Exercício de Programa 2

Caio Vnícius Dadauto 7994808

23 de abril de 2013

Item A

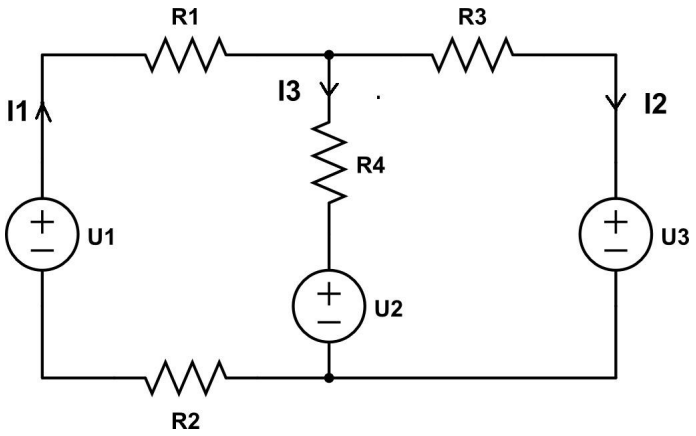

Figura 1: Circuito A.

Aplicando Kirchhoff ao circuito A se obtem as seguintes relações.

$$U_1 - I_1R_1 - I_3R_4 - U_2 - I_1R_2 = 0 \quad (1)$$

$$U_3 + I_2R_3 - I_3R_4 - U_4 = 0 \quad (2)$$

$$I_1 - I_2 - I_3 = 0 \quad (3)$$

Onde $R_1 = 5\Omega$, $R_2 = 7\Omega$, $R_3 = 5\Omega$, $R_4 = 2\Omega$, $U_1 = 24V$, $U_2 = 24V$ e $U_3 = 24V$.

A partir das relações apresentadas acima, tem-se o seguinte sistema:

$$\begin{cases} 12I_1 - 2I_3 &= 15 \\ 5I_2 - 2I_3 &= 3 \\ I_1 - I_2 - I_3 &= 0 \end{cases} \tag{4}$$

que na forma matricial é dado por:

$$\begin{bmatrix} 12 & 0 & 2 \\ 0 & 5 & 2 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 15 \\ 3 \\ 0 \end{bmatrix}$$

Item B

Tendo em vista a solução do sistema apresentado no item A e, ainda, a solução de qualquer sistema de n equações e n variaveis, é possível implementar um programa para determinar tal solução. Programa que utiliza do método de eliminação de Gauss com pivotação parcial. O programa trabalha com $n < 100$ e matrizes que nao possuem fileiras nulas (colunas ou linhas).

Programa 1: Implementação para solução de sistemas pelo método de Gauss.

```
1  #include <stdio.h>
2  #define MAX 100
3
4  void resolucao_uper (double matriz[][MAX], int n, double *solucao) {
5  ···int
6  ·····i,
7  ·····j;
8
9  ···for (i = 0; i < n; ++ i)
10 ·····solucao[i] = matriz[i][n];
11 ···for (i = n - 1; i >= 0; -- i) {
12 ·····for (j = n - 1; j >= i; -- j) {
13 ·····if (i != j)
14 ·····solucao[i] -= matriz[i][j] * solucao[j];
15 ·····else
16 ·····solucao[i] /= matriz[i][j];
17 ·····}
18 ···}
19 }
20
```

```
21 void eliminacao (double matriz[][MAX], int n, int ep) {
22 ···int
23 ·····i,
24 ·····j;
25 ···double
26 ·····elimina,
27 ·····pivo;
28
29 ···pivo = matriz[ep][ep];
30 ···for (i = ep + 1; i < n; ++ i) {
31 ·····elimina = matriz[i][ep];
32 ·····for (j = ep; j <= n; ++ j)
33 ·····matriz[i][j] = matriz[i][j] -
34 ·····(matriz[ep][j] * elimina / pivo);
35 ···}
36 }
37
38 void imprime (FILE *entrada, double matriz[][MAX], int n) {
39 ···int
40 ·····i,
41 ·····j;
42
43 ···for (i = 0; i < n; ++ i) {
44 ·····for (j = 0; j <= n; ++ j) {
45 ·····if (j == n)
46 ·····fprintf (entrada, "%.5f\\\\\n", matriz[i][j]);
47 ·····else
48 ·····fprintf (entrada, "%.5f\t&\t", matriz[i][j]);
49 ·····}
50 ···}
51 ···fprintf (entrada, "\n");
52 }
53
54 void pivotamento (int etapa, double matriz[][MAX], int n) {
55 ···double
56 ·····aux [MAX],
57 ·····max;
58 ···int
59 ·····i,
60 ·····j,
61 ·····linha;
62
63 ···max = matriz[etapa][etapa];
64 ···linha = etapa;
65 ···for (i = etapa + 1; i < n; ++ i) {
66 ·····if (max < matriz[i][etapa]) {
67 ·····max = matriz[i][etapa];
```

```
68 ·········linha  = i;
69 ·····}
70 ···}
71 ···for (j = 0; (j <= n) && (linha != etapa); ++ j) {
72 ·····aux[j] = matriz[etapa][j];
73 ·····matriz[etapa][j] = matriz[linha][j];
74 ·····matriz[linha][j] = aux[j];
75 ···}
76 }
77
78
79 int main () {
80 ···int
81 ·····etapa,
82 ·····i,
83 ·····j,
84 ·····n;
85 ···double
86 ·····solucao[MAX],
87 ·····matriz [MAX][MAX];
88 ···FILE
89 ·····*arquivo;
90
91 ···arquivo = fopen ("saida.tex", "w");
92 ···printf ("Entre com o únmero n de equacoes do \
93 ···sistema (n < 100): ");
94 ···scanf ("%d", &n);
95 ···for (i = 0; i < n; ++ i) {
96 ·····for (j = 0; j <= n; ++ j) {
97 ·········printf ("Entre com o valor a(%d, %d) da matriz \
98 ·········expandida do sistema: ", i + 1, j + 1);
99 ·········scanf ("%lf", &matriz[i][j]);
100 ·····}
101 ···}
102
103 ···for (etapa = 0; etapa < n - 1; ++ etapa) {
104 ···pivotamento (etapa, matriz, n);
105 ···imprime (arquivo, matriz, n);
106 ···eliminacao (matriz, n, etapa);
107 ···}
108 ···imprime (arquivo, matriz, n);
109 ···resolucao_uper (matriz, n, &solucao[0]);
110 ···for (i = 0; i < n; ++ i)
111 ·····fprintf (arquivo, "I_%d = %.5f\n", i + 1, solucao[i]);
112 ···fclose (arquivo);
113 ···return 0;
114 }
```

Inserindo como entrada do programa acima a matriz expandida do sistema (4), obtem-se o seguinte conjunto de matrizes para cada etapa de eleminação;

Etapa 1:

$$\begin{bmatrix} 12.00000 & 0.00000 & 2.00000 & 15.00000 \\ 0.00000 & 5.00000 & -2.00000 & 3.00000 \\ 1.00000 & -1.00000 & -1.00000 & 0.00000 \end{bmatrix}$$

Etapa 2:

$$\begin{bmatrix} 12.00000 & 0.00000 & 2.00000 & 15.00000 \\ 0.00000 & 5.00000 & -2.00000 & 3.00000 \\ 0.00000 & -1.00000 & -1.16667 & -1.25000 \end{bmatrix}$$

Etapa 3:

$$\begin{bmatrix} 12.00000 & 0.00000 & 2.00000 & 15.00000 \\ 0.00000 & 5.00000 & -2.00000 & 3.00000 \\ 0.00000 & 0.00000 & -1.56667 & -0.65000 \end{bmatrix}$$

obtendo como solução:

$$I_1 = 1.18085A \quad I_2 = 0.76596A \quad I_3 = 0.41489A$$

Item C

Tendo em vista solucionar o mesmo sistema do item A, porém agora utilizando do método de Jacobi, é necessário que a matriz dos coeficientes do sistema (4) satisfaça o critério das linhas, dado por:

$$\sum_{j=1}^{n} \frac{|a_{kj}|}{|a_{kk}|} < 1, \quad \text{j} \neq \text{k} \tag{5}$$

onde a é o elemento de linha k e coluna j da matriiz dos coeficientes. Para isso as duas primeiras linhas do sistema (4) precisam ser trocadas, antes de

ser aplicado o método de Jacobi. Obtendo a seguinte matriz expandida:

$$\begin{bmatrix} 12.000 & 0.000 & 2.000 & 15.000 \\ 0.000 & 5.000 & -2.000 & 3.000 \\ 1.000 & -1.000 & -1.000 & 0.000 \end{bmatrix} \tag{6}$$

Segue abaixo o programa que implementa o método de Jacobi.

Programa 2: Implementação para o método de Jacobi.

```
#include <stdio.h>
# define EPS 1E-4

void imprime (FILE *entrada, double *solucao, double *solucao_k, int k);
void interacao (double matriz[][4], double *solucao, double *solucao_k);
double delta_d (double *solucao, double *solucao_k);
void copia_vetor (double *a, double *b);
double modulo (double a, double b);

void imprime (FILE *entrada, double *solucao, double *solucao_k, int k) {
   int
      i;

   fprintf (entrada, "%d", k);
   for (i = 0; i < 3; ++ i)
      fprintf (entrada, " & %.5f", solucao_k[i]);
   for (i = 0; i < 3; ++ i)
      fprintf (entrada, " & %.5f ", modulo (solucao[i], solucao_k[i]));
   fprintf (entrada, "\\\\\n\\midrule\n");
}

void interacao (double matriz[][4], double *solucao, double *solucao_k) {
   int
      j,
      i;

   for (i = 0; i < 3; ++ i) {
      solucao_k[i] = matriz[i][3] / matriz[i][i];
      for (j = 0; j < 3; ++ j) {
         if (i != j)
            solucao_k[i] -= ((solucao[j] * matriz[i][j]) / matriz[i][i]);
      }
   }
}

void copia_vetor (double *a, double *b) {
   int
```

```
38 ·····i;
39
40 ···for (i = 0; i < 3; ++ i)
41 ·····a[i] = b[i];
42 }
43
44 double modulo (double a, double b) {
45 ···if (a - b < 0)
46 ·····return b - a;
47 ···return a - b;
48 }
49
50 double delta_d (double *solucao, double *solucao_k) {
51 ···int
52 ·····i;
53 ···double
54 ·····max = 0;
55
56 ···for (i = 0; i < 3; ++ i) {
57 ·····if (max < modulo (solucao[i], solucao_k[i]))
58 ·····max = modulo (solucao[i], solucao_k[i]);
59 ···}
60 ···return max;
61 }
62
63 int main () {
64 ···FILE
65 ·····*arquivo;
66 ···int
67 ·····k = 0;
68 ···double
69 ·····solucao[3] = {0, 0, 0},
70 ·····solucao_k[3] = {1, 1, 1},
71 ·····matriz[3][4] = {{12, 0, 2, 15}, {0, 5, -2, 3}, {1, -1, -1, 0}};
72
73 ···arquivo = fopen ("saida1.tex", "w");
74 ···imprime (arquivo, solucao, solucao_k, k);
75 ···while (delta_d (solucao, solucao_k) >= EPS) {
76 ·····copia_vetor (solucao, solucao_k);
77 ·····++ k;
78 ·····interacao (matriz, solucao, solucao_k);
79 ·····imprime (arquivo, solucao, solucao_k, k);
80 ···}
81 ···fclose (arquivo);
82 ···return 0;
83 }
```

É apresentado na tabela 1 os valores das soluções aproximadas geradas a cada interação pelo programa 2 para a matriz (6).

Interação (k)	$I_1^{(k)}$	$I_2^{(k)}$	$I_3^{(k)}$	$erro_1^{(k)}$	$erro_2^{(k)}$	$erro_3^{(k)}$
0	1.00000	1.00000	1.00000	$\cdots$	$\cdots$	$\cdots$
1	1.08333	1.00000	0.00000	0.08333	0.00000	1.00000
2	1.25000	0.60000	0.08333	0.16667	0.40000	0.08333
3	1.23611	0.63333	0.65000	0.01389	0.03333	0.56667
4	1.14167	0.86000	0.60278	0.09444	0.22667	0.04722
5	1.14954	0.84111	0.28167	0.00787	0.01889	0.32111
6	1.20306	0.71267	0.30843	0.05352	0.12844	0.02676
7	1.19860	0.72337	0.49039	0.00446	0.01070	0.18196
8	1.16827	0.79616	0.47523	0.03033	0.07279	0.01516
9	1.17080	0.79009	0.37211	0.00253	0.00607	0.10311
10	1.18798	0.74885	0.38071	0.01719	0.04124	0.00859
11	1.18655	0.75228	0.43914	0.00143	0.00344	0.05843
12	1.17681	0.77565	0.43427	0.00974	0.02337	0.00487
13	1.17762	0.77371	0.40116	0.00081	0.00195	0.03311
14	1.18314	0.76046	0.40392	0.00552	0.01324	0.00276
15	1.18268	0.76157	0.42268	0.00046	0.00110	0.01876
16	1.17955	0.76907	0.42111	0.00313	0.00751	0.00156
17	1.17981	0.76845	0.41048	0.00026	0.00063	0.01063
18	1.18159	0.76419	0.41137	0.00177	0.00425	0.00089
19	1.18144	0.76455	0.41739	0.00015	0.00035	0.00602
20	1.18043	0.76696	0.41689	0.00100	0.00241	0.00050
21	1.18052	0.76676	0.41348	0.00008	0.00020	0.00341

Continuação na proxima página.

Continuação da página anterior.						
22	1.18109	0.76539	0.41376	0.00057	0.00137	0.00028
23	1.18104	0.76550	0.41570	0.00005	0.00011	0.00193
24	1.18072	0.76628	0.41554	0.00032	0.00077	0.00016
25	1.18074	0.76621	0.41444	0.00003	0.00006	0.00110
26	1.18093	0.76578	0.41453	0.00018	0.00044	0.00009
27	1.18091	0.76581	0.41515	0.00002	0.00004	0.00062
28	1.18081	0.76606	0.41510	0.00010	0.00025	0.00005
29	1.18082	0.76604	0.41475	0.00001	0.00002	0.00035
30	1.18088	0.76590	0.41478	0.00006	0.00014	0.00003
31	1.18087	0.76591	0.41498	0.00000	0.00001	0.00020
32	1.18084	0.76599	0.41496	0.00003	0.00008	0.00002
Solução aproximada						
$I_1 = 1.18084A \quad I_2 = 0.76599A \quad I_3 = 0.41496A$						

Tabela 1: Resultados obtidos pelo método de Jacobi.

Item C

Partindo do mesmo sistema (4), mas agoro utilizando o método de Gauss-Seidel para a solução do mesmo, há a necessidade de satisfazer o critério de Sassenfeld que é dado por:

$$\beta = \max_{1 \leq i \leq n} \{\beta_i\} < 1 \tag{7}$$

onde,

$$\beta_i = \frac{\sum_{j=1}^{i-1} \beta_j |a_{ij}| + \sum_{j=i+1}^{n} |a_{ij}|}{|a_{ii}|}$$

sendo que a matriz expandiada (6) satisfaz tal critério.

Segue o código para solucionar a matriz (6) pelo métododе Gauss-Seidel.

Programa 3: Implementação para o método de Jacobi.

```
1 #include <stdio.h>
2 # define EPS 1E-4
3
```

```
4  void imprime (FILE *entrada, double *solucao, double *solucao_k, int k);
5  void interacao (double matriz[][4], double *solucao, double *solucao_k);
6  double delta_d (double *solucao, double *solucao_k);
7  void copia_vetor (double *a, double *b);
8  double modulo (double a, double b);
9
10 void imprime (FILE *entrada, double *solucao, double *solucao_k, int k) {
11 ···int
12 ·····i;
13
14 ···fprintf (entrada, "%d", k);
15 ···for (i = 0; i < 3; ++ i)
16 ·····fprintf (entrada, " & %.5f", solucao_k[i]);
17 ···for (i = 0; i < 3; ++ i)
18 ·····fprintf (entrada, " & %.5f ", modulo (solucao[i], solucao_k[i]));
19 ···fprintf (entrada, "\\\\\n\\midrule\n");
20 }
21
22 void interacao (double matriz[][4], double *solucao, double *solucao_k) {
23 ···int
24 ·····j,
25 ·····i;
26 ···double
27 ·····aux[3];
28
29 ···copia_vetor (aux, solucao);
30 ···for (i = 0; i < 3; ++ i) {
31 ·····solucao_k[i] = matriz[i][3] / matriz[i][i];
32 ·····for (j = 0; j < 3; ++ j) {
33 ·····if (i != j)
34 ·····solucao_k[i] -= ((aux[j] * matriz[i][j]) / matriz[i][i]);
35 ·····}
36 ···aux[i] = solucao_k[i];
37 ···}
38 }
39
40 void copia_vetor (double *a, double *b) {
41 ···int
42 ·····i;
43
44 ···for (i = 0; i < 3; ++ i)
45 ·····a[i] = b[i];
46 }
47
48 double modulo (double a, double b) {
49 ···if (a - b < 0)
50 ·····return b - a;
```

```
51 ···return a - b;
52 }
53
54 double delta_d (double *solucao, double *solucao_k) {
55 ···int
56 ·····i;
57 ···double
58 ·····max = 0;
59
60 ···for (i = 0; i < 3; ++ i) {
61 ·····if (max < modulo (solucao[i], solucao_k[i]))
62 ·····max = modulo (solucao[i], solucao_k[i]);
63 ···}
64 ···return max;
65 }
66
67 int main () {
68 ···FILE
69 ·····*arquivo;
70 ···int
71 ·····k = 0;
72 ···double
73 ·····solucao[3] = {0, 0, 0},
74 ·····solucao_k[3] = {1, 1, 1},
75 ·····matriz[3][4] = {{12, 0, 2, 15}, {0, 5, -2, 3}, {1, -1, -1, 0}};
76
77 ···arquivo = fopen ("saida1.tex", "w");
78 ···imprime (arquivo, solucao, solucao_k, k);
79 ···while (delta_d (solucao, solucao_k) >= EPS) {
80 ·····copia_vetor (solucao, solucao_k);
81 ·····++ k;
82 ·····interacao (matriz, solucao, solucao_k);
83 ·····imprime (arquivo, solucao, solucao_k, k);
84 ···}
85 ···fclose (arquivo);
86 ···return 0;
87 }
```

É apresentado na tabela 2 os valores das soluções aproximadas geradas a cada interação pelo programa 3 para a matriz (6).

Interação (k)	$I_1^{(k)}$	$I_2^{(k)}$	$I_3^{(k)}$	$erro_1^{(k)}$	$erro_2^{(k)}$	$erro_3^{(k)}$
0	1.00000	1.00000	1.00000	$\cdots$	$\cdots$	$\cdots$
1	1.08333	1.00000	0.08333	0.08333	0.00000	0.91667
2	1.23611	0.63333	0.60278	0.15278	0.36667	0.51944
3	1.14954	0.84111	0.30843	0.08657	0.20778	0.29435
4	1.19860	0.72337	0.47523	0.04906	0.11774	0.16680
5	1.17080	0.79009	0.38071	0.02780	0.06672	0.09452
6	1.18655	0.75228	0.43427	0.01575	0.03781	0.05356
7	1.17762	0.77371	0.40392	0.00893	0.02142	0.03035
8	1.18268	0.76157	0.42111	0.00506	0.01214	0.01720
9	1.17981	0.76845	0.41137	0.00287	0.00688	0.00975
10	1.18144	0.76455	0.41689	0.00162	0.00390	0.00552
11	1.18052	0.76676	0.41376	0.00092	0.00221	0.00313
12	1.18104	0.76550	0.41554	0.00052	0.00125	0.00177
13	1.18074	0.76621	0.41453	0.00030	0.00071	0.00100
14	1.18091	0.76581	0.41510	0.00017	0.00040	0.00057
15	1.18082	0.76604	0.41478	0.00009	0.00023	0.00032
16	1.18087	0.76591	0.41496	0.00005	0.00013	0.00018
17	1.18084	0.76598	0.41486	0.00003	0.00007	0.00010
18	1.18086	0.76594	0.41491	0.00002	0.00004	0.00006
Solução aproximada						
$I_1 = 1.18086A \quad I_2 = 0.76594A \quad I_3 = 0.41491A$						

Tabela 2: Resultados obtidos pelo método de Gauss-Seidel.