

Relatório da atividade avaliativa 1 de Cálculo Numérico

Samira Haddad RA: 11201812350

Gabrieli Parra Silva RA: 11201721386

Juliane dos Santos Assis RA: 11201810271

Liandra Cardoso da Silva RA: 11064916

Samara Suellen Miranda de Azevedo RA: 11201820807

April 5, 2021

Exercício 1

(2.0) Calcule $f_{T_n}(\lambda)$ para $\lambda = 1, 2, 3$ e 4 , para $n = 7$. Para $n = 0$, exiba a matriz V e a matriz V' obtida após a aplicação do processo de eliminação à matriz V , com todas as casas decimais disponíveis.

Antes de calcular o valor de $f_{T_n}(\lambda)$ temos que criar nossa matriz T_n , os valores dessa matriz são definidos pela seguinte forma:

$$a_{x,y} = \begin{cases} \lambda, & \text{se } x = y \\ -1, & \text{se } x = y + 1 \text{ ou } x + 1 = y \\ 0, & \text{caso contrário} \end{cases}$$

De forma que a matriz T_n terá a seguinte forma:

$$\begin{bmatrix} \lambda & -1 & 0 & \cdots & 0 \\ -1 & \lambda & -1 & \cdots & 0 \\ 0 & -1 & \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda \end{bmatrix}$$

O algoritmo usado para criar tal matriz foi o seguinte:

```
1 public static Double[][] TGn(int lambda, int n){
2     Double A[][] = new Double[n][n];
3
4     //preenchendo a matriz com zeros
5     for(int linha = 0; linha < n; linha++){
6         for(int coluna = 0; coluna < n; coluna++){
7             A[linha][coluna] = 0d;
8         }
9     }
```

```

9     }
10    //transformando a matriz em Tn
11    for(int i = 0; i < n; i++) {
12        if(i < n-1){
13            A[i][i+1] = -1d;
14            A[i+1][i] = -1d;
15        }
16
17        A[i][i] = Double.valueOf(lambda);
18
19    }
20    return A;
21 }

```

Listing 1: Método TGn

A função $f_{Tn}(\lambda)$ tem o intuito de fazer o escalonamento da matriz para posteriormente calcular a determinante da mesma. O algoritmo usado para calcular $f_{TGn}(\lambda)$ foi o seguinte:

```

1 public static double fTGn(double lambda, int n, boolean debug){
2
3     boolean troca = true;
4     double m = 0;
5     double temp;
6     double determinante = 1;
7     Double [][] A = TGn(lambda, n);
8     Double [][] A_original = TGn(lambda, n);
9     int sgn = 1;
10    int l = 0;
11    int[] t = new int[2];
12    boolean erro = false;
13
14    ArrayList<int[]> changeLine = new ArrayList<int[]>();
15
16    for (int j = 0; j < n; j++){
17        troca = true;
18        l = j;
19        while(l < n){
20            if(A[l][j] == 0 && l != n - 1){
21                l = l + 1;
22                sgn = -sgn;
23
24            }else if(A[l][j] == 0 && l == n - 1){
25                l = l + 1;
26                sgn = -sgn;
27                troca = false;
28
29            }else if(A[l][j] != 0){
30                break;
31            }
32        }
33        if(troca == false && A[j][j] == 0){
34            // System.err.println("Erro: sistema singular");
35            erro = true;
36            break;
37        }
38        else if(troca == true && A[j][j] == 0){
39            for(int k = j; k < n; k++){
40                temp = A[j][k];

```

```

41         A[j][k] = A[l][k];
42         A[l][k] = temp;
43     }
44     t[0] = j + 1;
45     t[1] = l + 1;
46     changeLine.add(t.clone());
47 }
48 for (int i = j+1; i < n; i++){
49     m = - A[i][j]/A[j][j];
50     for(int k = j; k < n; k++){
51         A[i][k] = A[i][k] + m*A[j][k];
52     }
53 }
54 }
55 for(int i = 0; i < n; i++){
56     determinante = determinante*A[i][i];
57 }
58
59 if(erro == false){
60     determinante = determinante*sgn;
61 }
62 else{
63     determinante = 0;
64 }
65
66 if(lambda == 0 && debug){
67     System.out.println("V: ");
68     printArray(A_original);
69     System.out.println("V': ");
70     printArray(A);
71
72     if(changeLine.size() == 0){
73         System.out.println("trocaLinhas: Nao houveram trocas");
74     }else{
75         System.out.println("trocaLinhas:");
76         changeLine.forEach((d) -> printArray(d));
77     }
78 }
79 return determinante;
80 }

```

Listing 2: Método f_{TGN}

Vale ressaltar que os métodos `printArray(double A[][])` e `printArray(int A[])` usados dentro de $f_{Tn}(\lambda)$, são os seguintes:

```

1 public static void printArray(Double A[][])
2 {
3     int n = A.length;
4     for(int linha = 0; linha < n; linha = linha + 1)
5     {
6         for(int coluna = 0; coluna < n; coluna = coluna + 1)
7         {
8             System.out.print "[" + A[linha][coluna] + " ";
9         }
10        System.out.print "\n";
11    }
12
13 }

```

```

14 public static void printArray(int A[])
15 {
16     int n = A.length;
17
18     for(int i = 0; i < n; i = i + 1)
19     {
20         System.out.print "[" + A[i] + " ");
21     }
22     System.out.print("\n");
23
24 }

```

Listing 3: Método printArray

Por fim, para calcular $f_{Tn}(\lambda)$ com $\lambda = 0, 1, 2, 3$ e 4 e $n = 7$ temos que chamar o seguinte método dentro do método main, nesse método aproveitamos também para exibir o resultado de $f_{Tn}(\lambda)$

```

1 public static void exerc1(){
2
3     int x = 4; // valor maximo que lambda pode valer
4     int n = 6; //tamanho da matriz
5     double det = 0;
6     boolean debug = true; //variavel que permite a impressao da saida
7
8     for(int lambda = 0; lambda <= x; lambda++){
9
10         det = fTGn(lambda, n, debug);
11         System.out.println("fT"+n+"("+ lambda +") vale " + det);
12
13     }
14
15 }

```

Listing 4: Método exerc1()

Tendo a seguinte resposta:

```

1 V:
2 [0.0][-1.0][0.0][0.0][0.0][0.0][0.0]
3 [-1.0][0.0][-1.0][0.0][0.0][0.0][0.0]
4 [0.0][-1.0][0.0][-1.0][0.0][0.0][0.0]
5 [0.0][0.0][-1.0][0.0][-1.0][0.0][0.0]
6 [0.0][0.0][0.0][-1.0][0.0][-1.0][0.0]
7 [0.0][0.0][0.0][0.0][-1.0][0.0][-1.0]
8 [0.0][0.0][0.0][0.0][0.0][-1.0][0.0]
9 V':
10 [-1.0][0.0][-1.0][0.0][0.0][0.0][0.0]
11 [0.0][-1.0][0.0][0.0][0.0][0.0][0.0]
12 [0.0][0.0][-1.0][0.0][-1.0][0.0][0.0]
13 [0.0][0.0][0.0][-1.0][0.0][0.0][0.0]
14 [0.0][0.0][0.0][0.0][-1.0][0.0][-1.0]
15 [0.0][0.0][0.0][0.0][0.0][-1.0][0.0]
16 [0.0][0.0][0.0][0.0][0.0][0.0][0.0]
17 trocaLinhas:
18 [1][2]
19 [3][4]
20 [5][6]
21 fT7(0) vale 0.0
22 fT7(1) vale 1.0

```

```

23 fT7(2) vale 7.999999999999998
24 fT7(3) vale 987.00000000000001
25 fT7(4) vale 10864.0

```

Listing 5: Saída do exercício 1

Exercício 2

Exercício 3

Exercício 4

Exercício 5

Exercício 6

Repita os cálculos acima, porém, dessa vez, em cada intervalo $[x_j, x_{j+1}]$, utilize o ponto médio $\frac{x_j + x_{j+1}}{2}$ como aproximação inicial para o método de Newton. Pode-se provar que as raízes exatas de f_{T11} são

$$\lambda_k = 2 \cdot \cos\left(\frac{k\pi}{12}\right), \quad k = 1, 2, \dots, 11$$

Calcule os erros

$$|\lambda_k - \tilde{\lambda}_k|, \quad k = 1, 2, \dots, 11$$

entre as raízes exatas λ_k e as raízes aproximadas $\tilde{\lambda}_k$ calculadas pelo método de Newton e verifique se a precisão $\epsilon = 10^{12}$ foi atingida ou não

Para resolver esse exercício precisamos primeiro definir quais serão nossos intervalos, cada intervalo irá conter uma raiz, logo serão 11 intervalos para $n = 11$. Para definir os intervalos que seriam usados entre $[-2, 2]$ usamos como base o exercício 2 e chegamos no seguinte: $[-2.0, -1.8787879]$, $[-1.7575758, -1.6363636]$, $[-1.5151515, -1.3939394]$, $[-1.030303, -0.9090909]$, $[-0.5454545, -0.42424238]$, $[-0.060606003, 0.060606003]$, $[0.4242425, 0.5454545]$, $[0.909091, 1.030303]$, $[1.3939395, 1.5151515]$, $[1.6363637, 1.757576]$, $[1.878788, 2.0]$

Agora que já sabemos nossos intervalos de interesse podemos começar a fazer o exercício. O primeiro passo é criar o método que calculará a derivada da nossa função, que será posteriormente utilizada no método de Newton.

```

1 public static double fLinha(double lambda, int n){
2
3     double derivada = 1;
4     double z;
5
6     if (lambda == 2){
7
8         derivada = (Math.pow((n + 1), 5) - Math.pow((n+1), 3))/3;
9
10    }else if(lambda == -2){
11
12        derivada = Math.pow(-1, n+1)*((Math.pow((n + 1), 5) - Math.pow((n+1), 3))/3);

```

```

13
14     }else{
15         z = lambda/2;
16         derivada = ((n + 1)*Math.cos((n+1)*Math.acos(z)) - z*(Math.sin((n+1)
17 *Math.acos(z)))/(Math.sin(Math.acos(z)))/(2*(Math.pow(z,2) - 1));
18     }
19     return derivada;
20 }

```

Listing 6: Método fLinha

O próximo passo é calcular o método de Newton que será responsável por calcular a aproximação das raízes de $f_{Tn}(\lambda)$ dentro de cada intervalo. Seu código é o seguinte:

```

1 public static double newton(double a, double b, double prec, int n_max, double
  x0, int n){
2
3     double alpha = x0;
4     int i = 0;
5
6     while(
7         fTn(alpha - prec, n, false)*fTn(alpha + prec, n, false) > 0 &&
8         i <= n_max
9     ){
10
11         i = i + 1;
12         if(a <= alpha && b >= alpha){
13             alpha = alpha - (fTn(alpha, n, false)/fLinha(alpha, n));
14         }
15
16     }
17
18     return alpha;
19 }

```

Listing 7: Método de Newton

Além disso, como o enunciado indica, devemos realizar os mesmos cálculos feitos no exercício anterior, logo devemos usar o método de bissecção para conseguir outras aproximações para as raízes da função $f_{Tn}(\lambda)$. Para tal usaremos seguinte método:

```

1 public static double bisseccao(double m, double M, int n_max, double prec, int n
  ){
2
3     double alpha = 0.5*(m+M); // x que sera testado
4     int count = 0; // numero de interacoes
5     boolean debug = false;
6
7     while((fTn(alpha-prec, n, debug)*fTn(alpha+prec, n, debug)) > 0
8     && count < n_max){
9
10         count = count + 1;
11         alpha = 0.5*(m+M);
12
13         // verifica se a raiz esta no intervalo [m, alpha]
14         if(fTn(alpha, n, debug)*fTn(m, n, debug) < 0
15         || fTn(alpha, n, debug)*fTn(m, n, debug) == 0 ){
16             M = alpha;

```

```

17     }
18     // verifica se a raiz esta no intervalo [alpha, M]
19     if(fTGn(alpha, n, debug)*fTGn(m, n, debug) > 0){
20         m = alpha;
21     }
22 }
23 return alpha;
24 }

```

Listing 8: Método de Bissecção

Por fim, devemos chamar o seguinte método dentro do main() para executar nosso exercício. Nosso intuito com o método exerc6() é de encontrar a aproximação para as 11 raízes da função pelos métodos de Bissecção e Newton da mesma forma que iremos compará-los com o valor real das respectivas raízes.

```

1 public static void exerc6(){
2
3     Double[][] intervalos = {
4         {-2.0, -1.8787879},
5         {-1.7575758, -1.6363636},
6         {-1.5151515, -1.3939394},
7         {-1.030303, -0.9090909},
8         {-0.5454545, -0.42424238},
9         {-0.060606003, 0.060606003},
10        {0.4242425, 0.5454545},
11        {0.909091, 1.030303},
12        {1.3939395, 1.5151515},
13        {1.6363637, 1.757576},
14        {1.878788, 2.0}
15    };
16    int tamanho_intervalo = intervalos.length;
17    double precisao = Math.pow(10,-12);
18    double menor = 0, maior = 0, x0 = 0;
19    double raiz_exata = 0;
20    double raiz_aprox_newton = 0;
21    double raiz_aprox_bissec = 0;
22    int n_max = 50;
23    int n = 11;
24    int k = 11;
25
26    for(int j = 0; j < tamanho_intervalo; j++){
27
28        if(j <= tamanho_intervalo - 1){
29
30            menor = intervalos[j][0];
31            maior = intervalos[j][1];
32            x0 = (maior+menor)/2;
33
34            raiz_exata = 2*Math.cos(k*Math.PI/12);
35            raiz_aprox_newton = newton(menor,maior, precisao, n_max, x0, n);
36            raiz_aprox_bissec = bisseccao(menor,maior,n_max,precisao, n);
37
38            System.out.println(">>> intervalo: [" + menor
39                               + ", "+ maior+"]");
40            System.out.println("---> raiz exata: "
41                               + raiz_exata);
42            System.out.println("---> raiz aprox. (newton): "

```

```

43         + raiz_aprox_newton);
44     System.out.println("---> raiz aprox. (bissec): "
45         + raiz_aprox_bissec);
46     System.out.println("---> erro (newton): "
47         + Math.abs(raiz_exata - raiz_aprox_newton));
48     System.out.println("---> erro (bissec): "
49         + Math.abs(raiz_exata - raiz_aprox_bissec));
50     System.out.println("");
51
52     }
53     k = k - 1;
54
55     }
56
57     }

```

Listing 9: Método exerc6()

Ao executar o método exerc6() teremos a seguinte resposta:

```

1 >>>> intervalo: [-2.0, -1.8787879]
2 ---> raiz exata: -1.9318516525781364
3 ---> raiz aprox. (newton): -1.9318516525781366
4 ---> raiz aprox. (bissec): -1.931851652578306
5 ---> erro (newton): 2.220446049250313E-16
6 ---> erro (bissec): 1.6964207816272392E-13
7
8 >>>> intervalo: [-1.7575758, -1.6363636]
9 ---> raiz exata: -1.7320508075688774
10 ---> raiz aprox. (newton): -1.7320508075688776
11 ---> raiz aprox. (bissec): -1.732050807569201
12 ---> erro (newton): 2.220446049250313E-16
13 ---> erro (bissec): 3.235189893757706E-13
14
15 >>>> intervalo: [-1.5151515, -1.3939394]
16 ---> raiz exata: -1.414213562373095
17 ---> raiz aprox. (newton): -1.4142135623730971
18 ---> raiz aprox. (bissec): -1.4142135623727792
19 ---> erro (newton): 2.220446049250313E-15
20 ---> erro (bissec): 3.157474282033945E-13
21
22 >>>> intervalo: [-1.030303, -0.9090909]
23 ---> raiz exata: -0.9999999999999996
24 ---> raiz aprox. (newton): -1.00000000000000793
25 ---> raiz aprox. (bissec): -0.99999999999993041
26 ---> erro (newton): 7.971401316808624E-14
27 ---> erro (bissec): 6.954437026251981E-13
28
29 >>>> intervalo: [-0.5454545, -0.42424238]
30 ---> raiz exata: -0.5176380902050413
31 ---> raiz aprox. (newton): -0.5176380902050433
32 ---> raiz aprox. (bissec): -0.5176380902057762
33 ---> erro (newton): 1.9984014443252818E-15
34 ---> erro (bissec): 7.349676423018536E-13
35
36 >>>> intervalo: [-0.060606003, 0.060606003]
37 ---> raiz exata: 1.2246467991473532E-16
38 ---> raiz aprox. (newton): 0.0
39 ---> raiz aprox. (bissec): 0.0

```



```

40 ----> erro (newton): 1.2246467991473532E-16
41 ----> erro (bissec): 1.2246467991473532E-16
42
43 >>>> intervalo: [0.4242425, 0.5454545]
44 ----> raiz exata: 0.5176380902050415
45 ----> raiz aprox. (newton): 0.5176380902050433
46 ----> raiz aprox. (bissec): 0.5176380902047707
47 ----> erro (newton): 1.7763568394002505E-15
48 ----> erro (bissec): 2.707833957060757E-13
49
50 >>>> intervalo: [0.909091, 1.030303]
51 ----> raiz exata: 1.0000000000000002
52 ----> raiz aprox. (newton): 1.00000000000000793
53 ----> raiz aprox. (bissec): 1.0
54 ----> erro (newton): 7.904787935331115E-14
55 ----> erro (bissec): 2.220446049250313E-16
56
57 >>>> intervalo: [1.3939395, 1.5151515]
58 ----> raiz exata: 1.4142135623730951
59 ----> raiz aprox. (newton): 1.4142135623730971
60 ----> raiz aprox. (bissec): 1.4142135623726788
61 ----> erro (newton): 1.9984014443252818E-15
62 ----> erro (bissec): 4.163336342344337E-13
63
64 >>>> intervalo: [1.6363637, 1.757576]
65 ----> raiz exata: 1.7320508075688774
66 ----> raiz aprox. (newton): 1.7320508075688779
67 ----> raiz aprox. (bissec): 1.7320508075681347
68 ----> erro (newton): 4.440892098500626E-16
69 ----> erro (bissec): 7.427392034742297E-13
70
71 >>>> intervalo: [1.878788, 2.0]
72 ----> raiz exata: 1.9318516525781366
73 ----> raiz aprox. (newton): 1.9318516525781366
74 ----> raiz aprox. (bissec): 1.93185165257745
75 ----> erro (newton): 0.0
76 ----> erro (bissec): 6.865619184281968E-13

```

Listing 10: Saída do exercício 6()

Ao analisar nossa saída podemos perceber que o resultado decorrente do método de Newton obteve ótimas aproximações para as raízes do método $f_{Tn}(\lambda)$ com precisões que chegam a casa de 10^{-16} e em comparamos com o método da bissecção percebemos que o método Newton obtém resultados mais precisos. A diferença de resultado entre os dois métodos pode ser consequência do fato de que o método de bissecção precisa de um maior número de interações para convergir para uma resolução mais próxima da real se comparado com o método de Newton e no exemplo ambos foram executados n_{max} vezes.