```
package mat;
import java.util.*;
import javax.swing.JOptionPane;

/**
 * Classe criada com o intuito de lidar com os cálculos necessários no
programa
 * @author Danyel Clinário dos Santos
 * @email danyel.clinario@poli.ufrj.br
 */

public class MatFunc {

	public static double func (double[] consts, double ponto) {
		double resultado = consts[0] * Math.pow(Math.E, (consts[1] *
ponto)) + consts[2] * Math.pow(ponto, consts[3]);
		return resultado;
	}

	public static double func_derivada (double[] consts, double ponto)
{
		double resultado = consts[0] * consts[1] * Math.pow(Math.E,
(consts[1] * ponto)) + consts[2] * consts[3] *  Math.pow(ponto,
(consts[3]-1));
		return resultado;
	}

	public static double bissecao(double[] consts, double[] pontos,
double tolm) throws Exception {
		if (func(consts, pontos[0]) * func(consts, pontos[1]) > 0) {
			JOptionPane.showMessageDialog(null, "ERRO - Não há
raízes no intervalo");
			throw new Exception();
		}
		double raiz = 0;
		while (Math.abs(pontos[1] - pontos[0]) > tolm) {
			raiz = (pontos[0] + pontos[1]) / 2;
			if (func(consts, raiz) == 0) {
				return raiz;
			}
			if (func(consts, pontos[0]) * func(consts, raiz) < 0){
				pontos[1] = raiz;
			}else {
				pontos[0] = raiz;
			}
		}
		return raiz;
	}

	public static double raiz_newton(double[] consts, double[] pontos,
double tolm) throws Exception {
		if (func(consts, pontos[0]) * func(consts, pontos[1]) > 0) {
			JOptionPane.showMessageDialog(null, "ERRO - Não há
raízes no intervalo");
```

```
                throw new Exception();
            }
            double raiz = (pontos[0] + pontos[1]) / 2;
            int cont=0;
            while(cont<1000) {
                cont++;
                double raiz_anterior = raiz;
                raiz = raiz - func(consts, raiz) / func_derivada(consts,
raiz);
                if (Math.abs(raiz_anterior - raiz) < tolm){
                    return raiz;
                }
            }
            JOptionPane.showMessageDialog(null,"AVISO - Não foi possível
convergir a um resultado adequado");
            System.out.println("FALHOU");
            throw new Exception();
      }

      public static double quad_poli(double[] consts, double[] pontos,
int n) {
          double L = pontos[1] - pontos[0];
          double delta = (pontos[1] - pontos[0]) / (n - 1);
          double[] x=new double[n];
          double[] w=new double[n];
          for (int i=0; i<n; i++) {
            x[i] = pontos[0] + delta * i;
          }
          double I = 0;

          if (n == 2){
            double[] temp = {L/2,L/2};
            w=temp;
          }
          if (n == 3) {
            double[] temp = {L/6,(2*L)/3,L/6};
            w=temp;
          }
          if (n == 4) {
            double[] temp = {L/8,(3*L)/8, (3*L)/8, L/8};
            w=temp;
          }else{
            double[] temp = {(7*L)/90, (16*L)/45, (2*L)/15, (16*L)/45,
(7*L)/90};
            w=temp;
          }
          for(int i = 0; i<n; i++) {
              I += func(consts, x[i]) * w[i];
          }
          return I;
      }
      public static double quad_gauss(double[] consts, double[] pontos,
int n) throws Exception {
            double dif2 = (pontos[1] - pontos[0]) / 2;
```

```
            double temp = (pontos[0] + pontos[1]) / 2;
            double I = 0;
            double[][][] tabela_quad = { {}, {{2}, {0}},
                        { {1.0, 1.0}, {0.57735, -0.57735} },
                        { {0.555556, 0.88889, 0.55556}, {0.77459, 0, -
0.77459} },
                        { {0.34786, 0.65215, 0.65215, 0.34786}, {0.86114,
0.33998, -0.33999, -0.86114} },
                        { {0.23693, 0.47863, 0.56889, 0.47863, 0.23693},
{0.90618, 0.53847, 0, -0.53847, -0.90618} },
                        { {0.17133, 0.36076, 0.46791, 0.46791, 0.36076,
0.17133}, {0.93247, 0.66121, 0.23862, -0.23862, -0.66121, -0.93247} },
                        { {0.12949, 0.27971, 0.38183, 0.41796, 0.38183,
0.27971, 0.12949}, {0.94911, 0.74153, 0.40585, 0, -0.40585, -0.74153, -
0.94911} },
                        { {0.10123, 0.22238, 0.31371, 0.36268, 0.36268,
0.31371, 0.22238, 0.10123}, {0.96029, 0.79667, 0.53553, 0.18343, -
0.18343, -0.53553, -0.79667, -0.96029} },
                        { {0.08127, 0.18065, 0.26061, 0.31235, 0.33024,
0.31235, 0.26061, 0.18065, 0.08127}, {0.96816, 0.83603, 0.61337, 0.32425,
0, -0.32425, -0.61337, -0.83603, -0.96816} },
                        { {0.06667, 0.14945, 0.21909, 0.26927, 0.29552,
0.29552, 0.26927, 0.21909, 0.14945, 0.06667}, {0.97391, 0.86506, 0.67941,
0.43339, 0.14887, -0.14887, -0.43339, -0.67941, -0.86506, -0.97391} }
            };
            for (int i =0; i<n; i++) {
                  double xi = dif2 * tabela_quad[n][1][i] + temp;
                  I += func(consts, xi) * tabela_quad[n][0][i];
            }
            return I * dif2;
      }


      public static double deriv_frente(double[] consts, double ponto,
double delta) {
            double funcx = func(consts, ponto);
            double fdevx = func(consts, ponto + delta);
            return (fdevx - funcx) / delta;
      }

      public static double deriv_atras(double[] consts, double ponto,
double delta) {
            double funcx = func(consts, ponto);
            double f_atualizado = func(consts, ponto - delta);
            return (funcx - f_atualizado) / delta;
      }

      public static double deriv_central(double[] consts, double ponto,
double delta) {
            double fdeltamais = func(consts, ponto + delta);
            double fdeltamenos = func(consts, ponto - delta);
            return (fdeltamais - fdeltamenos) / (2 * delta);
      }
```

```
      public static double richard_ext(double[] consts, double ponto,
double delta1, double delta2) throws Exception {
            if(delta1==delta2) {
                  JOptionPane.showMessageDialog(null,"ERRO - Use valores
diferentes para delta_x1 e delta_x2");
                  System.out.println("FALHOU");
                  throw new Exception();
            }
            double deriv1 = deriv_frente(consts, ponto, delta1);
            double deriv2 = deriv_frente(consts, ponto, delta2);
            double quociente =delta2 / delta1;
            return deriv1 + (deriv1 - deriv2) / (quociente - 1);
      }


}
```