MATHEUS YOSHIMITSU TAMASHIRO PIRES LANZO

UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
Câmpus de São José do Rio Preto

ANÁLISE DE DADOS OBTIDOS A PARTIR DE EQUAÇÕES NÃO LINEARES

A primeira parte do trabalho consistiu em montar os algoritmos usando o Método da Regula Falsi e Método de Newton-Raphson. Começando com a implementação de Regula Falsi:

```
//function y = funcaoTeste(x)
//    y = x^2 - 7.6*x + 11.55;
//endfunction

function y = funcaoContinua(x);
    y = sin((%pi*(x + 1)) / 8) + (0.23 * x) - 1.5;
endfunction

function [raiz, x, iter, ea] = regulaFalsi(xl, xu, f, tol, imax)
    iter = 0;
    xr = xu-(f(xu)*(xl-xu))/(f(xl)-f(xu));
    a = f(xl)*f(xr);
    if((f(xl)*f(xr))<0) then
        xu=xr;
    elseif((f(xl)*f(xr))>0) then
        xl=xr;
    else
        ea(iter+1)=0;
    end
    xrold = xr;
    while(1) then
        xr = xu - (f(xu) * (xl-xu)) / (f(xl) - f(xu));
        iter = iter + 1;
        ea(iter) = abs((xr - xrold)/xr);
        if ((f(xl) * f(xr)) < 0) then
            xu = xr;
        elseif ((f(xl)*f(xr)) > 0) then
            xl = xr;
        else
            ea(iter) = 0;
        end
        xrold=xr;
        if(ea(iter)>tol && iter<imax) then
            raiz = xr;
            disp(xl, xu);
            return;
        end
    end
endfunction
```

A função utilizada para testar se o código estava correto foi a funcaoTeste comentada. O resultado esperado era que com essa função, a raiz encontrada fosse 2.1.

```
--> regulaFalsi(1, 3, funcao, 0.001, 5)

   1.

   2.1715976
 ans  =

   2.1715976
```

É possível perceber que sua aproximação está correta ao esperado.

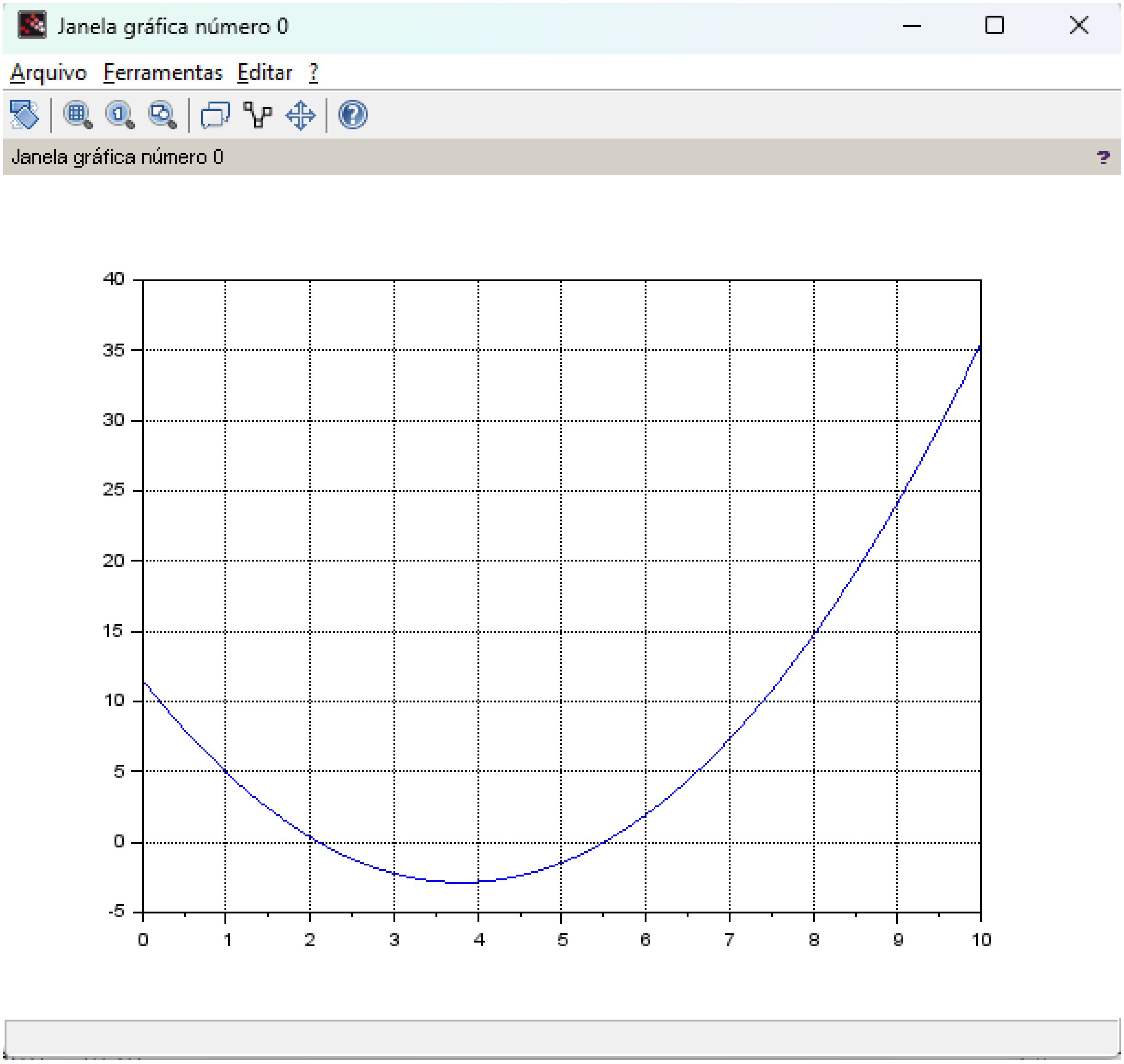
Agora, apresentando a implementação de Newton-Raphson:

```
//function y = funcaoTeste(x)
//    y = x^2 - 7.6*x + 11.55;
//endfunction

//function z = funcaoDerivadaTeste(x);
//    z = 2*x - 38/5;
//endfunction

function y = funcaoContinua(x);
    y = sin((%pi*(x + 1)) / 8) + 0.23 * x - 1.5;
endfunction

// Usei o aplicativo Photomath para encontrar a derivada da funcao continua
function z = funcaoDerivada(x);
    z = (%pi*cos((%pi*x+%pi)/8)) / 8 + (23/100);
endfunction

function [raiz, x, iter, ea]=newtonRaphson(x0,f,fp,tol,imax)
iter = 0; // inicializa numero de iteracoes
xr = x0; // inicializa raiz aproximada com a inicial
x(iter+1)=x0; // insere raiz inicial no vetor de raizes
while (1)
    xrold = xr;
    xr = xrold - f(xrold)/fp(xrold); // aplica formula de Newton
    iter = iter+1; // incrementa numero de iteracoes
    x(iter+1) = xr; // insere raiz aproximada no respectivo vetor
    if(xr ~= 0) then // calcula erro relativo
        ea(iter)=abs((xr-xrold)/xr);
    end;
    if(ea(iter) <= tol) then // se erro relativo menor que tol, FIM
        raiz = xr;
        disp('Iteracoes necessarias: ', iter);
        return;
    end;
    if(iter >= imax) then // se excedeu num. maximo de iteracoes, FIM
        error('Número Máximo de Iterações Alcançado');
    end;
end
endfunction
```

A função como teste é a mesma da implementação anterior, também gerando o resultado esperado:

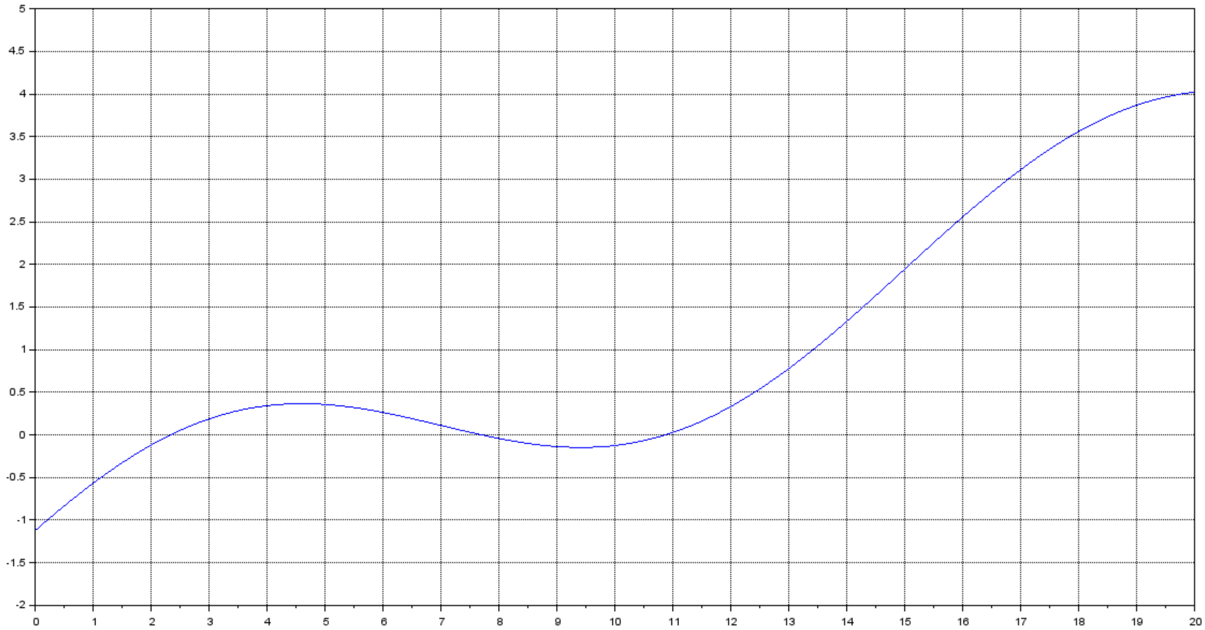
```
--> newtonRaphson(2.5, funcaoTeste, funcaoDerivada, 0.001, 10)
 ans  =

   2.0999997
```

Além disso, o gráfico plotado, resultante da função de teste, confirma os valores obtidos:



Agora que sabemos que as implementações estão funcionando corretamente, é hora de fazer as aproximações da função da atividade.

Plot da função da atividade:



IMPLEMENTAÇÃO:

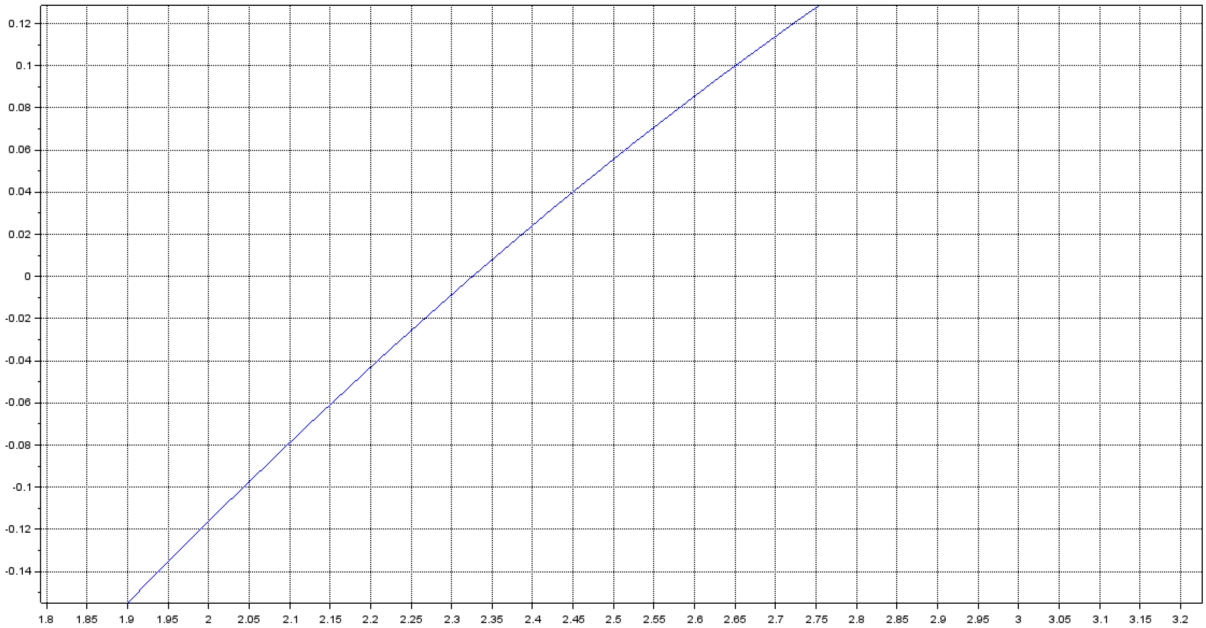
Encontrando os intervalos do primeiro zero da função usando Regula Falsi:

```
--> regulaFalsi(0, 5, funcaoContinua, 0.000001, 3)

   0.

   2.9375517
 ans  =

   2.9375517

--> regulaFalsi(1, 3, funcaoContinua, 0.000001, 3)

   1.

   2.3636465
 ans  =

   2.3636465

--> regulaFalsi(2, 2.5, funcaoContinua, 0.000001, 3)

   2.

   2.3263511
 ans  =

   2.3263511
```

Encontrando as aproximações e o número de iterações do primeiro zero da função usando Newton-Raphson:

```
--> newtonRaphson(0, funcaoContinua, funcaoDerivada, 0.000001, 10)

  "Iteracoes necessarias: "

   5.
 ans  =

   2.3255298
--> newtonRaphson(2, funcaoContinua, funcaoDerivada, 0.000001, 4)

  "Iteracoes necessarias: "

   4.
 ans  =

   2.3255298
```

Plot aproximado do primeiro zero:



Encontrando os intervalos do segundo zero da função usando Regula Falsi:

```
--> regulaFalsi(5, 10, funcaoContinua, 0.000001, 3)

  5.

  7.7841156
ans  =

  7.7841156
```

```
--> regulaFalsi(6, 8, funcaoContinua, 0.000001, 3)

  6.

  7.6981003
ans  =

  7.6981003
```

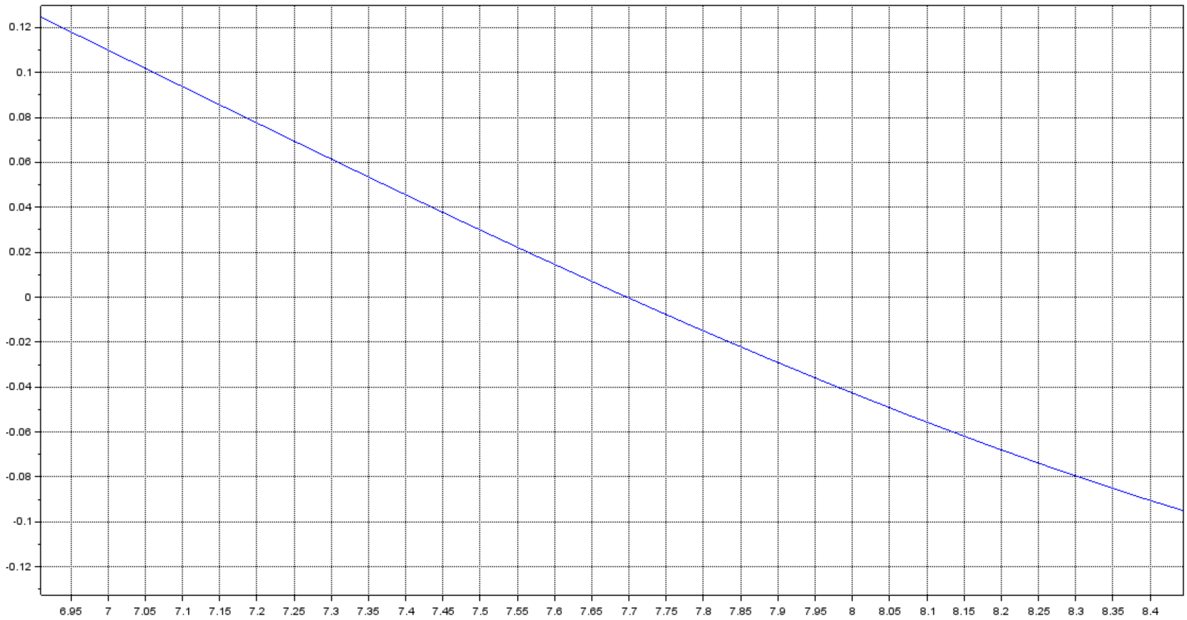
```
--> regulaFalsi(7.5, 8, funcaoContinua, 0.000001, 3)

  7.5

  7.6972462
ans  =

  7.6972462
```

Encontrando as aproximações e o número de iterações do segundo zero da função usando Newton-Raphson:

```
--> newtonRaphson(9, funcaoContinua, funcaoDerivada, 0.000001, 10)

 "Iteracoes necessarias: "

  5.
ans  =

  7.6970243
```

```
--> newtonRaphson(7, funcaoContinua, funcaoDerivada, 0.000001, 10)

 "Iteracoes necessarias: "

  4.
ans  =

  7.6970243
```

Plot aproximado do segundo zero:



Encontrando os intervalos do terceiro zero da função usando Regula Falsi:

```
--> regulaFalsi(9, 15, funcaoContinua, 0.000001, 10)

   9.7871492

   15.
 ans  =

   9.7871492

--> regulaFalsi(9, 12, funcaoContinua, 0.000001, 10)

   10.473608

   12.
 ans  =

   10.473608

--> regulaFalsi(10, 11, funcaoContinua, 0.000001, 10)

   10.860449

   11.
 ans  =

   10.860449
```

Encontrando as aproximações e o número de iterações do terceiro zero da função usando Newton-Raphson:

```
--> newtonRaphson(15, funcaoContinua, funcaoDerivada, 0.000001, 10)

  "Iteracoes necessarias: "

   6.
 ans  =

   10.863302

--> newtonRaphson(10, funcaoContinua, funcaoDerivada, 0.000001, 10)

  "Iteracoes necessarias: "

   5.
 ans  =

   10.863302
```

Plot aproximado do terceiro e último zero: