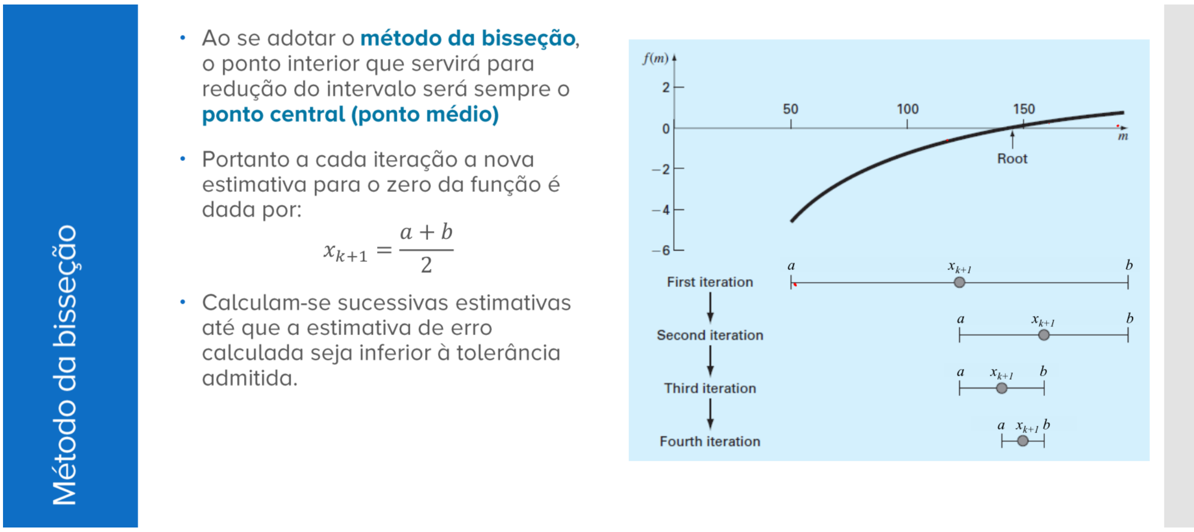
Monitoria: Cálculo Numérico (EPET-019A)

Data: 08/04/2021

- **Monitores:**
  - Paulo Victor L. Santos
  - Leonardo T. Ferreira
  - Ricardo A. Fernandes

- **Assuntos abordados:**
  - Zero de funções
  - Sistemas de equações lineares
  - Sistemas de equações não lineares

Zero de funções: Método da Bisseção

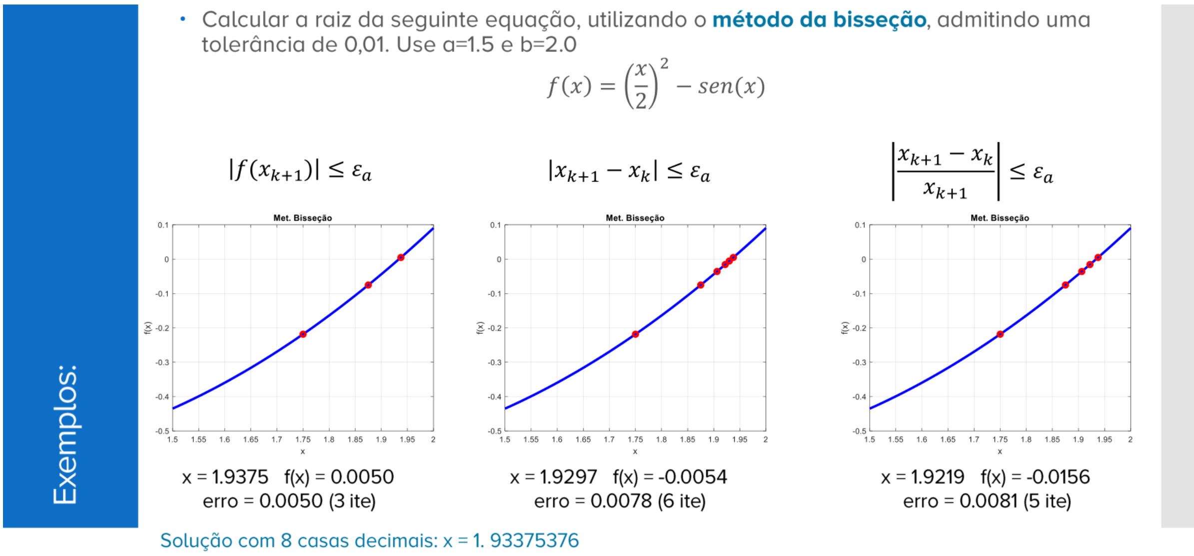


In [1]:

```julia
function metodo_bissecao(f, a, b, erro_max, n_int_max)

    # Verificação da existência de raiz no intervalo dado
    @assert f(a)*f(b) ≤ 0

    # metodo
    iteracao = 0
    c0 = Inf

    while iteracao < n_int_max

        iteracao += 1

        c = (a+b)/2
        if f(a)*f(c) < 0
            b = c
        elseif f(b)*f(c) < 0
            a = c
        end

        # Calcular erro
        # erro = abs(f(c))
        # erro = abs(c-c0)
        erro = abs((c-c0)/c)
```

```julia
        # Checar erro
        if erro ≤ erro_max
            return c, iteracao, erro
        end
        c0 = c
    end
end;
```

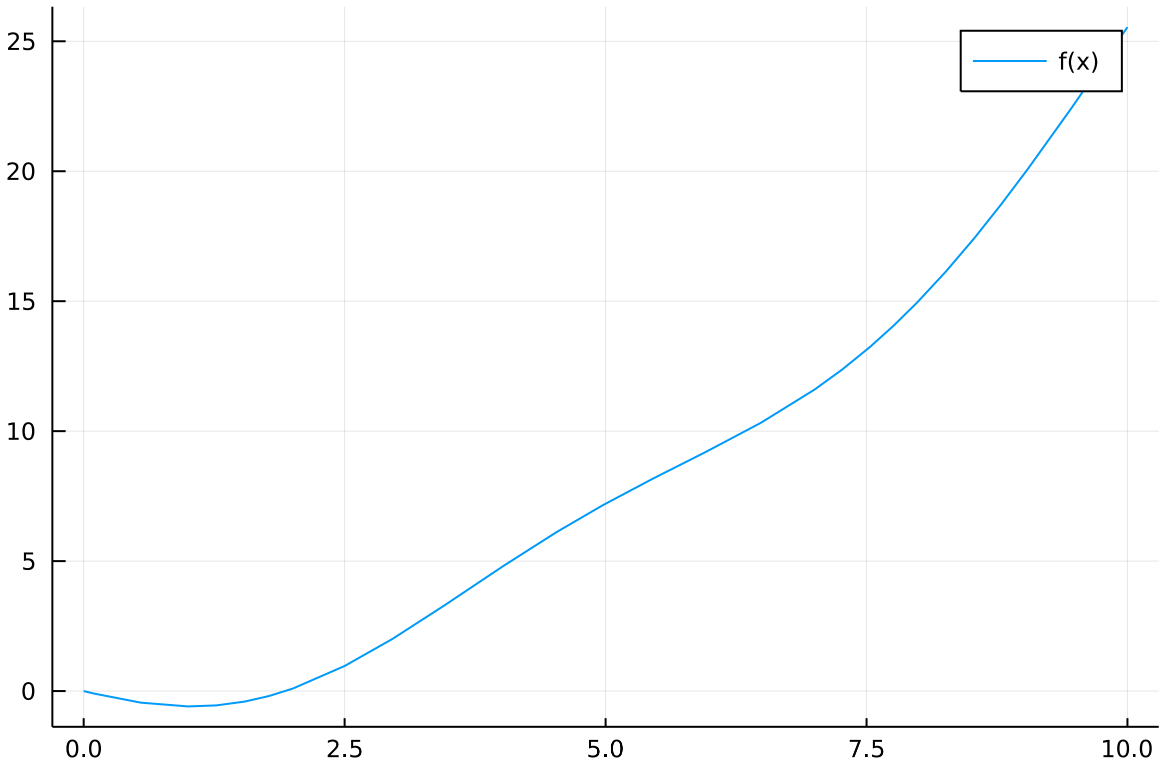
Exemplo: Método da Bisseção



Plotando a função $f(x)$

In [2]:
```julia
using Plots

f(x) = (x/2)^2 - sin(x)
plot(f, 0, 10, label="f(x)")
```

Out[2]:



Avaliando o zero da função $f(x)$

In [3]:
```
raiz, iteracao, erro = metodo_bissecao(f, 0.25, 7.5, 1e-3, 100);
```

In [4]:
```
println("A raiz da função é: ", raiz, ", onde f(c) = ",f(raiz))
println("Iterações: ", iteracao)
println("Erro: ", erro)
```

```
A raiz da função é: 1.93328857421875, onde f(c) = -0.000614785670079887
Iterações: 12
Erro: 0.0009155485398579321
```

Sistema de Equações Lineares: Método de Gauss-Seidel

- A fórmula de recorrência geral do método de **Gauss-Seidel** torna-se:

$$x_i^{(k+1)} = \frac{\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}\right)}{a_{ii}}, i = 1, \dots, n$$

- Façam 2 iterações de **G-S** para o seguinte sistema:

$$\begin{cases} 3x_1 - 0.1x_2 - 0.2x_3 = 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 = -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 = 71.4 \end{cases} \qquad usando\ x^{(0)} = [0 \quad 0 \quad 0]^T$$

Dados de entrada

Chute inicial

$x^{(0)} = [0, 0, 0]$

Tolerância

$\varepsilon_s = 0.01$

Primeira iteração

$x_1^{(1)} = \frac{7.85 + 0.1x_2^{(0)} + 0.2x_3^{(0)}}{3} = \frac{7.85+0+0}{3} = 2.6167$

$x_2^{(1)} = \frac{-19.3 - 0.1x_1^{(1)} + 0.3x_3^{(0)}}{7} = \frac{-19.3 - 0.1 \cdot 2.6167 + 0}{7} = -2.7945$

$x_3^{(1)} = \frac{71.4 + 0.3x_1^{(1)} + 0.2x_2^{(1)}}{10} = \frac{71.4 + 0.3 \cdot 2.6167 + 0.2 \cdot (-2.7945)}{10} = 7.1626$

Verificação do erro

$\varepsilon_a = \frac{||x^{(k+1)} - x^{(k)}||}{||x^{(k+1)}||}$

Na primeira iteração:

$$\varepsilon_a = \frac{||x_1^{(1)} - x^{(0)}||}{||x^{(1)}||}$$

$$\varepsilon_a^{(1)} = \frac{||(2.6167, -2.7945, 7.1626) - (0,0,0)||}{||(2.6167, -2.7945, 7.1626)||}$$

$$\varepsilon_a^{(1)} = \frac{||(2.6167, -2.7945, 7.1626)||}{||(2.6167, -2.7945, 7.1626)||} = 1$$

$$\varepsilon_a^{(1)} > \varepsilon_s$$

Segunda iteração

$$x_1^{(2)} = \frac{7.85 + 0.1x_2^{(1)} + 0.2x_3^{(1)}}{3} = \frac{7.85 + 0.1(-2.7945) + 0.2 \cdot 7.1626}{3} = 3.001$$

$$x_2^{(2)} = \frac{-19.3 - 0.1x_1^{(2)} + 0.3x_3^{(1)}}{7} = \frac{-19.3 - 0.1 \cdot 3.001 + 0.3 \cdot 7.1626}{7} = -2.4930$$

$$x_3^{(2)} = \frac{71.4 + 0.3x_1^{(2)} + 0.2x_2^{(2)}}{10} = \frac{71.4 + 0.3 \cdot 3.001 + 0.2(-2.4930)}{10} = 7.1802$$

Verificação do erro

$$\varepsilon_a = \frac{||x^{(k+1)} - x^{(k)}||}{||x^{(k+1)}||}$$

Na segunda iteração:

$$\varepsilon_a^{(2)} = \frac{||x^{(2)} - x^{(1)}||}{||x^{(2)}||}$$

$$\varepsilon_a^{(2)} = \frac{||(3.001, -2.4930, 7.1802) - (2.6167, -2.7945, 7.1626)||}{||(3.001, -2.4930, 7.1802)||} = 0.0598$$

$$\varepsilon_a^{(2)} > \varepsilon_s$$

Cálculos auxiliares

In [5]:
```
7.85/3
```
Out[5]: 2.6166666666666667

In [6]:
```
(-19.3-0.1*2.6167+0)/7
```
Out[6]: -2.7945242857142856

In [7]:
```
(71.4 + 0.3*2.6167 + 0.2*(-2.7945))/10
```
Out[7]: 7.162611000000001

In [8]:
```
(7.85+0.1*(-2.7945)+0.2*7.1626)/3
```
Out[8]: 3.001023333333334

In [9]:
```
(-19.3-0.1*3.001+0.3*7.1626)/7
```

Out[9]: -2.4930457142857145

In [10]:
```
(71.4+0.3*3.001+0.2*(-2.4930))/10
```

Out[10]: 7.180170000000001

In [11]:
```
x₂ = [3.001 -2.4930 7.1802]
x₁ = [2.6167 -2.7945 7.1626]

using LinearAlgebra
norm(x₂-x₁, 2)/norm(x₂, 2)
```

Out[11]: 0.05981298325293443

Sistema de Equações Não Lineares: Método de Newton-Raphson

Métodos de solução: Newton-Raphson

- O método mais utilizado para solução de SENL é o **Método de Newton-Raphson**, que já vimos anteriormente em zeros de funções.
- Nessa ocasião vimos que o método de NR poderia ser obtido a partir de uma expansão em série de Taylor de primeira ordem:

$$f\left(x^{(k+1)}\right) \cong f\left(x^{(k)}\right) + \left(x^{(k+1)} - x^{(k)}\right) f'\left(x^{(k)}\right) = 0$$

- Mas agora, sendo a função $f(x_{k+1})$ uma função vetorial de várias variáveis $\left(F\left(x^{(k+1)}\right)\right)$
- Para 2 equações:

$$\begin{aligned} f_1^{(k)} + \left(\left(x_1^{(k+1)} - x_1^{(k)}\right)\frac{\partial f_1}{\partial x_1} + \left(x_2^{(k+1)} - x_2^{(k)}\right)\frac{\partial f_1}{\partial x_2}\right) = 0 \\ f_2^{(k)} + \left(\left(x_1^{(k+1)} - x_1^{(k)}\right)\frac{\partial f_2}{\partial x_1} + \left(x_2^{(k+1)} - x_2^{(k)}\right)\frac{\partial f_2}{\partial x_2}\right) = 0 \end{aligned} \quad \rightarrow \quad \begin{bmatrix} f_1^{(k)} \\ f_2^{(k)} \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} - x_1^{(k)} \\ x_2^{(k+1)} - x_2^{(k)} \end{bmatrix} = 0$$

Métodos de solução: Newton-Raphson

- Utilizando a nomenclatura matricial:

$$\begin{bmatrix} f_1^{(k)} \\ f_2^{(k)} \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} - x_1^{(k)} \\ x_2^{(k+1)} - x_2^{(k)} \end{bmatrix} = 0 \quad \rightarrow \quad F\left(x^{(k)}\right) + J\left(x^{(k)}\right) dx^{(k+1)} = 0$$

$-J\left(x^{(k)}\right) dx^{(k+1)} = F\left(x^{(k)}\right)$ Temos um Sistema de Equações Lineares: $Ax = b$

$dx^{(k+1)} = -\left(J\left(x^{(k)}\right)\right)^{-1} F\left(x^{(k)}\right)$ Precisamos determinar o valor de $dx^{(k+1)}$

onde: $dx^{(k+1)} = x^{(k+1)} - x^{(k)}$ e $J\left(x^{(k)}\right)$ é chamada de Jacobiana de $F\left(x^{(k)}\right)$

- No processo iterativo de NR, calculam-se os valores de $x^{(k+1)}$ até que os critérios de parada sejam atingidos:

$$k \leq itemax \quad \text{ou} \quad \varepsilon_a \leq tol\_admitida$$

sendo $itemax$ o número máximo de iterações e $tol\_admitida$ a tolerância admitida para o erro

Implementação computacional

In [12]:
```
using LinearAlgebra

function newtonraphson(F, J, x, ε=1e-4, nitemax=200)

    nite = 0
    while norm(F(x))>ε && nite<nitemax
```

```julia
        dx = -J(x) \ F(x)
        x = x + dx

        nite = nite + 1
    end

    err = norm(F(x))
    return x, nite, err
end;
```

Exemplo: Newton-Raphson

Métodos de solução: Newton-Raphson

- Vamos ver um exemplo. Dado o seguinte sistema não linear:
- $F(x) = \begin{bmatrix} x_1^2 + x_1 x_2 - 10 \\ x_2 + 3x_1 x_2^2 - 57 \end{bmatrix}$
- A matriz Jacobiana desse sistema será:
- $J(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 & x_1 \\ 3x_2^2 & 1 + 6x_1 x_2 \end{bmatrix}$
- A Jacobiana calculada para $x^{(0)} = [1.5 \quad 3.5]^T$:
- $J(x^{(0)}) = \begin{bmatrix} 2(1.5) + (3.5) & (1.5) \\ 3(3.5)^2 & 1 + 6(1.5)(3.5) \end{bmatrix}$
- $J^{(0)} = \begin{bmatrix} 6.5 & 1.5 \\ 36.75 & 32.5 \end{bmatrix}$

Se tiver dúvidas sobre o cálculo analítico da Jacobiana, sugiro o estudo dos capítulos sobre derivadas parciais dos livros de cálculo.

In [13]:

```julia
# Sistema de equações não lineares
F(x) = [ x[1]^2 + x[1]*x[2] - 10, x[2] + 3*x[1]*x[2]^2 - 57]

# Matriz Jacobiana
using ForwardDiff
J(x) = ForwardDiff.jacobian(F, x)

# Estimativa inicial
x0 = [1.5, 3.5];

# Verificação da Jacobiana avaliada em x0
J(x0)
```

Out[13]:

```
2×2 Array{Float64,2}:
  6.5    1.5
 36.75  32.5
```

Avaliação com **tolerância** e **número máximo de iterações** padrão **(1e-4, 200)**

In [14]:

```julia
x, nite, err = newtonraphson(F, J, x0);
```

In [15]:

```julia
println("x: ", x)
println("nite: ", nite)
println("err: ", err)
```

```
x: [1.99999998387626, 2.999999413388913]
nite: 3
err: 2.2177271312785038e-5
```

Avaliação com **tolerância = 1e-6** e **número máximo de iterações = 2**

In [16]:

```julia
x, nite, err = newtonraphson(F, J, x0, 1e-6, 2);
```

In [17]:

```julia
println("x: ", x)
println("nite: ", nite)
println("err: ", err)
```

```
x: [1.9987006090558244, 3.002288562924508]
nite: 2
err: 0.04977678561725888
```

Avaliação com **tolerância = 1e-6** e **número máximo de iterações padrão = 200**

In [18]:

```julia
x, nite, err = newtonraphson(F, J, x0, 1e-6);
```

In [19]:

```julia
println("x: ", x)
println("nite: ", nite)
println("err: ", err)
```

```
x: [1.99999999999998, 3.000000000000075]
nite: 4
err: 2.2382349940267364e-12
```

Verificação da solução encontrada

In [20]:

```julia
F(x)
```

Out[20]:

```
2-element Array{Float64,1}:
 1.0658141036401503e-14
 2.2382096176443156e-12
```