Atividade Avaliativa 1

Integrantes

Samira Haddad RA: 11201812350
Gabrieli Parra Silva RA: 11201721386
Juliane dos Santos Assis RA: 11201810271
Liandra Cardoso da Silva RA: 11064916

Resultados

Exercício 2.1

1. (2.0) Calcule $f_{Tn}(\lambda)$ 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 $f_{Tn}(\lambda)$ temos que criar nossa matriz T_n, os valores dessa matriz são definidos pela seguinte forma:

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

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:

```
public static Double[][] TGn(int lambda, int n){

        Double A[][] = new Double[n][n];

        //preenchendo a matriz com zeros
        for(int linha = 0; linha < n; linha++){
            for(int coluna = 0; coluna < n; coluna++){
                A[linha][coluna] = 0d;
            }
        }
        //transformando a matriz em Tn
        for(int i = 0; i < n; i++){
            if(i < n-1){
                A[i][i+1] = -1d;
                A[i+1][i] = -1d;
            }

            A[i][i] = Double.valueOf(lambda);

        }
        return A;
    }
```

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:

```
public static double fTGn(double lambda, int n, boolean debug){

      boolean troca = true;
      double m = 0;
      double temp;
      double determinante = 1;
      Double [][] A = TGn(lambda, n);
```

```
        Double [][] A_original = TGn(lambda, n);
        int sgn = 1;
        int l = 0;
        int[] t = new int[2];
        boolean erro = false;

        ArrayList<int[]> changeLine = new ArrayList<int[]>();

        for (int j = 0; j < n; j++){
            troca = true;
            l = j;
            while(l < n){
                if(A[l][j] == 0 && l != n - 1){
                    l = l + 1;
                    sgn = -sgn;

                }else if(A[l][j] == 0 && l == n - 1){
                    l = l + 1;
                    sgn = -sgn;
                    troca = false;

                }else if(A[l][j] != 0){
                    break;
                }
            }
            if(troca == false && A[j][j] == 0){
                System.err.println("Erro: sistema singular");
                erro = true;
                break;
            }
            else if(troca == true && A[j][j] == 0){
                for(int k = j; k < n; k++){
                    temp = A[j][k];
                    A[j][k] = A[l][k];
                    A[l][k] = temp;
                }
                t[0] = j + 1;
                t[1] = l + 1;
                changeLine.add(t.clone());
            }
            for (int i = j+1; i < n; i++){
                m = - A[i][j]/A[j][j];
                for(int k = j; k < n; k++){
                    A[i][k] = A[i][k] + m*A[j][k];
                }
            }
        }
        for(int i = 0; i < n; i++){
            determinante = determinante*A[i][i];
        }

        if(erro == false){
            determinante = determinante*sgn;
        }
        else{
            determinante = 0;
        }

        if(lambda == 0 && debug){
            System.out.println("V: ");
            printArray(A_original);
            System.out.println("V': ");
            printArray(A);

            if(changeLine.size() == 0){
                System.out.println("trocaLinhas: Não houveram trocas");
            }else{
                System.out.println("trocaLinhas:");
                changeLine.forEach((d) -> printArray(d));
            }
        }
        return determinante;
    }
```

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

```
public static void printArray(Double A[][])
{
    int n = A.length;
    for(int linha = 0; linha < n; linha = linha + 1)
    {
        for(int coluna = 0; coluna < n; coluna = coluna + 1)
        {
            System.out.print("[" + A[linha][coluna] + "]");
```

```
        }
        System.out.print("\n");
    }

}
public static void printArray(int A[])
{
    int n = A.length;

    for(int i = 0; i < n; i = i + 1)
    {
        System.out.print("[" + A[i] + "]");
    }
    System.out.print("\n");

}
```

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 $f_{Tn}(\lambda)$

```
public static void exerc1(){

    int x = 4; // valor máximo que lambda pode valer
    int n = 6; //tamanho da matriz
    double det = 0;
    boolean debug  = true; //variável que permite a impressão da saída

    for(int lambda = 0; lambda <= x; lambda++){

        det = fTGn(lambda, n, debug);
        System.out.println("fT"+n+"("+ lambda +") vale " + det);

    }

}
```

Tendo a seguinte resposta:

```
Erro: sistema singular
V:
[0.0][-1.0][0.0][0.0][0.0][0.0][0.0]
[-1.0][0.0][-1.0][0.0][0.0][0.0][0.0]
[0.0][-1.0][0.0][-1.0][0.0][0.0][0.0]
[0.0][0.0][-1.0][0.0][-1.0][0.0][0.0]
[0.0][0.0][0.0][-1.0][0.0][-1.0][0.0]
[0.0][0.0][0.0][0.0][-1.0][0.0][-1.0]
[0.0][0.0][0.0][0.0][0.0][-1.0][0.0]
V':
[-1.0][0.0][-1.0][0.0][0.0][0.0][0.0]
[0.0][-1.0][0.0][0.0][0.0][0.0][0.0]
[0.0][0.0][-1.0][0.0][-1.0][0.0][0.0]
[0.0][0.0][0.0][-1.0][0.0][0.0][0.0]
[0.0][0.0][0.0][0.0][-1.0][0.0][-1.0]
[0.0][0.0][0.0][0.0][0.0][-1.0][0.0]
[0.0][0.0][0.0][0.0][0.0][0.0][0.0]
trocaLinhas:
[1][2]
[3][4]
[5][6]
fT7(0) vale 0.0
fT7(1) vale 1.0
fT7(2) vale 7.999999999999998
fT7(3) vale 987.0000000000001
fT7(4) vale 10864.0
```

Exercício 2.6

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

$$\lambda_k = 2 * cos\left(\frac{k\pi}{12}\right), \qquad k = 1, 2, ..., 11$$

Calcule os erros

$$|\lambda_k - \lambda_k|, \qquad k = 1, 2, ..., 11$$

Para resolver esse exercícios precisamos primeiro definir quais serão nossos intervalos, cada intervalo irá conter uma raiz, logo serão 11 intervalos para $n = 11$. Como essas raízes não são regularmente espaçadas a distância entre os nossos intervalos também não será de forma que nossos intervalos serão os seguintes:

[2.0, 1.8]
[1.8, 1.5]
[1.5, 1.4]
[1.4, 1.0]
[1.0, 0.5]
[0.5, 0.0]
[0.0, -0.6]
[-0.6, -1.0]
[-1.0, -1.5]
[-1.5, -1.8]
[-1.8, -2.0]

Agora que já sabemos nossos intervalos de interesse podemos declarar a derivada da nossa função que será posteriormente utilizada no método de newton

```
public static double fLinha(double lambda, int n){

    double derivada = 1;
    double z;

    if (lambda == 2){

        derivada = (Math.pow((n + 1),5) - Math.pow((n+1),3))/3;

    }else if(lambda == -2){

        derivada = Math.pow(-1, n+1)*((Math.pow((n + 1),5) - Math.pow((n+1),3))/3);

    }else{
        z = lambda/2;
        derivada = ((n + 1)*Math.cos((n+1)*Math.acos(z)) - z*(Math.sin((n+1)*Math.acos(z)))/(math.sin(math.acos(z))))/(2*(math.pow(z,2
    }

    return derivada;
}
```

O método que calculará a aproximação do $f(x) = 0$ dentro de cada intervalo é o método de Newton e seu código é o seguinte

```
public static double newton(double a, double b, double prec, int n_max, double x0, int n){

    double alpha = x0;
    int i = 0;

    while(fTGn(alpha - prec, n, false)*fTGn(alpha + prec, n, false) > 0 && i <= n_max){

        i = i + 1;

        if(a >= alpha && alpha <= b){
            alpha = alpha - fTGn(alpha, n, false)/fLinha(alpha, n);
        }

    }

    return alpha;
}
```

Por fim devemos chamar esse método dentro do método `main` para executar nosso exercício

```
public static void exerc6(){
    double [] intervalos = {2, 1.8, 1.5, 1.4, 1, 0.5, 0, -0.6, -1, -1.5, -1.8, -2};
    int tamanho_intervalo = intervalos.length;
    double precisao = Math.pow(10,-12);
```

```
    double menor = 0, maior = 0, x0 = 0;
    double raiz_exata = 0;
    double raiz_aprox = 0;
    int n_max = 50;
    int n = 11;
    int k = 1;

    for(int j = 0; j < tamanho_intervalo; j++){

        if(j + 1 <= tamanho_intervalo - 1){

            menor = intervalos[j];
            maior = intervalos[j + 1];
            x0 = (maior+menor)/2;

            raiz_exata = 2*Math.cos(k*Math.PI/12);
            raiz_aprox = newton(menor,maior, precisao, n_max, x0, n);

            System.out.println("~> intervalo: [" + menor +","+ maior+"]");
            System.out.println("~~~> raiz exata: " + raiz_exata);
            System.out.println("~~~> raiz aprox: " + raiz_aprox);
            System.out.println("~~~> erro: " + Math.abs(raiz_exata - raiz_aprox));

        }
        k = k + 1;

    }

}
```

E teremos a seguinte resposta

```
~> intervalo: [2.0, 1.8]
~~~> raiz exata: 1.9318516525781366
~~~> raiz aprox: 1.9
~~~> erro: 0.03185165257813671
~> intervalo: [1.8, 1.5]
~~~> raiz exata: 1.7320508075688774
~~~> raiz aprox: 1.65
~~~> erro: 0.0820508075688775
~> intervalo: [1.5, 1.4]
~~~> raiz exata: 1.4142135623730951
~~~> raiz aprox: 1.45
~~~> erro: 0.03578643762690481
~> intervalo: [1.4, 1.0]
~~~> raiz exata: 1.0000000000000002
~~~> raiz aprox: 1.2
~~~> erro: 0.19999999999999973
~> intervalo: [1.0, 0.5]
~~~> raiz exata: 0.5176380902050415
~~~> raiz aprox: 0.75
~~~> erro: 0.23236190979495852
~> intervalo: [0.5, 0.0]
~~~> raiz exata: 1.2246467991473532e-16
~~~> raiz aprox: 0.25
~~~> erro: 0.2499999999999999
~> intervalo: [0.0, -0.6]
~~~> raiz exata: -0.5176380902050413
~~~> raiz aprox: -0.3
~~~> erro: 0.21763809020504127
~> intervalo: [-0.6, -1.0]
~~~> raiz exata: -0.9999999999999996
~~~> raiz aprox: -0.8
~~~> erro: 0.1999999999999995
~> intervalo: [-1.0, -1.5]
~~~> raiz exata: -1.414213562373095
~~~> raiz aprox: -1.25
~~~> erro: 0.16421356237309492
~> intervalo: [-1.5, -1.8]
~~~> raiz exata: -1.7320508075688774
~~~> raiz aprox: -1.65
~~~> erro: 0.0820508075688775
~> intervalo: [-1.8, -2.0]
~~~> raiz exata: -1.9318516525781364
~~~> raiz aprox: -1.9
~~~> erro: 0.03185165257813649
```

Ao analisar nossa saída podemos perceber que muitas vezes o resultado decorrente do método de Newton não obtém boas aproximações para todas as raízes, isso pode ser decorrente de vários fatores como número escolhido inicialmente como aproximação de x_0 ou mesmo o intervalo escolhido para realizar tal operação.