculado_ por isso a alocação dinâmica de memória.

```
for(cy=0;cy<5;cy++){


    sol_analitica =  (double*)malloc((np+1)*sizeof(double));

    A = (double**)malloc(np*sizeof(double*));
    for(c=0;c<np;c++)
        A[c]=(double*)malloc(np*sizeof(double));

    u = (double*)malloc((np+1)*sizeof(double));

    b = (double*)malloc(np*sizeof(double));


    A_aux = (double**)malloc(np*sizeof(double*));
    for(c=0;c<np;c++)
        A_aux[c]=(double*)malloc(np*sizeof(double));



    b_aux = (double*)malloc(np*sizeof(double));



    Enmax=0;
```

Figura 2 - Loop para o cálculo da ordem.

No final da criação do sistema linear as respostas são calculadas em cima das equações analíticas e então armazenadas no vetor *sol_analitica*. No MEGPP, definimos que *En,* o erro de cada ponto, como a diferença entre o vetor da solução analítica e a solução calculada obtida via resolução do sistema linear, Figura 3.

Como desejamos a norma infinita do vetor em módulo, aplicamos dois “if”: o primeiro faz com que o erro seja sempre um valor positivo e o segundo procura no vetor com maior valor de erro.

```
for(i=1;i<=n;i++){
    if(teste3||teste5){
        printf("x%d = %lf\n",i,x[i]);
        fprintf(arqs,"%lf\n",x[i]);
        }
    En=sol_analitica[i]-x[i];
    if(En<0)
        En=-1.0*En;
    if(En>Enmax)
        Enmax=En;

}
if(teste3||teste5){
    printf("x%d = %lf\n",np,u[np]);
    fprintf(arqs,"%lf\n\n",u[np]);}
ordem=log2(E2n/Enmax);
if(teste1||teste2)
    printf("\nOrdem=%lf (np=%d=>np=%d)\n",ordem,np/2,np);
E2n=Enmax;
        //RECUPERAÇÃO DA MATRIZ ORIGINAL
```

Figura 3 - Lógica para o calculo da ordem no MEGPP

Quando o loop das soluções termina, temos então a norma infinita do vetor erro apresentada na variável *Enmax.*

Depois de calculada a norma infinita do erro do vetor para o numero de passos do caso, calculamos a ordem e depois dizemos que esta norma é a do loop "anterior" representado pela variável *E2n.*

Isto explica o motivo do primeiro cálculo de ordem apresentar um valor alto e irreal: o primeiro *Enmax* não terá um *E2n* compatível do loop anterior. Este comportamento pode ser visualizado pela Figura 4. A partir da segunda iteração, o código já possui um *E2n* compatível e, portanto a ordem do erro é confiável.

$$Ordem = \log_2 \frac{E2n}{Enmax} \quad (29)$$

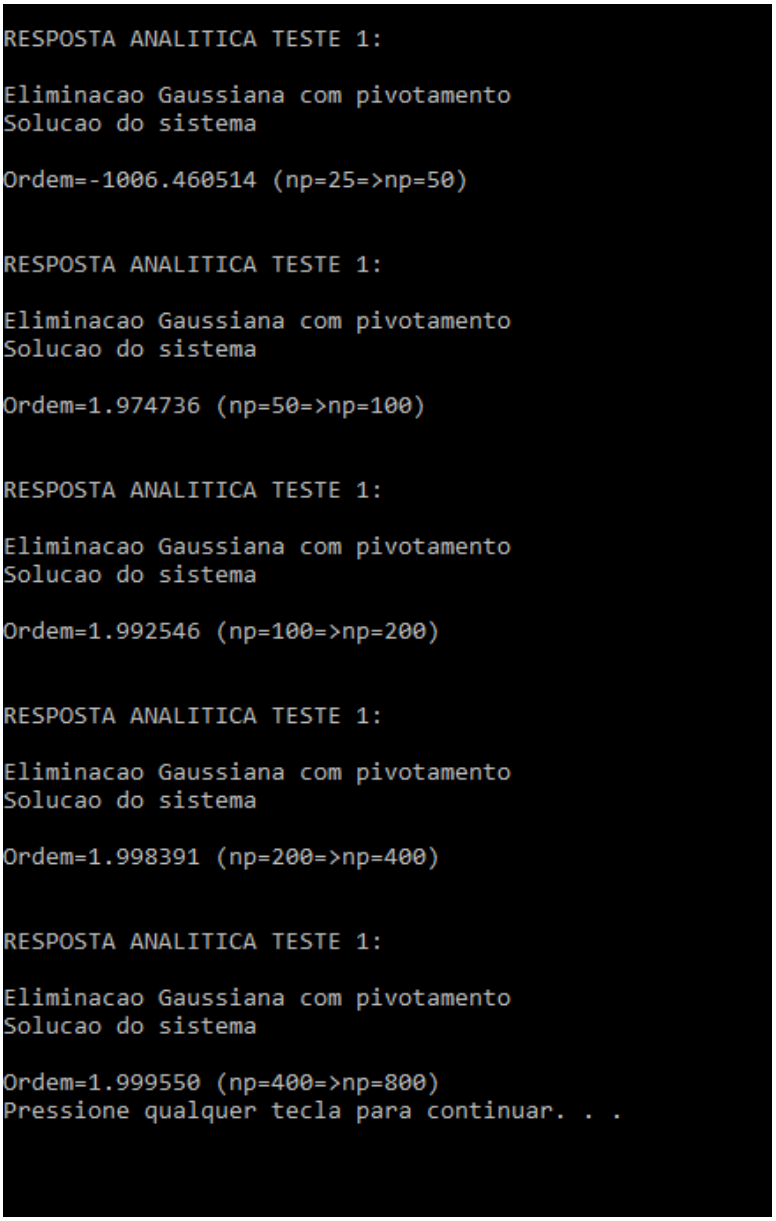

Figura 4 - Output da verificação da ordem do MDF.

Para evitar que o *Enmax* do loop anterior afete *En* do loop novo, este é zerado no começo de cada iteração.

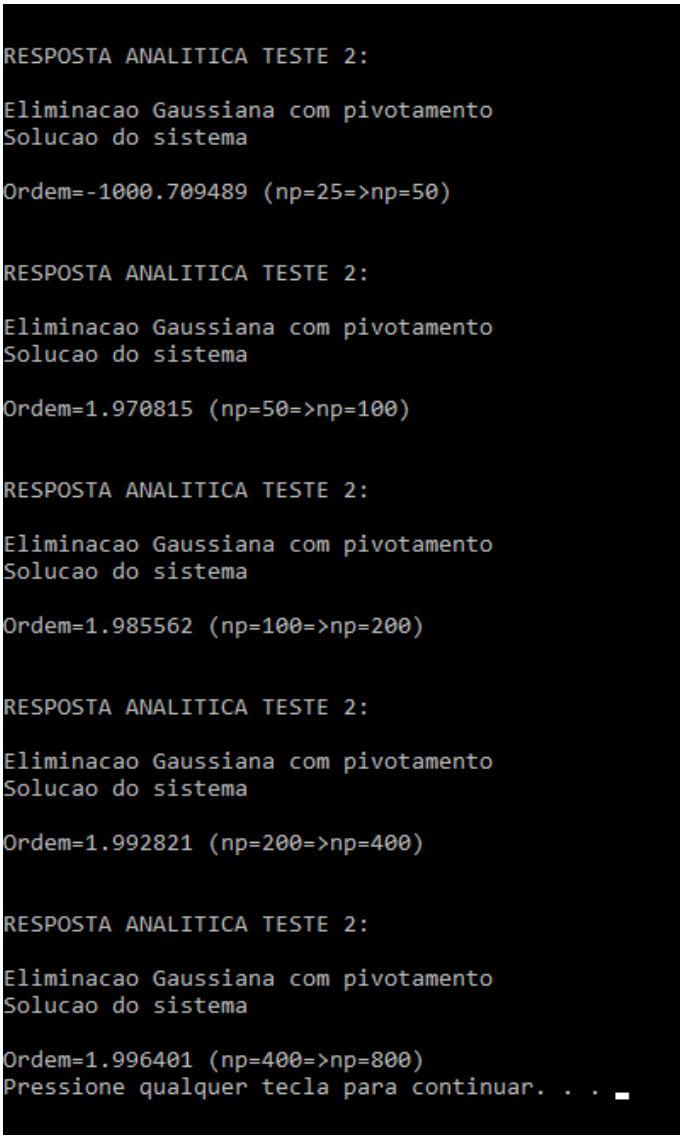

Figura 5 - Output da verificação de ordem do MDF no teste 2.

Como é possível observar foi encontrada, então, a ordem do decaimento do erro de aproximadamente dois, mostrando a confiabilidade do código.

OBSERVAÇÕES DO TESTE 5

O teste número 5 foi empregado para verificar a convergência dos métodos de resolução de sistemas lineares, pois os métodos iterativos exigem a satisfação de critérios de convergência assim como foi dito na parte que explica os métodos iterativos.

O teste 1 e teste 2 não foi possível de achar a solução para os testes iterativos, pois a matriz gerada pelo método das diferenças finitas não deve satisfazer a condição dos módulos dos autovalores serem menores que 1, descrita na expressão (24). Assim o teste 5 é um caso específico de EDO de segunda ordem que apresenta apenas o termo de segunda ordem e essa condição gera uma matriz que garante a condição da expressão (24).

Assim da mesma maneira que os teste 1 e 2 foram implementados o teste 5 também foi, alterando apenas as funções utilizadas e as condições de contorno, porém nele não havia o laço condicional que variaria o número de passos para o cálculo da ordem, e nele seria printada a solução analítica, e as soluções aproximadas para os 5 métodos de resolução de sistema linear. Com essas soluções calculadas foi gerado um arquivo com esses dados que foi utilizado para plotar um gráfico no software *Excell*, comparando todas as soluções simultaneamente.

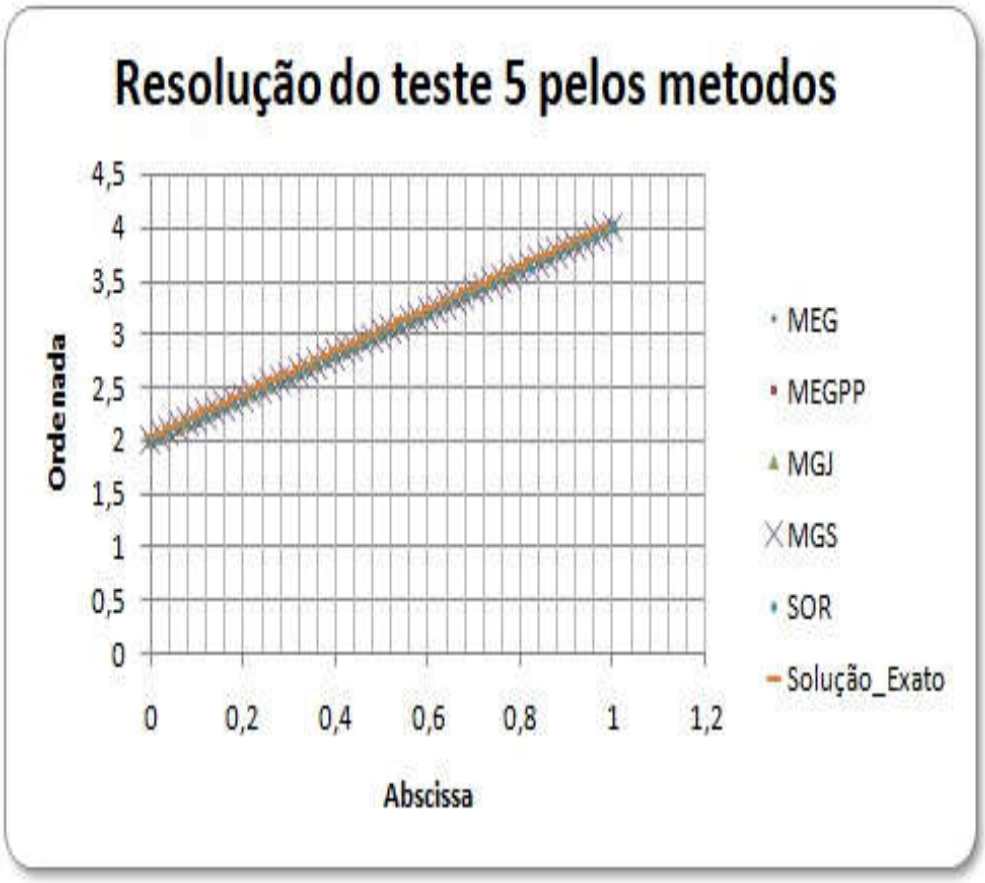

Figura 6 – Comparação entre as soluções aproximadas e a solução analítica para número de passo igual a 50

Como as soluções deram muito próximas elas se sobrepujam, e para facilitar a visualização foi utilizada a representação apenas dos pontos calculados. Ao analisar o gráfico podemos garantir que os métodos estão convergindo como era o esperado, mas não podemos garantir que a matriz das diferenças finitas é de ordem 2, pois essa ordem está relacionada com o quanto a melhoria do espaçamento influencia na melhoria da solução, assim reforço a importância do teste 1 e teste 2.

Além disso, é possível analisar no terminar quando se executa o programa que a solução do SOR e dos métodos analíticos deram mais próximas da solução exata para esse caso específico, utilizado $w = 1{,}98$.

OBSERVAÇÕES DO TESTE 4: REFINAMENTO DO VALOR DA CONSTANTE ω DO SOR

Ao pesquisar na literatura sobre o método de resolução de sistemas lineares via SOR, vimos que é um método muito similar ao MGS, porém com alguns termos a mais que dependem da constante ω, infelizmente é difícil estipular o valor para essa constante analiticamente, mas seus valores de maior eficácia estão entre zero e dois.

Para fazer o refinamento do valor dessa constante para o problema proposto fizemos um método iterativo que iria calcular a solução do sistema linear via o método de SOR diversas vezes, nesse teste utilizou-se o número de passos fixos e igual a 50, iniciando a constante em $\omega = 0{,}01$ e finalizando em $\omega = 1{,}99$, em cada iteração o código armazenaria o número de iterações do SOR e armazenaria o valor do ω e de iterações, sempre armazenando os de menores valores, e no final da execução do programa ele printa no terminal qual é o valor de ω que resulta em menor número de iterações e assim menor tempo de processamento. O resultado desse refinamento pode ser visto a seguir:

```
Metodo de Sobre relaxacao sucessiva
numero de iteracoes = 455       w=1.970000      Tempo = 0.000000e+000

Metodo de Sobre relaxacao sucessiva
numero de iteracoes = 692       w=1.980000      Tempo = 1.600000e-002

Metodo de Sobre relaxacao sucessiva
numero de iteracoes = 1392      w=1.990000      Tempo = 1.500000e-002
w otimizado = 1.920000
```

Figura 7 - Refinamento do ω para o problema proposto e número de passos igual a 50

Era possível achar o valor de ω com uma resolução melhor, porém a execução do programa demandaria um tempo muito mais elevado e por isso foi feito o refinamento dessa constante com apenas duas casas decimais.

EXERCÍCIO PROPOSTO

O exercício proposto é resolvido quando o usuário escolhe a opção “*3*” do *menu*. Logo após escolhido a opção 3, o console pedirá um valor “n” ao qual altera, de acordo com a Figura 8 e Figura 9.

```
if (teste3||teste4){
    int ene;
    arq = fopen("Matriz_exercicio_proposto.txt","w");
    arqs = fopen("Solucao_exercicio_proposto.txt","w");
    printf("Qual o valor do n?\n");
    printf("Valores recomendados 10, 20 , 80, 160, 320\n");
    scanf("%d",&ene);
    double epsilon=1.0/ene;
```

Figura 8 - Parâmetro recebido pelo leitor alterando o parâmetro épsilon.

```
56  double f(double X, double epsilon){//indice do termo independente do teste 1
57      return epsilon*exp(-X*X);
58
59  }
485             //CALCULO DA PRIMEIRA LINHA DE A, e DE b
486             A[1][1]=2.0*h*h*r(X)-4.0*p(X)-4.0*alfa2*(2.0*p(X)-h*q(X))/(2.0*h*alfa1-3.0*alfa2);//calculo a[1];
487             A[1][2]=2.0*p(X)+h*q(X)+alfa2*(2.0*p(X)-h*q(X))/(2*h*alfa1-3.0*alfa2);
488             b[1]=2.0*h*h*f(X,epsilon)-(2*h*va)*(2.0*p(X)-h*q(X))/(2*h*alfa1-3.0*alfa2);
```

Figura 9 - Parâmetro épsilon entrando na função proposta.

O programa então criará um sistema linear e mostrará ao leitor os resultados.

ANALISE DOS RESULTADOS

Como o código é executado para todos os métodos de resolução linear, resolvemos comparar o comportamento dos métodos frente à variação do *n.* Os gráficos foram criados a partir do software *“Excell”* com os dados disponibilizados pelo arquivo gerado “Solucao_exercicio_proposto.txt”.

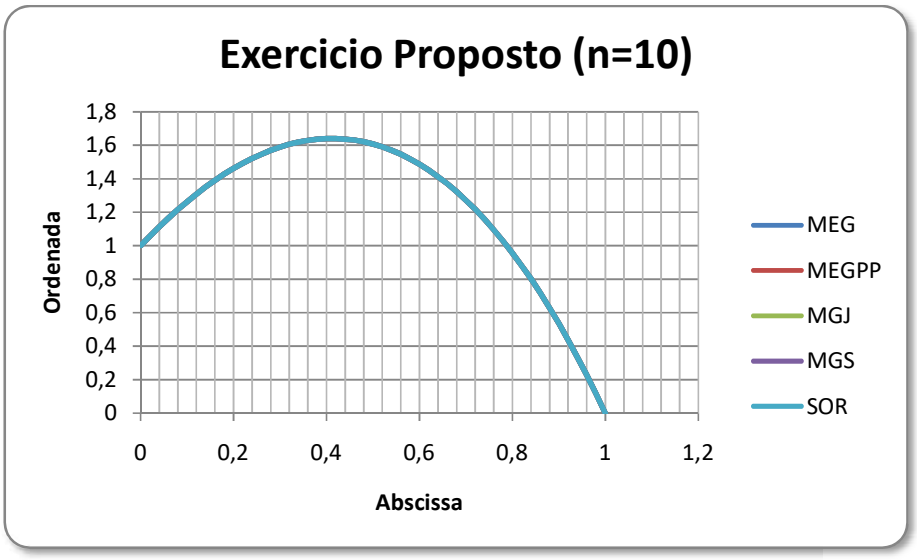

Figura 10 - Comparação do resultado frente aos métodos de resolução com número de passo igual a 50

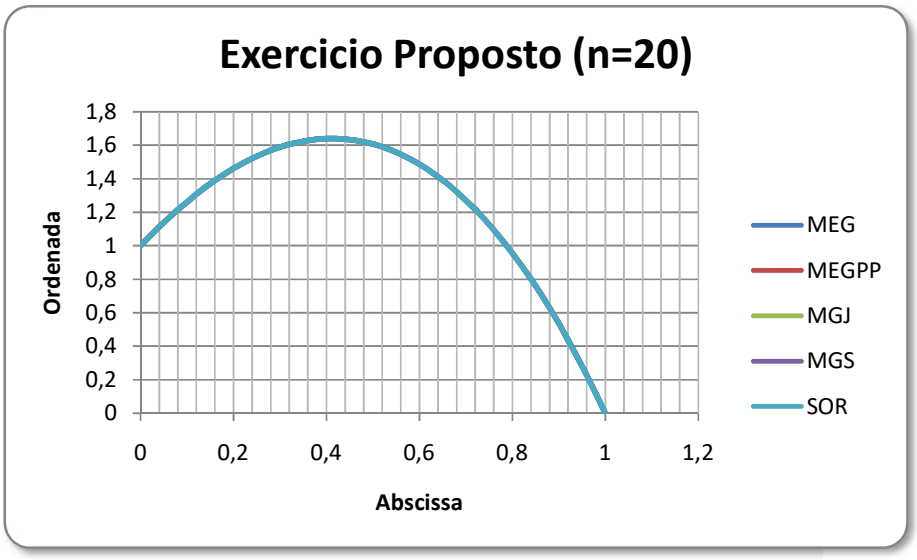

Figura 11 - Comparação do resultado frente aos métodos de resolução com número de passo igual a 50

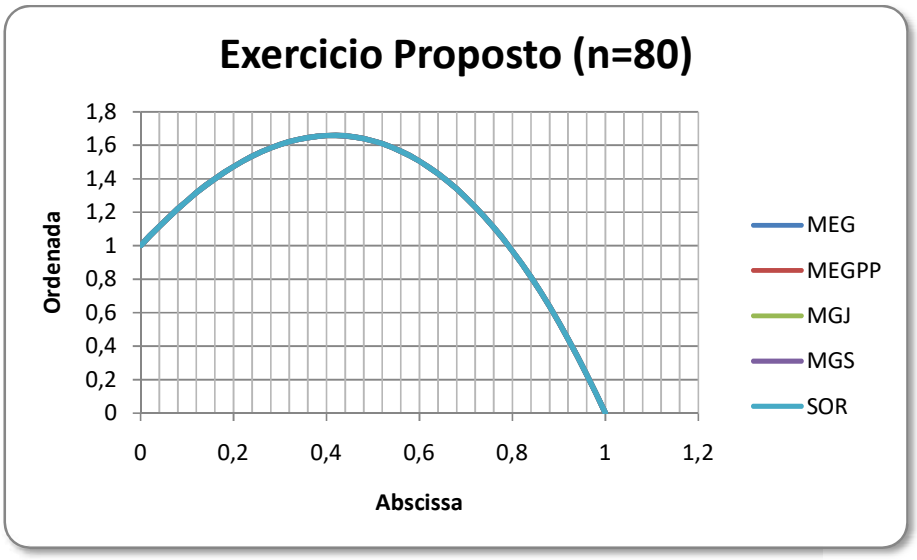

Figura 12 - Comparação do resultado frente aos métodos de resolução com número de passo igual a 50

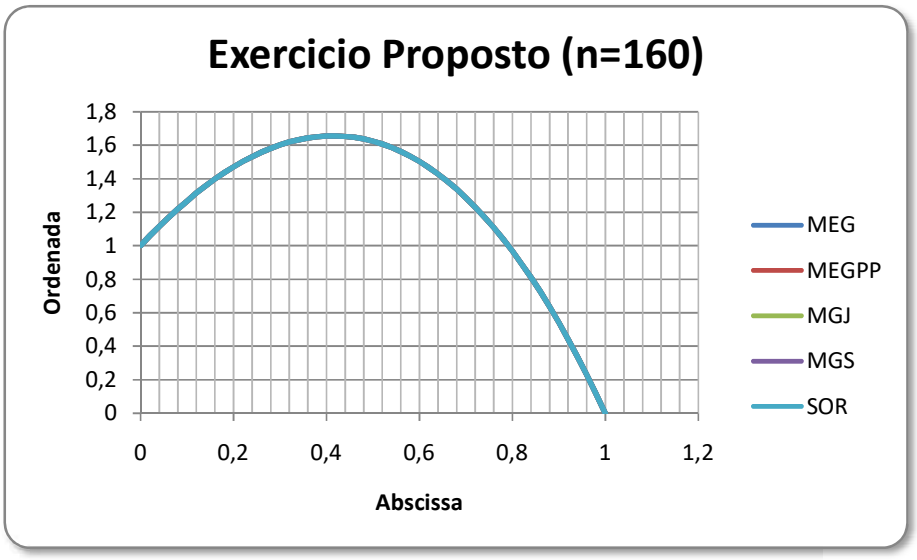

Figura 13 - Comparação do resultado frente aos métodos de resolução com número de passo igual a 50

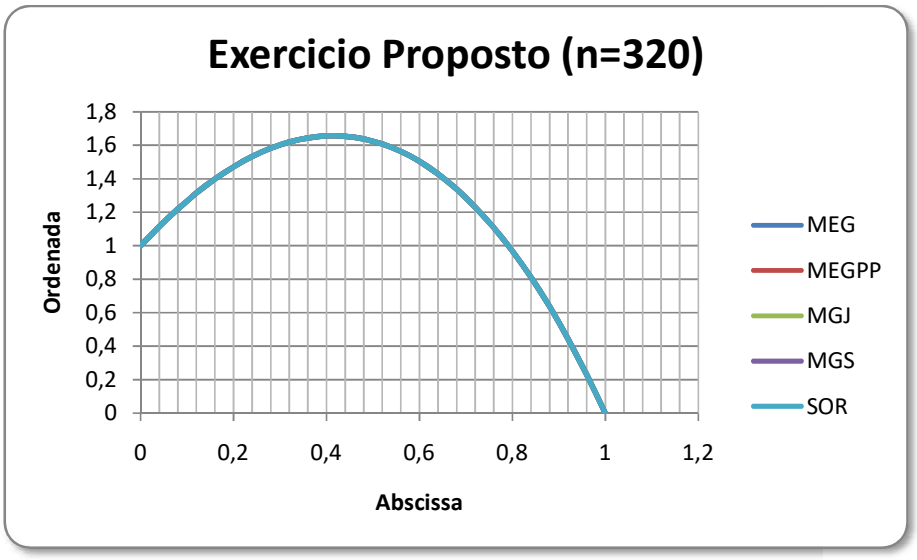

Figura 14 - Comparação do resultado frente aos métodos de resolução com número de passo igual a 50

Como é possível perceber, não existe variação na convergência de nenhum método frente à variação do "*n*".

Para analisar com maiores detalhes o resultado do exercício proposto, Figura 15.

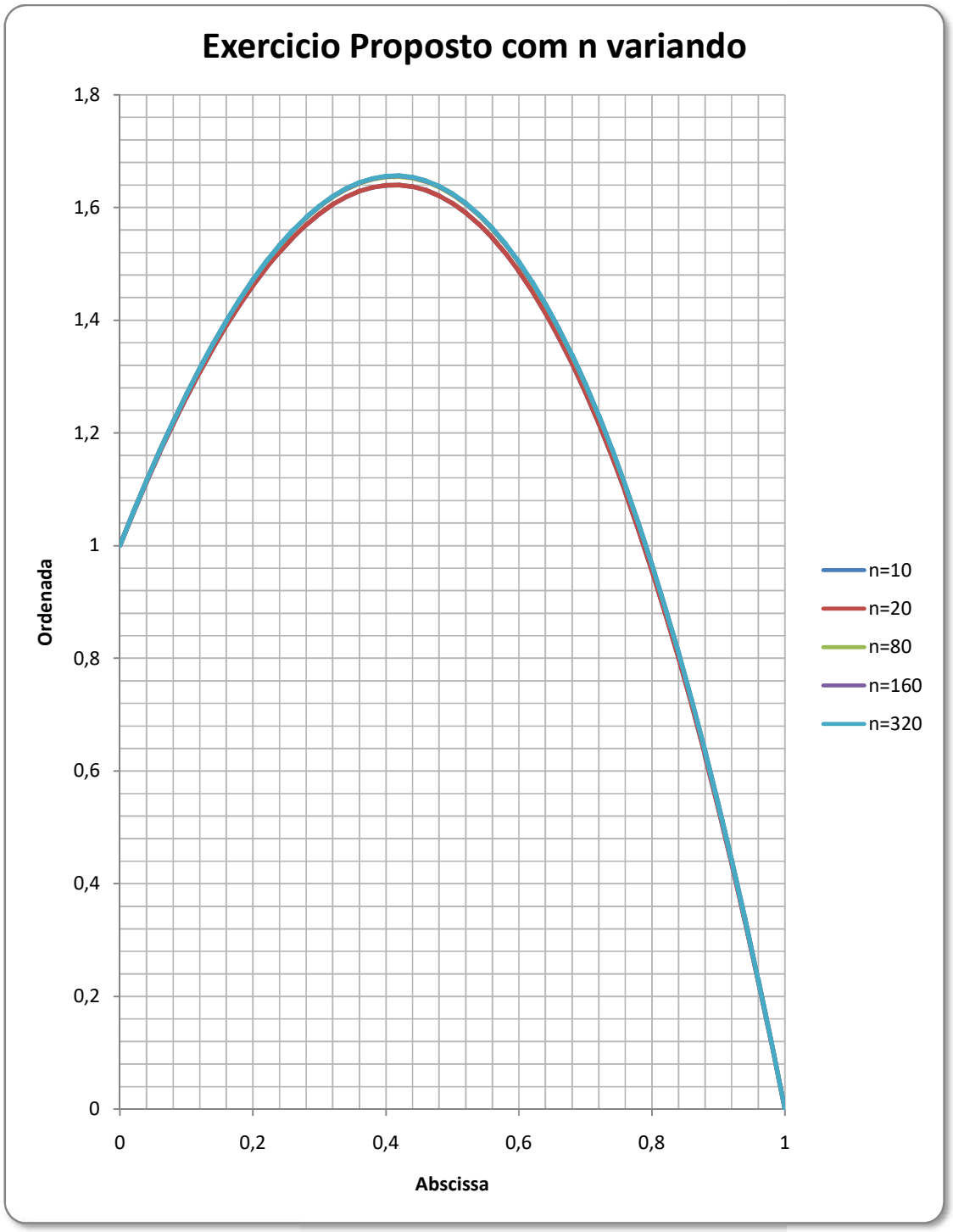

Figura 15 - Comparação do resultado frente à variação do *n* com número de passo igual a 50

Nele podemos concluir ainda, que no intervalo de [0,1] a função possui um máximo local e que a função resposta da EDO é pouco dependente do "*n*" quando $n \geq 1$.

UTILIZANDO O CÓDIGO PARA RESOLVER OUTRA EDO (NÃO PROPOSTA):

Antes de resolver um problema via métodos númericos é preciso modelar o fenômeno analisado e adequá-lo à ferramenta ou função.

Este código foi implementado especificamente para resolver EDO`s de segunda ordem por MDF.

Para adequar qualquer problema à ele é preciso acessar o seu arquivo programável (*.c*) e alterar os parâmetros *P, Q, R* e *F*. Estes parâmetros são as funções que acompanham os diferenciais e aparecerão no código como subrotinas. Haverá muitas outras subrotinas que servem para mostrar a eficácia do método de resolução: altere somente as que não tiverem indice, como mostrado na Figura 16.

```
double p(double X){// indice da derivada segunda do PVC
double q(double X){//indice da derivada primeira do PVC
double r(double X){//indice da derivada de ordem 0 do PVC
double f(double X){//indice do termo independente do teste
```

Figura 16 - Sub-rotinas que podem ser alteradas.

Além disso, o código aceita condições iniciais de contorno de Dirichlet, Neumann e Robin. Para isto, o código foi construido conforme a equação (30).

$$\begin{cases} \alpha_1 u(a) + \alpha_2 \dfrac{du}{dx}(a) = v_a \\ \\ \beta_1 u(b) + \beta_2 \dfrac{du}{dx}(b) = v_b \end{cases} \qquad (30)$$

Para alterar estes parâmetros, procure dentro do rotina *"main"* a Figura 17.

```
//------------------------------------------------------ EXERCICIO OFICIAL ---------------------

if (teste3||teste4){

    arq = fopen("Matriz_exercicio_proposto.txt","w");
    int ene=160;
    double epsilon=1/ene;
    alfa1=1; alfa2=0; va=1; beta1=1; beta2=0; vb=0;// constantes das condicoes de contorno
    x_inicial=0;
    x_final=1;
    h=(x_final-x_inicial)/np;// calculo do espassamento
    N=1; // contador do loop para atualizar o h

    //printf("u[0] = %lf, u[%d] = %lf\n", u[0], np, u[np]);


    //Criando uma matriz [np-2][np-2] de zeros
    for(i=1;i<=np-1;i++){
        for(j=1;j<=np-1;j++){
            A[i][j]=0;                  //zera a matriz dos coeficientes
        }
    }
```

Figura 17 - Local do código para alterar os parâmetros.

Outro parâmetro que pode ser de vontade alteração é o número de passos "*np*", encontrado na rotina principal (*main)* na declaração das variáveis.

```
int np=50;//numero de passos
```

Figura 18 - Imagem da declaração destas variáveis.

CONCLUSÃO:

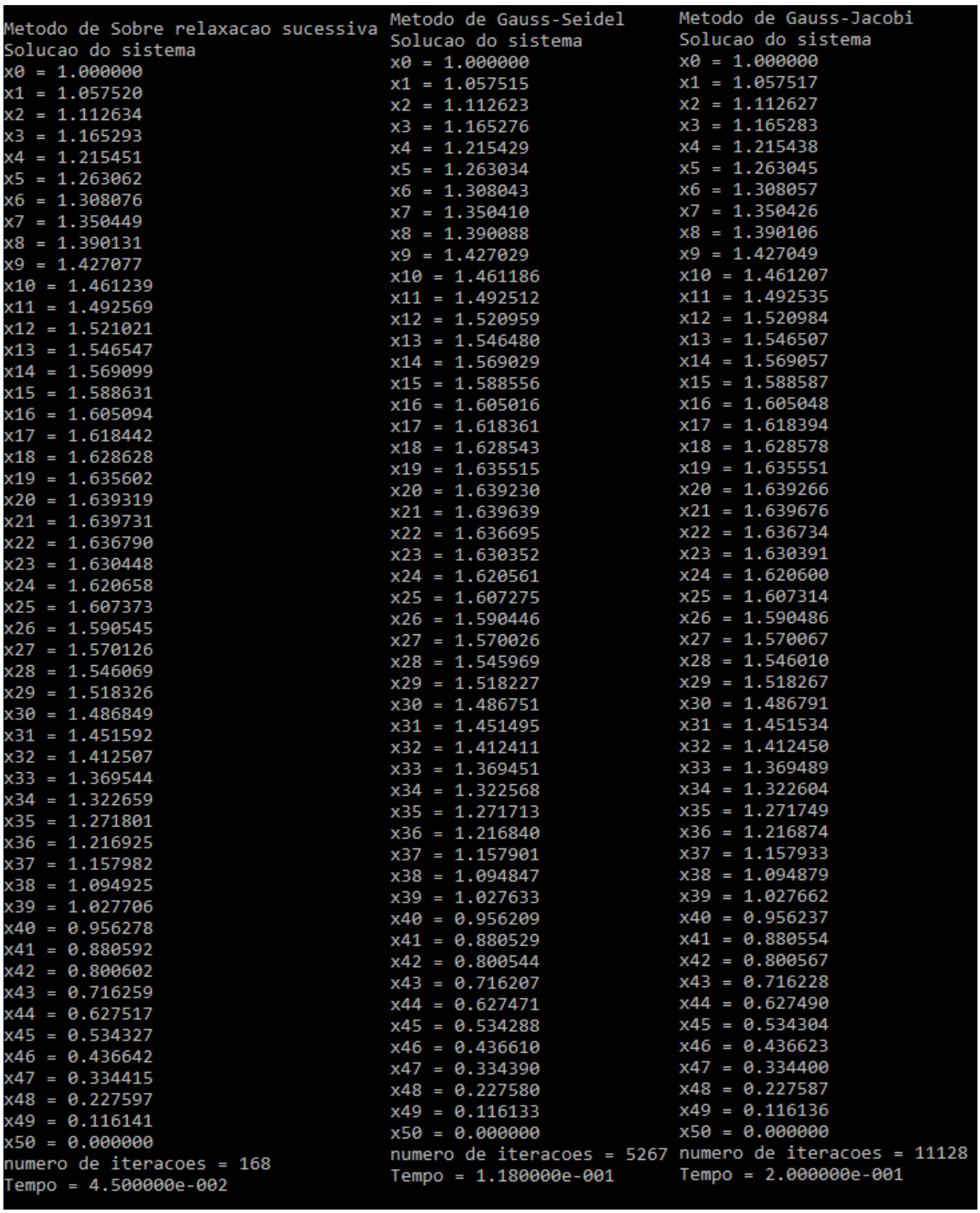

Figura 19 - Iterações e tempo de cada método iterativo quando n=10 e número de passos = 50.

Como é possível concluir pela Figura 19, a diferença entre as iterações utilizadas em cada método é gritante, com o SOR ($\omega = 1.92$), calculado no teste 4, sendo o mais eficaz e, portanto mais rápido; logo em seguida, temos o MGS como o intermediário e o MGJ como o mais lento de todos.

Além disso, pode-se observar que para o exercício proposto todas as soluções convergiram com número de passo igual a 50, assim a matriz das diferenças finitas satisfaz a condição da expressão (24). Outra observação: ao variar o *n*, não houve significativa mudança da solução do PVC, assim a expressão que acompanha o *n,* apresenta pouca influência em relação à curva da solução.