

UFSC – Universidade Federal de Santa Catarina

Nome: Leonardo Vailatti Eichstaedt

Matrícula: 14100847

Cálculo Numérico para Computação

Profº Sergio Peters

Trabalho 1

a) Ao analisar o sistema, não podemos afirmar que o sistema possui convergência garantida, pois segundo o Critério de Scarborough, para convergir o sistema deveria possuir diagonal dominante.

O sistema em questão não possui diagonal dominante. Para verificarmos basta observar que para um sistema possuir diagonal dominante devem ocorrer duas condições:

$$1^{\circ}) |a_{ii}| \geq S_i, i = 1, \dots, n \text{ e}$$

$$2^{\circ}) |a_{ii}| > S_i, \text{ para pelo menos uma linha } i \text{ de } A.$$

Obs : $S_i \rightarrow$ soma dos coeficientes fora da diagonal principal.

A primeira condição é satisfeita já que em todas as linhas $|a_{ii}| = S_i$, porém a segunda condição nunca é satisfeita pois em nenhuma linha $|a_{ii}| > S_i$.

Como o sistema de equações lineares não satisfaz os critérios de convergência, o processo iterativo poderá oscilar ou até mesmo divergir. Neste caso é recomendado usar fatores de sub ou sobre-relaxação em cada equação.

b) **Valores iniciais:**

Chute da valores iniciais como 0s:

```
for k=1:100
    xi(k) = 0;
end
```

Limite de Iterações = 2000 (o uso de limite como 1000 não foi suficiente para o programa parar pelo critério)

```
limite = 2000;
```

Critério de Parada = 1.e-1 (erro máximo de uma casa decimal. O enunciado pede uso de critério de parada maiores, para efetuar menos iterações.)

```
critério = 1.e-1;
```

Sem o uso de fatores de relaxação foram necessários muitas iterações para que o valor tivesse convergência.

```
leonardo@leonardo: ~/Documents/CalcNum
ans =
    1.0000    150.0000    256.2500
ans =
    2.0000    166.6667    82.4653
ans =
    3.0000    208.3333    51.3274
ans =
    4.0000    222.5630    39.3052
ans =
    5.0000    237.0544    32.3652
ans =
    6.0000    247.2724    27.7700
ans =
    7.0000    255.5672    24.3865
ans =
    8.0000    262.3230    21.8085
ans =
    9.0000    267.9041    19.7644
ans =
   10.000    272.572    18.101
-- less -- (f)orward, (b)ack, (q)uit
```

```
leonardo@leonardo: ~/Documents/CalcNum
ans =
    1.4080e+03   -2.8147e+02    1.0185e-01
ans =
    1.4090e+03   -2.8149e+02    1.0162e-01
ans =
    1.4100e+03   -2.8151e+02    1.0139e-01
ans =
    1.4110e+03   -2.8154e+02    1.0117e-01
ans =
    1.4120e+03   -2.8156e+02    1.0094e-01
ans =
    1.4130e+03   -2.8158e+02    1.0071e-01
ans =
    1.4140e+03   -2.8161e+02    1.0049e-01
ans =
    1.4150e+03   -2.8163e+02    1.0026e-01
ans =
    1.4160e+03   -2.8165e+02    1.0004e-01
ans =
    1.4170e+03   -2.8167e+02    9.9810e-02
octave:36>
```

Obs: Primeira coluna representa o número do passo, a segunda coluna representa $x(1)$ e a terceira coluna representa o módulo da diferença entre x e x_i

Para acelerar o processo de convergência é recomendado o uso de fatores de relaxação.

Nossas equações de formação do sistema ficam de tal forma:

```
i=1;
x(i)= (1 - lambda)*x(i) + lambda*(150-x(i+1));
for i=2:n/2
    x(i)=(1 - lambda)*x(i) + lambda*(100-x(i-1)-x(i+1)-x(i+50))/3;
end
for i=(n/2)+1:n-1
    x(i)=(1 - lambda)*x(i) + lambda*(200-x(i-50)-x(i-1)-x(i+1))/3;
end
i=n;
x(i)=((1 - lambda)*x(i) + lambda*(300-x(i-1)));
```

$$\lambda \ (0 < \lambda < 2)$$

Para encontrar o fator(lambda) que nós dá o menor número de equações realizamos testes sucessivos, alterando o valor da variável até encontrarmos o melhor valor.

Lambda	Nº Iterações
0.5	2000+
1.5	660
1.6	540
1.7	423
1.8	308
1.9	815

Portanto, é recomendado o uso de um fator de sobre-relaxação($1 \leq \lambda < 2$) para que seja obtida a convergência com o menor número de iterações.

```
leonardo@leonardo: ~/Documents/CalcNum
ans =
    299.00000   -290.73328    0.11674
ans =
    300.00000   -290.64071    0.11457
ans =
    301.00000   -290.54962    0.11243
ans =
    302.00000   -290.45999    0.11033
ans =
    303.00000   -290.37180    0.10827
ans =
    304.00000   -290.28502    0.10624
ans =
    305.00000   -290.19965    0.10424
ans =
    306.00000   -290.11566    0.10228
ans =
    307.00000   -290.03304    0.10035
ans =
    3.0800e+02   -2.8995e+02    9.8455e-02
octave:43>
```

Iterações com fator lambda = 1.8

- c) Utilizando o método de diagonalização iterativa de Gauss-Seidel com fator de relaxação e critério de parada $\text{Max}|\Delta x_i| \leq 1.10 \cdot 10^{-4}$, a solução é calculada em 638 iterações.
A solução do sistema está na imagem abaixo, a partir da segunda coluna.

```
leonardo@leonardo: ~/Documents/CalcNum
Columns 1 through 11:
    6.3800e+02   -2.8562e+02    4.3562e+02   -4.1137e+02    4.0384e+02   -3.3739e+02    3.1855e+02   -2.4906e+02    2.2941e+02   -1.5971e+02    1.4000e+02
Columns 12 through 22:
   -7.0281e+01    5.0565e+01    1.9152e+01   -3.8869e+01    1.0859e+02   -1.2830e+02    1.9802e+02   -2.1774e+02    2.8745e+02   -3.0717e+02    3.7689e+02
Columns 23 through 33:
   -3.9661e+02    4.6632e+02   -4.8604e+02    5.5576e+02   -5.7547e+02    6.4519e+02   -6.6491e+02    7.3462e+02   -7.5434e+02    8.2406e+02   -8.4377e+02
Columns 34 through 44:
    9.1349e+02   -9.3321e+02    1.0029e+03   -1.0226e+03    1.0924e+03   -1.1121e+03    1.1818e+03   -1.2015e+03    1.2712e+03   -1.2909e+03    1.3606e+03
Columns 45 through 55:
   -1.3803e+03    1.4498e+03   -1.4685e+03    1.5348e+03   -1.5416e+03    1.5630e+03   -1.4024e+03    7.9929e+02   -5.0986e+02    4.9466e+02   -3.6276e+02
Columns 56 through 66:
    3.8978e+02   -2.6919e+02    2.9924e+02   -1.7946e+02    2.0972e+02   -9.0001e+01    1.2028e+02   -5.6494e-01    3.0848e+01    8.8869e+01   -5.8586e+01
Columns 67 through 77:
    1.7830e+02   -1.4802e+02    2.6774e+02   -2.3745e+02    3.5717e+02   -3.2689e+02    4.4661e+02   -4.1632e+02    5.3604e+02   -5.0576e+02    6.2547e+02
Columns 78 through 88:
   -5.9519e+02    7.1491e+02   -6.8462e+02    8.0434e+02   -7.7406e+02    8.9377e+02   -8.6349e+02    9.8321e+02   -9.5292e+02    1.0726e+03   -1.0424e+03
Columns 89 through 99:
    1.1621e+03   -1.1318e+03    1.2515e+03   -1.2212e+03    1.3409e+03   -1.3107e+03    1.4305e+03   -1.4004e+03    1.5211e+03   -1.4943e+03    1.6269e+03
Columns 100 and 101:
   -1.6449e+03    1.9449e+03
octave:45>
```

Para calcular o número de operações em ponto flutuante devemos colocar um contador que acrescente, em cada iteração, a quantidade de operações com ponto flutuante foram realizadas. Esse contador pode ser visto no arquivo “**SistemaC.m**”.

Após todas as iterações temos que realizamos: **570372** operações em ponto flutuante.

```
op = 570372
```

d) O erro **relativo** de Arredondamento obtido foi: **0.023412**

Para chegar nesse resultado foi usado um programa escrito em Java (**SistemaD.java**). A função do programa é realizar as iterações para a variável do tipo float e para o tipo double.

Depois basta fazer a operação:

$$\text{Erro Relativo \%} = \left| \frac{VA - VE}{VE} \right| \cdot 100\%$$

e) O erro **absoluto** de Truncamento máximo na solução S obtida acima, em variável ‘double’, é **0.00543368057810767**, como observado na imagem abaixo:

```
erroTruncamento = 0.00543368057810767
```

Esse erro foi calculado utilizando o mesmo algoritmo da questão c) . Primeiramente com $\text{Max}|\Delta x_i| \leq 1.10^{-4}$, armazenando o valor de x para esse caso.

```
passo = 0;
limite = 2000;
dif = 1;
for k=1:100
    xi(k) = 0;
end
x = xi;
n=100;
criterio = 1.e-4;
lambda = 1.8;

while (passo < limite && dif > criterio )

    passo++;

    i=1;
    x(i)= (1 - lambda)*xi(i) + lambda*(150-x(i+1));
    for i=2:n/2
        x(i)=(1 - lambda)*xi(i) + lambda*(100-x(i-1)-x(i+1)-x(i+50))/3;
    end
    for i=(n/2)+1:n-1
        x(i)=(1 - lambda)*xi(i) + lambda*(200-x(i-50)-x(i-1)-x(i+1))/3;
    end
    i=n;
    x(i)=((1 - lambda)*xi(i) + lambda*(300-x(i-1)));

    dif = max(abs(x - xi));

    [passo, x(1)]

    xi = x;

end
```

Depois utilizamos o mesmo algoritmo mas com $\text{Max}|\Delta x_i| \leq 1.10^{-8}$ e armazenamos o valor de x_n .

```
limite = 2000;
criterio = 1.e-8;
dif = 1;
passo = 0;
xn = x;

while (passo < limite && dif > criterio )

    passo++;

    i=1;
    xn(i)= (1 - lambda)*xi(i) + lambda*(150-xn(i+1));
    for i=2:n/2
        xn(i)=(1 - lambda)*xi(i) + lambda*(100-xn(i-1)-xn(i+1)-xn(i+50))/3;
    end
    for i=(n/2)+1:n-1
        xn(i)=(1 - lambda)*xi(i) + lambda*(200-xn(i-50)-xn(i-1)-xn(i+1))/3;
    end
    i=n;
    xn(i)=((1 - lambda)*xi(i) + lambda*(300-xn(i-1)));

    dif = max(abs(xn - xi));

    [passo, xn(1)]

    xi = xn;

end
```

Por fim, realizamos a operação para calcular o erro de truncamento propriamente dito:

```
erroTruncamento = max(abs(x - xn))
```

Os nomes dos programas correspondem à questão em que eles são utilizados:

- b) - SistemaB
- c) – SistemaC
- d) – SistemaD
- e) - SistemaE