Atividade Avaliativa 2 de Cálculo Numérico

Samira Haddad RA: 11201812350
Gabrieli Parra Silva RA: 11201721386
Juliane dos Santos Assis RA: 11201810271
Samara Suellen Miranda de Azevedo RA: 11201810807

6 de maio de 2021

1 Introdução

A integral pode ser definida como área sob uma curva no plano cartesiano, trata-se da operação inversa da derivada e tem diversas aplicações práticas dentro de áreas do conhecimento como a biologia, a química e a física, podemos usá-las para, por exemplo, determinar o volume e massa de sólidos arbitrários ou para determinar a posição futura de um corpo a partir da sua posição atual.
Muitas vezes o processo para se calcular uma integral pode ser muito trabalhoso, por isso fez-se necessário a criação de métodos que possibilitassem uma apoximação para esses valores dentro de um intervalo fechado. A seguir apresentaremos alguns métodos para realizar tais aproximações bem como resolver a lista de exercícios apresentada na disciplina NB1MCTB009-17SA do 1° quadrimestre de 2021.
Vale ressaltar que optamos por usar majoritáriamente a linguagem de programação Java por considerarmos que ela seria mais relevante para nosso aprendizado já que todos os membros da equipe já dominavam tal linguagem, além do fato dela possuir uma ampla documentação e ser mais "didática"em comparação com outras linguagens de programação como o C.

2 Polinômios de Lagrange

Os métodos de interpolação tem o objetivo de determinar uma função que englobe certos pontos conhecidos O polinômio de Lagrange é um método de interpolação de tal forma que para vários pontos distintos (x_k, y_k) temos que $P_n(x_k) = y_k$, onde $k = 0, 1, ..., n$. A solução do polinômio de Lagrange obedece a seguinte fórmula:

$$P_n(x) = f_0 \cdot l_0(x) + ... + f_n \cdot l_n(x)$$

onde li é dado por

$$li = \frac{(xx_0)...(xx_{k1})(xx_{k+1})...(xx_n)}{(x_kx_0)...(x_kx_{k1})(x_kx_{k+1})...(x_kx_n)}$$

A implementação de tal método de interpolação é apresentada abaixo, vale ressaltar que esse classe tem como criar tal polinômio para x = pi/6 e para x = 2, como forma de testá-la (os resultados obtidos serão apresentados na subseção "Exercício 1"

```java
public class Lagrange {

    public static void main(String[] args) {
    //declaracao de variaveis
    int i,j;
    int p1=2;
    double p2= 0.523598775;
    int a=0;
    int b=1;
    double h;
    double p;
    int n=6;

    //calculo de xi

    h= 0.1666666666;
    double vetor []=new double [n+1];

    for (i=0;i<n+1;i++) {
        vetor[i]= a+i*h;
      }


         //calcula imprime os resultados para x=pi\6
   System.out.print("x=pi/6  ");
   p=1;
    for(j=0;j<n+1;j++) {
       p=1;
        for(i=0;i<n+1;i++) {
            if (j!=i) {
            p=p*((p2-vetor[i])/(vetor[j]-vetor[i]));
          }

        }

           System.out.print(p+"    ");
        }
        System.out.println();


        //calcula e imprime os resultados para x=2
 System.out.print("x=2  ");
   p=1;
    for(j=0;j<n+1;j++) {
       p=1;
        for(i=0;i<n+1;i++) {
            if (j!=i) {
            p=p*((p1-vetor[i])/(vetor[j]-vetor[i]));
          }

        }

           System.out.print(p+"    ");
        }

    }
```

```
}
```

Listing 1: Método de Lagrange

3 Regra do Trapézio

Como já dito antes precisamos encontrar métodos para que possamos calcular a integral de uma função f no intervalo [a,b] de uma forma mais fácil. Para isso devemos primeiramente dividir o intervalo [a, b] em n vezes com amplitude igual a $h = \frac{(b-a)}{n}$ onde a primeira dessas partições começa com a e a última das partições termina com b de forma que ($x_0 = a$ e $x_n = b$). Logo após isso devemos calcular a área dos trapézios que são formados considerando a base como as distancias entre os valores de x de cada partição da mesma e as alturas como os valores de f(x) dos valores de x da base. Após somar a área dos trapézios de cada subdivisão temos que:

$$\frac{h}{2} \cdot \{f(x_0) + 2f(x_1) + 2f(x_2) + ... + 2f(x_{n-1}) + f(x_n)\}$$

Como já sabemos a integral é o valor obtido pela soma da área de baixo da curva e isso é exatamente o que a regra do Trapézio proporciona. A implementação desse regra, com f(x) sendo um polinômios de lagrange, é a seguinte:

```
public static double trapezio(double a, double b, int m, int n, int i_li){

    double h = (b - a)/m;
    double h_li = (b - a)/n;
    double soma = 0;
    double x = 0;
    int c = 0;
    int i = 0;

    for (int j = 0; j < m + 1; j++){
        i = j;
        if(i == 0 || i == m){
            c = 1;
        }else{
            c = 2;
        }
        x = a + i*h;
        soma = soma + c*li(x, h_li, i_li, a, n);
    }

    double integral = soma*0.5*h;
    return integral;
}
```

Listing 2: Regra do Trapézio

Vale ressaltar que o valor de li foi obtido através do seguinte método:

```
public static double li(double x, double h, int i, double a, int n){

    double x_i = a + i*h;
    double x_j = 0;
    double y = 1;

```

```
    for(int j = 0; j < n+1; j++){
        if(i != j){
            x_j = a + j*h;
            y = y*((x - x_j)/(x_i - x_j));
        }
    }

    return y;
}
```

Listing 3: Valor de li

4 Regra de Simpson

Outro técnica que podemos usar para realizar a integração de uma função em um intervalo é a regra de Simpson, esse método tem o intuito de realizarmos uma aproximação para $f(x)$, assim como no método do trapézio, mas iremos usar uma estratégia mais rebuscada, tentando aproximar a função por um polinômio de grau 2. Isso garante uma resultante mais próximo do valor exato. A fórmula que realiza tal aproximação é a seguinte:

$$\int_a^b f(x)\,dx \approx \frac{h}{3} \cdot \{y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + ... + 2y_{2n2} + 4y_{2n1} + y_{2n}\}$$

A implementação dessa regra pode ser vista abaixo (lembrando que essa implementação terá como função f(x) um polinômios de lagrange):

```
public static double simpson(double a, double b, int m, int n, int i_li){

    double h = (b - a)/m;
    double h_li = (b - a)/n;
    double soma = 0;
    double x = 0;
    int c = 0;
    int i = 0;

    for(int j = 0; j < m+1; j++){
        i = j;
        if(i == 0 || i == m){
            c = 1;
        }
        else if(i%2 == 1){
            c = 4;
        }
        else if(i%2 == 0){
            c = 2;
        }

        x = a + i*h;
        soma = soma + c*li(x, h_li, i_li, a, n);
    }

    double integral = (h*soma)/3;
    return integral;

}
```

Listing 4: Regra de Simpson

5 Exercícios

Exercício 1

Nesse exercício utilizamos o método de Lagrange, apresentado na seção 2, com n = 6. Os resultados obtidos foram os seguintes:

```
x=pi/6  [-0.0021923743233714056    0.01929651858027903    -0.09049918629494093
     0.9728678798444176    0.1203545215512006    -0.022236934417050044
   0.0024095750594652983]

x=2  [ 461.9999999999922    -3023.999999999948    8315.99999999986
   -12319.999999999798    10394.999999999834    -4751.999999999926
   923.9999999999854]
```

Listing 5: Saída do exercício 1

Exercício 2

Para realizar esse exercício e obter os valores de $w_{3,0}$, $w_{3,1}$, $w_{3,2}$ e $w_{3,3}$ teremos que utilizar o método exerc2(), que irá chamar o método trapezio(), passando os seguintes parametros: $n = 3$, $a = 0$, $b = 1$ e $m = 10^5$, e calculará o valor de $w_{n,i}$ como $w_{n,i} = \frac{1}{h} \cdot \int_a^b li(x)\,dx$
A implementação do método exer2() é a seguinte:

```
public static void ex2(){

    double a = 0;
    double b = 1;
    int m = 100000;
    int n = 3;
    double h = (b - a)/n;
    double result = 0;
    for (int i = 0; i < n+1; i++){
        result = trapezio(a, b, m, n, i)/h;
        System.out.println("w["+n+","+i+"] = " + result);
    }
}
```

Listing 6: Método exerc2()

Iremos obter como saída o seguinte:

```
w[3,0] = 0.37500000011249834
w[3,1] = 1.1249999998874738
w[3,2] = 1.1249999998875007
w[3,3] = 0.3750000001124996
```

Listing 7: Saída do exercício 2

Exercício 3

Faremos esse exercício de forma teórica, calculando as frações que se expandem nas dízimas periódicas dos valores obtidos no ítem anterior usando o procedimento que é mostrado na

questão 5.

$w_{3,0} = 0.37500000011249834...$
$w_{3,0} \approx 0.375$
$w_{3,0} \approx \frac{375}{1000}$

$w_{3,0} \approx \frac{3}{8}$

$w_{3,1} = 1.1249999998874738...$
$w_{3,1} \approx 1.124999999...$
$w_{3,1} \approx 1.124 + 0.000999999...$
$w_{3,1} \approx 1.124 + 10^{-4} \times 9 \times 1.111111...$
$w_{3,1} \approx 1.124 + 10^{-4} \times 9 \times (1 + (10^{-1}) + (10^{-2}) + (10^{-3}) + ...)$
$w_{3,1} \approx 1.124 + 10^{-4} \times 9 \times (1 + (10^{-1})^1 + (10^{-1})^2 + (10^{-1})^3 + ...)$
$w_{3,1} \approx 1.124 + 10^{-4} \times 9 \times \frac{1}{1-10^{-1}}$
$w_{3,1} \approx \frac{1124}{1000} + \frac{9}{10^4} \times \frac{10}{9}$

$w_{3,1} \approx \frac{1125}{1000} = \frac{9}{8}$

$w_{3,2} = 1.1249999998875007...$
$w_{3,2} \approx 1.124999999...$
$w_{3,2} \approx 1.124 + 0.000999999...$
$w_{3,2} \approx 1.124 + 10^{-4} \times 9 \times 1.111111...$
$w_{3,2} \approx 1.124 + 10^{-4} \times 9 \times (1 + (10^{-1}) + (10^{-2}) + (10^{-3}) + ...)$
$w_{3,2} \approx 1.124 + 10^{-4} \times 9 \times (1 + (10^{-1})^1 + (10^{-1})^2 + (10^{-1})^3 + ...)$
$w_{3,2} \approx 1.124 + 10^{-4} \times 9 \times \frac{1}{1-10^{-1}}$
$w_{3,2} \approx \frac{1124}{1000} + \frac{9}{10^4} \times \frac{10}{9}$

$w_{3,2} \approx \frac{1125}{1000} = \frac{9}{8}$

$w_{3,3} = 0.3750000001124996...$
$w_{3,3} \approx 0.375$
$w_{3,3} \approx \frac{375}{1000}$

$w_{3,3} \approx \frac{3}{8}$

Exercício 4

Esse exercício se assemelha muito ao exercício 2, porém agora iremos usar n = 6 e iremos usar o método simpson() para poder realizar uma aproximação mais precisa. Para isso criamos um método chamado exerc4() que chamará o método de simpson (apresentado na seção 4), sua implementação é a seguinte:

```
public static void ex4(){
    double a = 0;
    double b = 1;
    int m = 100000;
```

```
    int n = 6;
    double h = (b - a)/n;
    double result = 0;

    for (int i = 0; i < n + 1; i++){
        result = simpson(a, b, m, n, i)/h;
        System.out.println("w["+n+","+i+"] = " + result);
    }
}
```

Listing 8: Método exerc4()

Após executá-lo iremos obter como saída o seguinte:

```
w[6,0] = 0.29285714285714537
w[6,1] = 1.5428571428571185
w[6,2] = 0.19285714285713698
w[6,3] = 1.9428571428571477
w[6,4] = 0.1928571428571369
w[6,5] = 1.5428571428571438
w[6,6] = 0.2928571428571428
```

Listing 9: Saída do exercício 4

Exercício 5

Esse exercício é similar ao exercício 3, vamos calcular teoricamente os valores encontrados no ítem anterior na forma de fração irredutível utilizando o método apresentado no enunciado da questão.

$$w_{6,1} = 1.5428571428571185...$$
$$w_{6,1} \approx 1.54285714285714...$$
$$w_{6,1} \approx 1.54 + 0.00285714285714...$$
$$w_{6,1} \approx 1.54 + 10^{-8} \times 285714 \times 1.000001000001...$$
$$w_{6,1} \approx 1.54 + 10^{-8} \times 285714 \times (1 + (10^{-6}) + (10^{-12}) + (10^{-18}) + ...)$$
$$w_{6,1} \approx 1.54 + 10^{-8} \times 285714 \times (1 + (10^{-6})^1 + (10^{-6})^2 + (10^{-6})^3 + ...)$$
$$w_{6,1} \approx 1.54 + 10^{-8} \times 285714 \times \frac{1}{1-10^{-6}}$$
$$w_{6,1} \approx \frac{154}{100} + \frac{285714}{10^8} \times \frac{10^6}{10^6-1}$$

$$w_{6,1} \approx \frac{154285560}{99999900}$$

$$w_{6,2} = 0.19285714285713698...$$
$$w_{6,2} \approx 0.19285714285714...$$
$$w_{6,2} \approx 0.19 + 0.00285714285714...$$
$$w_{6,2} \approx 0.19 + 10^{-8} \times 285714 \times 1.000001000001...$$
$$w_{6,2} \approx 0.19 + 10^{-8} \times 285714 \times (1 + (10^{-6}) + (10^{-12}) + (10^{-18}) + ...)$$
$$w_{6,2} \approx 0.19 + 10^{-8} \times 285714 \times (1 + (10^{-6})^1 + (10^{-6})^2 + (10^{-6})^3 + ...)$$
$$w_{6,2} \approx 0.19 + 10^{-8} \times 285714 \times \frac{1}{1-10^{-6}}$$
$$w_{6,2} \approx \frac{19}{100} + \frac{285714}{10^8} \times \frac{10^6}{10^6-1}$$

$$w_{6,2} \approx \frac{19285695}{99999900}$$

$$w_{6,3} = 1.9428571428571477...$$
$$w_{6,3} \approx 1.94285714285714...$$
$$w_{6,3} \approx 1.94 + 0.00285714285714...$$
$$w_{6,3} \approx 1.94 + 10^{-8} \times 285714 \times 1.000001000001...$$
$$w_{6,3} \approx 1.94 + 10^{-8} \times 285714 \times (1 + (10^{-6}) + (10^{-12}) + (10^{-18}) + ...)$$
$$w_{6,3} \approx 1.94 + 10^{-8} \times 285714 \times (1 + (10^{-6})^1 + (10^{-6})^2 + (10^{-6})^3 + ...)$$
$$w_{6,3} \approx 1.94 + 10^{-8} \times 285714 \times \frac{1}{1-10^{-6}}$$
$$w_{6,3} \approx \frac{194}{100} + \frac{285714}{10^8} \times \frac{10^6}{10^6-1}$$

$$w_{6,3} \approx \frac{194285520}{99999900}$$

$$w_{6,4} = 0.1928571428571369...$$
$$w_{6,4} \approx 0.19285714285714...$$
$$w_{6,4} \approx 0.19 + 0.00285714285714...$$
$$w_{6,4} \approx 0.19 + 10^{-8} \times 285714 \times 1.000001000001...$$
$$w_{6,4} \approx 0.19 + 10^{-8} \times 285714 \times (1 + (10^{-6}) + (10^{-12}) + (10^{-18}) + ...)$$
$$w_{6,4} \approx 0.19 + 10^{-8} \times 285714 \times (1 + (10^{-6})^1 + (10^{-6})^2 + (10^{-6})^3 + ...)$$
$$w_{6,4} \approx 0.19 + 10^{-8} \times 285714 \times \frac{1}{1-10^{-6}}$$
$$w_{6,4} \approx \frac{19}{100} + \frac{285714}{10^8} \times \frac{10^6}{10^6-1}$$

$$w_{6,4} \approx \frac{19285695}{99999900}$$

$$w_{6,5} = 1.5428571428571438...$$
$$w_{6,5} \approx 1.54285714285714...$$
$$w_{6,5} \approx 1.54 + 0.00285714285714...$$
$$w_{6,5} \approx 1.54 + 10^{-8} \times 285714 \times 1.000001000001...$$
$$w_{6,5} \approx 1.54 + 10^{-8} \times 285714 \times (1 + (10^{-6}) + (10^{-12}) + (10^{-18}) + ...)$$
$$w_{6,5} \approx 1.54 + 10^{-8} \times 285714 \times (1 + (10^{-6})^1 + (10^{-6})^2 + (10^{-6})^3 + ...)$$
$$w_{6,5} \approx 1.54 + 10^{-8} \times 285714 \times \frac{1}{1-10^{-6}}$$
$$w_{6,5} \approx \frac{154}{100} + \frac{285714}{10^8} \times \frac{10^6}{10^6-1}$$

$$w_{[6,5]} \approx \frac{154285560}{99999900}$$

$$w_{6,6} = 0.2928571428571428...$$
$$w_{6,6} = 0.29 + 0.00285714285714...$$
$$w_{6,6} = 0.29 + 10^{-8} \times 285714 \times 1.000001000001...$$
$$w_{6,6} = 0.29 + 10^{-8} \times 285714 \times (1 + (10^{-6}) + (10^{-12}) + (10^{-18}) + ...)$$
$$w_{6,6} = 0.29 + 10^{-8} \times 285714 \times (1 + (10^{-6})^1 + (10^{-6})^2 + (10^{-6})^3 + ...)$$
$$w_{6,6} = 0.29 + 10^{-8} \times 285714 \times \frac{1}{1-10^{-6}}$$
$$w_{6,6} = \frac{29}{100} + \frac{285714}{10^8} \times \frac{10^6}{10^6-1}$$

$$w_{6,6} = \frac{29285685}{99999900}$$

Para chegar ao resultado irredutível de forma mais prática, utilizaremos um pequeno algoritmo.

```java
public class FracaoIrredutivel {
    public static void main(String args[]) {

        int a[] = {154285560, 19285695, 194285520, 19285695, 154285560,
    29285685};
        int b = 99999900;
        int i = 2;
        for(int j=0;j<a.length;j++){
            while(i<10000){
                if(a[j]%i==0 &&b%i==0){
                    a[j]=a[j]/i;
                    b=b/i;
                    i=2;
                }
                else{
                    i++;
                }
            }
            int x=j+1;
            System.out.println("w[6,"+x+"]= "+a[j]+"/"+b);
            b=99999900;
            i=2;
        }
    }
}
```

Listing 10: Fração Irredutível

Executando-o, obtemos a seguinte saída:

```
w[6,1]= 54/35
w[6,2]= 27/140
w[6,3]= 68/35
w[6,4]= 27/140
w[6,5]= 54/35
w[6,6]= 41/140
```

Listing 11: Saída do exercício 5

Exercício 6

O exercício 6 usa de um outro método para descobrir os valores das constantes $w_{6,0}$, $w_{6,1}$, $w_{6,2}$, $w_{6,3}$, $w_{6,4}$, $w_{6,5}$ e $w_{6,6}$. O método que iremos utilizar agora é o método de eliminação de Gauss, e ele pretende basicamente resolver um sistema linear tal como o proposto no enunciado. A estratégia utilizada nesse algoritmo é a de transformar a matriz por escalonamento em um sistema triangularizado para que se possa chegar facilmente a uma solução mesmo em um sistemas com muitas incógnitas. A implementação desse algoritmo é a seguinte:

```java
public static double[] gauss(double[][] A, double[] y, int n){
    boolean troca = true;
    double m = 0;
    double temp;
    double determinante = 1;
    int sgn = 1;
    int l = 0;
    boolean erro = false;
```

```
    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){
            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;
            }
            temp = y[j];
            y[j] = y[l];
            y[l] = temp;

        }
        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];
            }
            y[i] = y[i] + m*y[j];
        }
    }
    for(int i = 0; i < n; i++){
        determinante = determinante*A[i][i];
    }

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

    double[] x = new double[n];
    x[n-1] = y[n-1]/A[n-1][n-1];

    for(int i = n - 2; i >= 0; i--){
        x[i] = y[i];
        for (int k = i+1; k < n; k++){
            x[i] = x[i] - A[i][k]*x[k];
```

```
        }
        x[i] = x[i]/A[i][i];
    }

    return x;
}
```

Listing 12: Método de eliminação de Gauss

Para gerar o sistema linear que será resolvido e executar o método de eliminação de Gauss, que irá nos retornar os valores de $w_{n,i}$, iremos chamar a seguinte função:

```
public static void ex6(){

    double a = 0;
    double b = 1;
    int n = 6;
    double h = (b-a)/n;
    double[] x = new double[n+1];
    double[][] A = new double[n+1][n+1];
    double[] y = new double[n+1];

    for(int i = 0; i < n+1; i++){
        x[i] =  a + i*h;
    }
    for(int i = 0; i < n+1; i++){
        for(int j = 0; j < n+1; j++){
            A[i][j] = (double) Math.pow(x[j], i);
        }
    }
    for(int i = 0; i < n+1; i++){
        y[i] = (double) n/(i+1);
    }
    double[] w = gauss(A, y, n+1);
    for(int linha = 0; linha < n + 1; linha++)
    {
        System.out.print("w["+ n +"," + linha + "] = " + w[linha]);
        System.out.print("\n");
    }

}
```

Listing 13: Método exerc6()

E a resultante da execução do método acima será o seguinte:

```
w[6,0] = 0.29285714285711273
w[6,1] = 1.5428571428572693
w[6,2] = 0.19285714285692285
w[6,3] = 1.942857142857349
w[6,4] = 0.19285714285703098
w[6,5] = 1.5428571428571767
w[6,6] = 0.2928571428571387
```

Listing 14: Saída do exercício 6

Exercício 7

Nesse exercício utilizamos os métodos de Trapézio e de Simpson que foram implementados nos exercícios 2 e 3 para calcular a integral da função dada e inserimos um looping para que pudéssemos obter um resultado para cada um dos dez valores de m. Nesse mesmo método calculamos os valores do logaritmo natural do erro de Em, Es e m e os armazenamos. Com os dados armazenados calculamos os coeficientes angular e linear da retas ajustadas tanto para Em como para Es.

```
public class Exercicio7 {

    public static void main(String[] args) {
        ex7();

      }

     public static void ex7(){
    double a = 0;
    double b = 1;
    int m = 6;
    double ETm = 0;
    double ESm = 0;


    double x=0; //somatorio m
    double y1=0; //somatorio Etm
    double xy1=0; //somatorio m*Etm;
    double xquad=0; //somatorio m ao quadrado

    double y2=0; //somatorio ESm
    double xy2=0; //somatorio m*ESm;
    double angular1=0;
    double linear1=0;

    double angular2=0;
     double linear2=0;

        while(m<=60)  {

            ETm = trapezio_f(a, b, m) - integral_f(a, b);
            ESm = simpson_f(a, b, m) - integral_f(a, b);


       //calculo do MMQ para ETm

             x=x+Math.log(m);
             xquad=xquad+(Math.pow(Math.log(m),2));

            y1=y1+Math.log(ETm);
            xy1=xy1+(Math.log(m)*+Math.log(ETm));



       //calculo do MMQ para ESm


            y2=y2+Math.log(ESm);
```

```
            xy2=xy2+(Math.log(m)*+Math.log(ESm));

            System.out.println("Para m: "+ m);
            System.out.println("    log(m): " + Math.log(m));
            System.out.println("    log(ETm): " + Math.log(ETm));
            System.out.println("    log(ESm): " + Math.log(ESm));
             m = m + 6;
        }

         System.out.println();

         System.out.println("Para Etm os coeficientes obtidos pelo MMQ sao: ");

         angular1= (10*(xy1)-x*y1)/(10*xquad-Math.pow(x,2));
         System.out.println("a: "+angular1);

         linear1=(x*xy1-y1*xquad)/(Math.pow(x,2)-10*xquad);
         System.out.println("b: "+linear1);

       System.out.println("Para ESm os coeficientes obtidos pelo MMQ sao: ");
          angular2= (10*(xy2)-x*y2)/(10*xquad-Math.pow(x,2));
         System.out.println("a: "+angular2);

         linear2=(x*xy2-y2*xquad)/(Math.pow(x,2)-10*xquad);
         System.out.println("b: "+linear2);


    }
```

Listing 15: Mínimos Quadrados

Os resultados obtidos foram:

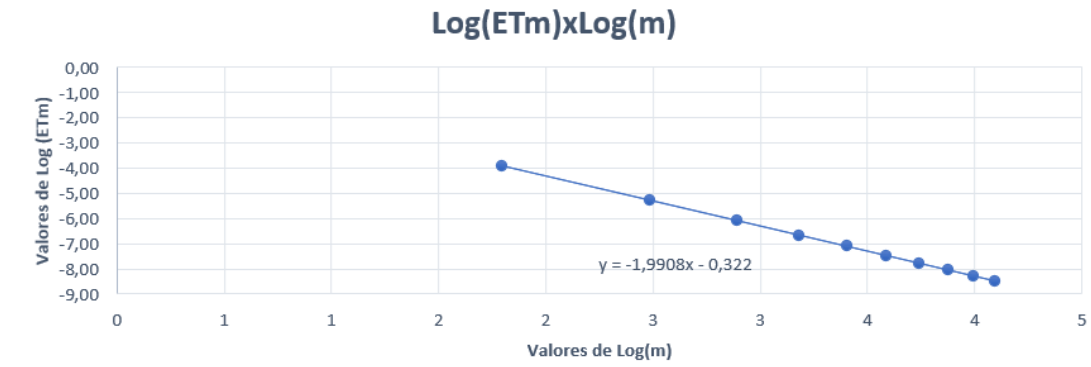
```
Para m: 6
    log(m): 1.791759469228055
    log(ETm): -3.8969453809623307
    log(ESm): -6.238050314243682
Para m: 12
    log(m): 2.4849066497880004
    log(ETm): -5.263965657507151
    log(ESm): -8.93489900215015
Para m: 18
    log(m): 2.8903717578961645
    log(ETm): -6.071304040674812
    log(ESm): -10.542908400945056
Para m: 24
    log(m): 3.1780538303479458
    log(ETm): -6.645409406139206
    log(ESm): -11.688801955433709
Para m: 30
    log(m): 3.4011973816621555
    log(ETm): -7.091113587283377
    log(ESm): -12.579140680172673
Para m: 36
    log(m): 3.58351893845611
    log(ETm): -7.455439976306069
    log(ESm): -13.307213186032602
Para m: 42
    log(m): 3.7376696182833684
```

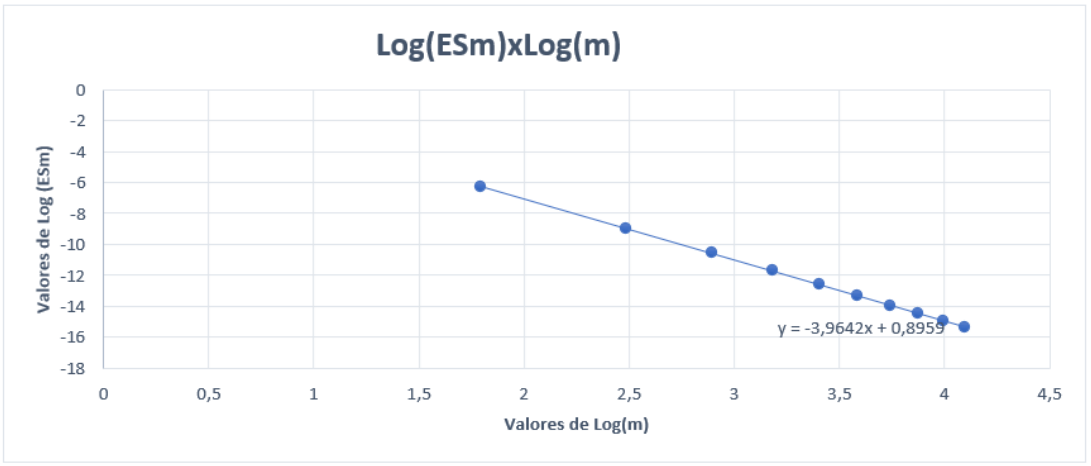
```
    log(ETm): -7.763550335248435
    log(ESm): -13.92308427859827
Para m: 48
    log(m): 3.871201010907891
    log(ETm): -8.030489143311172
    log(ESm): -14.456735079401726
Para m: 54
    log(m): 3.9889840465642745
    log(ETm): -8.265970211554288
    log(ESm): -14.927541759791335
Para m: 60
    log(m): 4.0943445622221
    log(ETm): -8.476630438319452
    log(ESm): -15.348751040135749

Para Etm os coeficientes obtidos pelo MMQ s o:
a: -1.9908251517781022
b: -0.32197755512359927
Para ESm os coeficientes obtidos pelo MMQ s o:
a: -3.9642053761962472
b: 0.8958893037210867
```

Listing 16: Saída do exercício 7

Os gráficos obtidos foram:





Exercício 8

Nesse exercício, a partir da função apresentada no exercício 7 ($f(x) = x^9$), dentro do intervalo [0, 1], verifica a partir da somatória dos resultados da função multiplicados pelos coeficientes w qual é o melhor método para a aproximação do resultado da integral. No método NC*, o erro esperado é de ± 0,0001. Comparado ao erro dos exercícios anteriores, este erro é equivalente ao encontrado no Sm, sendo portanto os dois métodos mais eficazes para a aproximação do problema.

```
public class MyClass {
 //Declarando 'fora do main' para que possa ser acessado em outro m dulos
 public static void main(String[] args) {
 //A instru  o abaixo obriga o programa a rodar com configura  o de PONTO
    DECIMAL
 java.util.Locale.setDefault(new java.util.Locale("en","US"));
  //O vetor w abriga os valores dos coeficientes W
  //Manter os valores em vetor facilita no somatorio
  double w[] = {41.0/140.0, 54.0/35.0, 27.0/140.0, 68.0/35.0 , 27.0/140.0,
    54.0/35.0, 41.0/140.0};
  double f=0.0, x=0.0, b=1.0, i=0.0;
   double soma=0.0,vetor_x[] = new double[15],vetor_funcao[] = new double[15],
    aprox=0.0,multiw=0.0;
   int a=0,j=0,c=0;
   int m=6;
   //Calcula valores de x para aplicar na funcao
   while( m<=60){
        f=0.0; x=0.0; b=1.0; i=0.0;
        soma=0.0;  aprox=0.0; multiw=0.0;
        a=0;j=0;c=0;
   for (i=0.0;i<15.0;i++){
    x = i*( b/m);
    if (i>6){
     x= ( i-1)* ( b/m);
    }
    if (i==0 || i==6 || i==7){
     x = 0.0;
    }
    vetor_x[j]  =  x;
    j = j+1;
   }
   //Calcula a funcao
   for (j=0;j<15;j++){
    vetor_funcao[j] = Math.pow( vetor_x[j], 9.0);
   }
   //Calcula a soma de todos os valores
   c = 0;
   for (j=0;j<14;j++){
    if (c>6){
     c = 0;
    }
    multiw  =  w[c]*vetor_funcao[j];
    soma  =  soma+multiw;
    c = c+1;
   }
   aprox = ( b/m)*soma;
   double erro = 0.1-aprox;
```

```
45    System.out.println("Para m = "+m);
46    System.out.println("log(ENCm): "+Math.log(erro));
47    m=m+6;
48   }
49  }}
```

Listing 17: Método exerc8()