Métodos Numéricos para Engenharia

MÓDULO 3 - RAÍZES DE EQUAÇÕES – MÉTODOS ABERTOS

PROFESSOR LUCIANO NEVES DA FONSECA

Métodos Abertos

Os métodos abertos são adequados para o cálculo de tanto raízes reais como complexas conjugadas, tanto simples como múltiplas

- Seja uma função não-linear $y = f(x)$
- A sua expansão por série de Taylor em torno do ponto x_0 será da por:

$$f(x) = f(x_o) \quad + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots.$$

- Se pararmos na expansão linear da função teremos:

$$f(x) \approx f(x_o) \quad + f'(x_0)(x - x_0)$$

- Na raiz $x = x_1$ da função sabemos que $f(x_1) = 0$ então:

$$f(x_1) = 0$$

$$f(x_o) + f'(x_0)(x_1 - x_0) \approx 0$$

$$\boxed{x_1 \approx x_0 - \frac{f(x_0)}{f'(x_0)}}$$

Esta equação pode ser usada como uma equação de recorrência. Começamos com uma valor inicial x_0 e calculamos a raiz x_1. Fazemos então $x_0 = x_1$ e recalculamos a nova estimativa para a raiz x_1, e assim sucessivamente até atingirmos a tolerância desejada.

Métodos Abertos - Newton Raphson

Algoritmo:

- Escolha uma estimativa inicial x_0 para a raiz da função $f(x)$
- Defina uma tolerância (precisão) desejada

1 – Faça uma estimativa para a raiz da equação

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

2 – Se $(x_1 <> 0)$ erro = $|(x_1 - x_0)/x_1|$

3 – Se erro $<$ tolerância

Raiz $= x_1$

Pare

4 – $x_o = x_1$

5- Repita passo 1

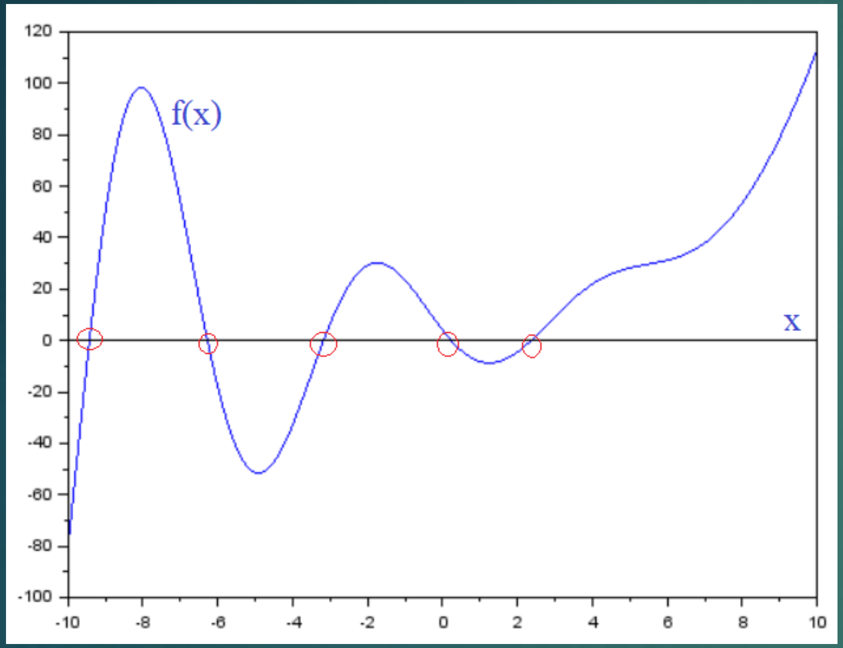
```
function x1=NewtonRaphson(f,df,x0,tol,prt)
  erro = 1;
  if (prt)
     if(imag(x0)<>0) printf ( '%i\t%.10f+%.10fi\t%.1e\n',1,real(x0),imag(x0),erro)
     else            printf ( '%i\t%.10f\t%.1e\n',1,x0,erro)  end
  end
  for (k=2:100)
     x1 = x0 - f(x0)/df(x0)
     if (x1<>0) erro =abs((x1-x0)/x1) end
     if (prt)
        if(imag(x1)<>0)  printf ('%i\t%.10f+%.10fi\t%.1e\n',k,real(x1),imag(x1),erro)
        else             printf ( '%i\t%.10f\t%.1e\n',k,x0,erro)   end
     end
     if  ( (erro<tol) | (f(x1)==0) )  break  end
     x0=x1
  end
endfunction
```

$$E(i+1) = \frac{-f''(x_r)}{2f'(x_r)} E(i)^2$$

→ Erro diminui de forma quadrática a cada iteração

- Notar que o algoritmo não exige um intervalo inicial [a,b] que contenha a raiz no qual f(a)f(b)<0
- Sendo assim, pode ser utilizado com raízes complexa e, com algumas ressalvas, com em raízes múltiplas

Exemplo $y = 4e^{\frac{x}{3}} - 20\mathrm{e}^{-\left(\frac{x}{5}\right)}\sin(x)$

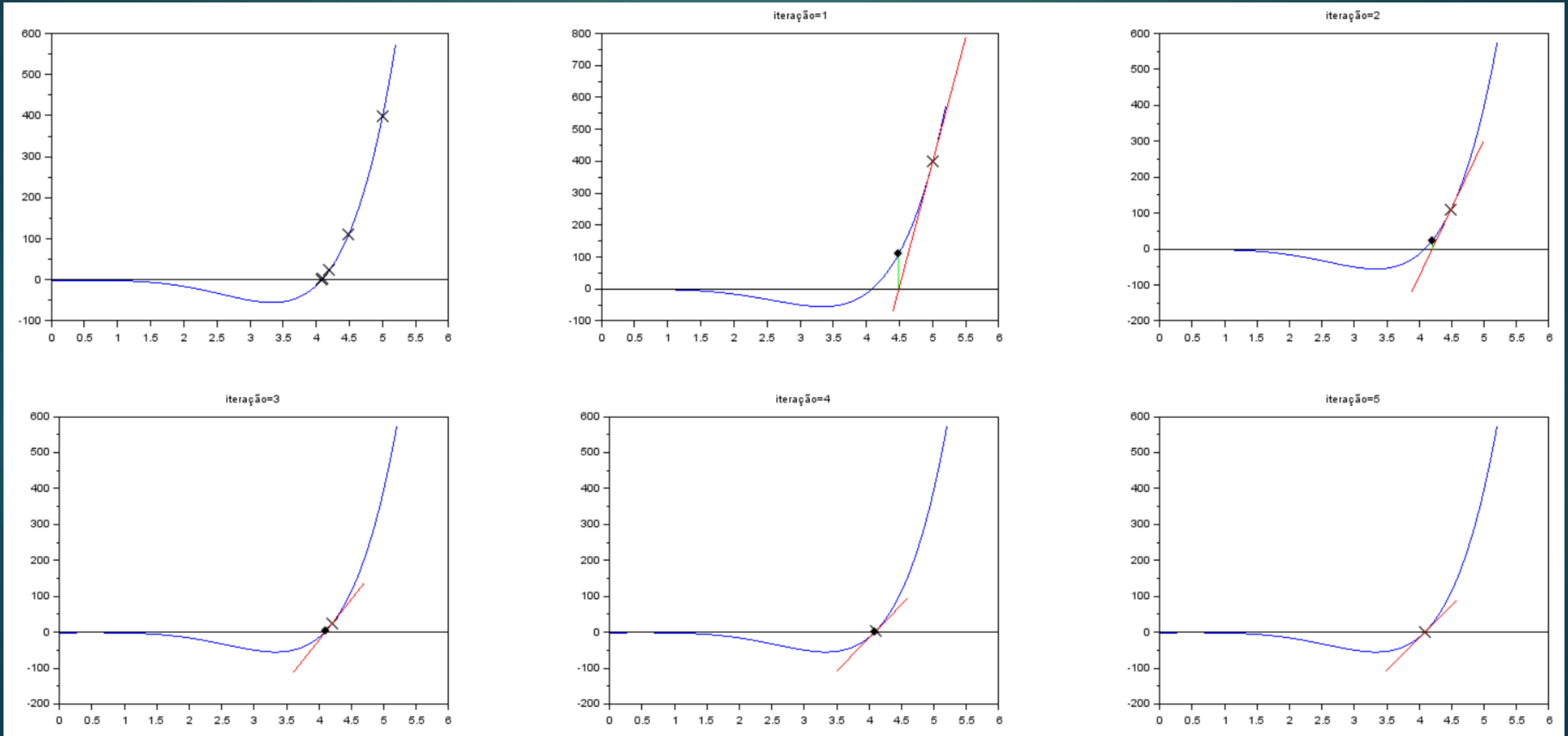



```
--> deff ('y=f(x)' , 'y=4*exp(x/3)-20*exp(-x/5).*sin(x)')

--> deff ('y=df(x)' , 'y=4/3*exp(x/3)+4*exp(-x/5).*sin(x)-20*exp(-x/5).*cos(x)')

--> xr=NewtonRaphson(f,df,2,1e-15,%t)
1       2.0000000000     1.0e+00
2       2.0000000000     1.7e-01
3       2.4144932997     2.3e-02
4       2.3608200577     2.4e-04
5       2.3602450897     3.0e-08
6       2.3602450181     5.6e-16
 xr  =

   2.3602450180995
```

- Notar a convergência quadrática de Newton Raphson , alcançando uma tolerância de 10^{-16} em apenas 6 iterações
- A convergência do método não é garantida e depende da escolha de x_1.
- Exercício:
- Usar os dois métodos para encontrar as outras raízes da função.

- O método de Newton Raphson usa linhas tangenciais que passam por aproximações sucessivas da raiz
- O método precisa de uma boa estimativa inicial, caso contrário a solução pode divergir, ou convergir para uma raiz que não seja relevante.
- O método diverge ser a derivada de uma raiz intermediária for zero.
- Caso o método funcione, sua taxa de convergência é muito alta.

```
--> deff ('y=f(x)','y=x^5-6*x^4+10*x^3-10*x^2+5*x-2')

--> exemploNewtonRaphson(f)
1       5.0000000000    1.0e+00
2       4.4897435898    1.1e-01
3       4.1993115371    6.9e-02
4       4.0973134180    2.5e-02
5       4.0853313209    2.9e-03

raiz = 4.085331 após 5 iterações
```



Raízes Complexas Conjugadas

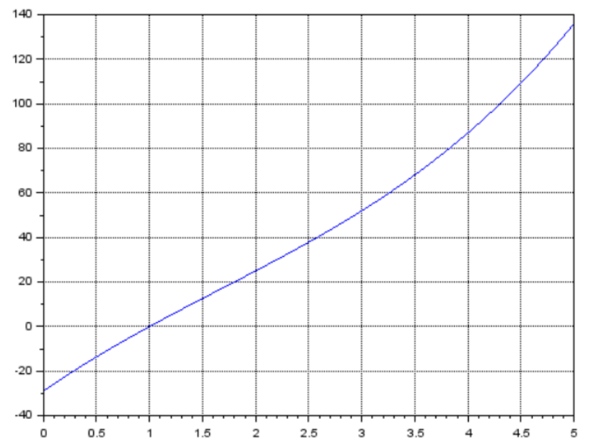
-Podemos utilizar o mesmo algoritmo, porém com estimativa inicial complexa

```
--> deff('y=f(x)','y=x^3-5*x^2+33*x-29');

--> deff('y=df(x)','y=3*x^2-10*x+33');

--> xr=NewtonRaphson(f,df,10,1e-14,%t)
1        10.0000000000    1.0e+00
2        10.0000000000    5.2e-01
3        6.5622317597     6.7e-01
4        3.9233427090     1.2e+00
5        1.8229812468     8.4e-01
6        0.9903076298     9.7e-03
7        0.9999927146     7.3e-06
8        1.0000000000     4.1e-12
9        1.0000000000     2.2e-16
 xr  =

   1.
```

```
--> xr=NewtonRaphson(f,df,10+10*%i,1e-14,%t)
1        10.0000000000+10.0000000000i    1.0e+00
2        6.9396201800+7.0120908959i      4.3e-01
3        4.7584545845+5.2163199480i      4.0e-01
4        3.0992655307+4.3597087756i      3.5e-01
5        1.8416284902+4.5272055216i      2.6e-01
6        2.0506821988+5.0797121476i      1.1e-01
7        2.0021086245+5.0013988971i      1.7e-02
8        2.0000018192+4.9999995012i      4.7e-04
9        2.0000000000+5.0000000000i      3.5e-07
10       2.0000000000+5.0000000000i      1.9e-13
11       2.0000000000+5.0000000000i      0.0e+00
 xr  =

   2. + 5.i
```

```
--> deff ('y=f(x)' , 'y=x^3-5*x^2+33*x-29')

--> x=linspace(0,5,1000);

--> plot(x,f(x))

--> xgrid()
```

Raízes
1
2+j5
2-j5

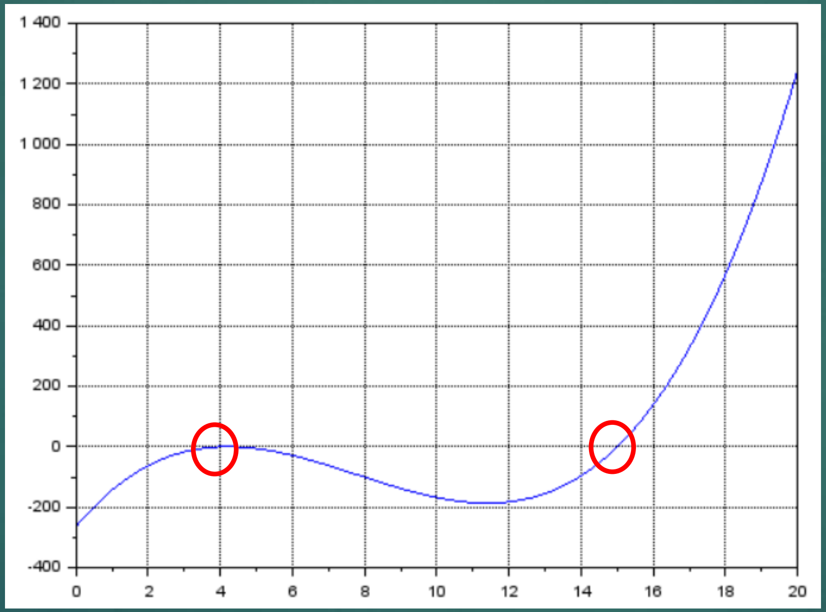


Raízes Múltiplas
Exemplo : $y = x^3 - 23.4\,x^2 + 143.64\,x - 264.6$

```
--> deff ('y=f(x)' , 'y=x^3-23.4*x^2+143.64*x-264.6')

--> x=[0.5:20.5];

--> [x' f(x)']
 ans  =

   0.5    -198.505
   1.5    -98.415
   2.5    -36.125
   3.5    -5.635
   4.5    -0.945
   5.5    -16.055
   6.5    -44.965
   7.5    -81.675
   8.5    -120.185
   9.5    -154.495
   10.5   -178.605
   11.5   -186.515
   12.5   -172.225
   13.5   -129.735
   14.5   -53.045
   15.5    63.845
   16.5    226.935
   17.5    442.225
   18.5    715.715
   19.5    1053.405
   20.5    1461.295
```

Notar que na raiz dupla não há inversão de sinal!!!

```
--> deff ('y=f(x)' , 'y=x^3-23.4*x^2+143.64*x-264.6')

--> x=linspace(0,20,1000);

--> plot(x,f(x))

--> xgrid()
```




```
--> deff("y=f(x)","y=x^3-23.4*x^2+143.64*x-264.6")

--> deff("y=df(x)","y=3*x^2-46.8*x+143.64")

--> xr=NewtonRaphson(f,df,4,1e-7,%t)
1        4.0000000000     1.0e+00
2        4.0000000000     2.4e-02
3        4.0990990991     1.2e-02
4        4.1493171355     6.1e-03
5        4.1745995215     3.0e-03
6        4.1872848784     1.5e-03
7        4.1936387034     7.6e-04
8        4.1968184158     3.8e-04
9        4.1984089737     1.9e-04
10       4.1992044283     9.5e-05
11       4.1996021995     4.7e-05
12       4.1998010961     2.4e-05
13       4.1999005471     1.2e-05
14       4.1999502734     5.9e-06
15       4.1999751367     3.0e-06
16       4.1999875684     1.5e-06
17       4.1999937846     7.4e-07
18       4.1999968929     3.7e-07
19       4.1999984479     1.9e-07
20       4.1999992261     9.3e-08
 xr  =

   4.200
```

- Notar que nenhum método intervalar converge para a raiz dupla, pois não há inversão de sinal
- No entanto, o método de Newton-Raphson converge, mas a convergência é lenta.
- O erro não diminui de forma quadrática, diminui de forma linear (como nos métodos intervalares)

Modificação do Algoritmo de Newton Raphson para Raízes Múltiplas

$$\cancel{x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}}$$

```
function x1=NewtonRaphson_p(f,df,x0,tol,p,prt)
  erro = 1;
  if (prt)
    if(imag(x0)<>0)    printf ( '%i\t%.10f+%.10fi\t%.1e\n',1,real(x0),imag(x0),erro)
    else               printf ( '%i\t%.10f\t%.1e\n',1,x0,erro)  end
  end
  for (k=2:100)
    x1 = x0 - p*f(x0)/df(x0)
    if(x1<>0)  erro =abs((x1-x0)/x1)  end
    if (prt)
      if(imag(x1)<>0)   printf ('%i\t%.10f+%.10fi\t%.1e\n',k,real(x1),imag(x1),erro)
      else              printf ( '%i\t%.10f\t%.1e\n',k,x0,erro)   end
    end
    if  ( (erro<tol) | (f(x1)==0) )  break  end
    x0=x1
  end
endfunction
```

```
--> deff("y=f(x)","y=x^3-23.4*x^2+143.64*x-264.6")

--> deff("y=df(x)","y=3*x^2-46.8*x+143.64")

--> xr=NewtonRaphson(f,df,4,1e-7,%t)
1       4.0000000000    1.0e+00
2       4.0000000000    2.4e-02
3       4.0990990991    1.2e-02
4       4.1493171355    6.1e-03
5       4.1745995215    3.0e-03
6       4.1872848784    1.5e-03
7       4.1936387034    7.6e-04
8       4.1968184158    3.8e-04
9       4.1984089737    1.9e-04
10      4.1992044283    9.5e-05
11      4.1996021995    4.7e-05
12      4.1998010961    2.4e-05
13      4.1999005471    1.2e-05
14      4.1999502734    5.9e-06
15      4.1999751367    3.0e-06
16      4.1999875684    1.5e-06
17      4.1999937846    7.4e-07
18      4.1999968929    3.7e-07
19      4.1999984479    1.9e-07
20      4.1999992261    9.3e-08
 xr  =

   4.200
```

O algoritmo de Newton Raphson encontra a raiz dupla, mas com convergência linear

Com a multiplicidade correta p=2, a convergência volta a ser quadrática.

$$x_1 = x_0 - p\,\frac{f(x_0)}{f'(x_0)}$$

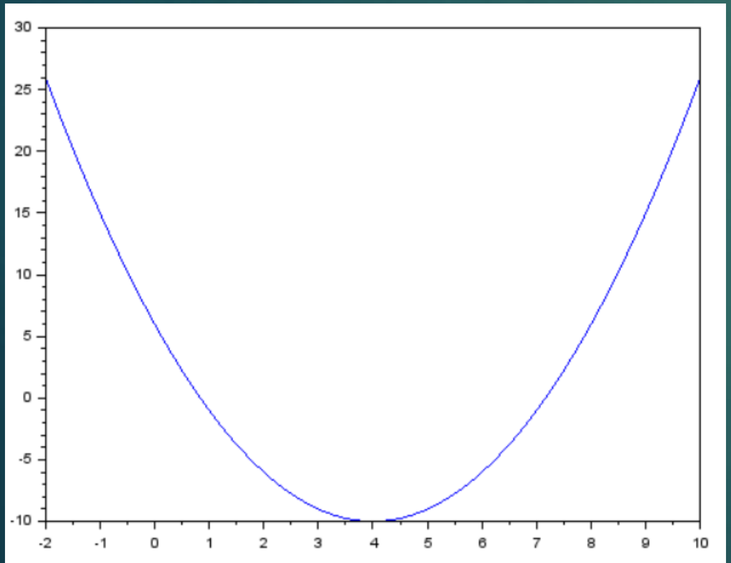
```
--> xr=NewtonRaphson_p(f,df,4,1e-7,2,%t)
1       4.0000000000    1.0e+00
2       4.0000000000    4.7e-02
3       4.1981981982    4.3e-04
4       4.1999998497    4.2e-08
 xr  =

   4.200
```

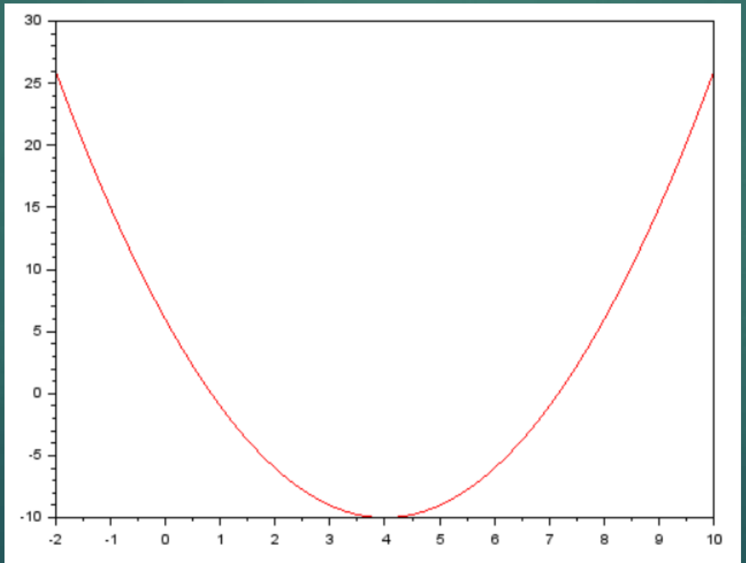
Polinômios podem ser representados de maneira especial no Scilab

```
--> deff ('y=f(x)' , 'y=x^2-8*x+6')

--> x=linspace(-2,10,1000);

--> plot(x,f(x))
```

```
--> ps=poly([6,-8,1],'s','coeff')
 ps  =

  6 -8s +s²

--> x=linspace(-2,10,1001);

--> y=horner(ps,x);

--> plot(x,y,'r')
```

```
--> s=poly(0,'s');

--> ps=s^2-8*s+6
 ps  =

  6 -8s +s²

--> x=linspace(-2,10,2001);

--> y=horner(ps,x);

--> plot(x,y,'r')
```





Podemos facilmente calcular a derivada literal de um polinômio:

```
function dps=derivat_fga(ps)
    a = coeff(ps)
    N=length(a)
    b= a(2:N).*[1:N-1]
    dps=poly(b,"s","coeff")
endfunction
```

```
--> ps
 ps  =

  6 -8s +s²

--> dps=derivat_fga(ps)
 dps  =

  -8 +2s
```

Modificação do Algoritmo de Newton Raphson para Polinômios

```
function x1=NewtonRaphson_pol(ps,x0,prt)
  erro = 1;
   if (prt)
       if(imag(x0)<>0)    printf ( '%i\t%.10f+%.10fi\t%.1e\n',1,real(x0),imag(x0),erro)
       else               printf ( '%i\t%.10f\t%.1e\n',1,x0,erro)  end
   end
  for (k=2:100)
     f0=horner(ps,x0)
     df0=horner(derivat_fga(ps),x0)
     x1=x0-f0/df0
     if (x1<>0) erro =abs((x1-x0)/x1) end
      if (prt)
         if(imag(x1)<>0)    printf ('%i\t%.10f+%.10fi\t%.1e\n',k,real(x1),imag(x1),erro)
          else              printf ( '%i\t%.10f\t%.1e\n',k,x0,erro)   end
      end
     if  ( (erro<1e-16) | (f(x1)==0) )  break  end
     x0=x1
  end
  x1=clean(x1,1e-10)
endfunction
```

- ps é o polinômio para a qual se que calcular a raiz
- A derivada analítica do polinômio é encontrada diretamente pelo programa derivat_fga()
- x_0 é uma estimativa inicial para a raiz
- A tolerância está fixada 10^{-16}

```
--> deff('y=f(x)','y=x^2-8*x+6');

--> deff('y=df(x)','y=2*x-8');

--> xr=NewtonRaphson(f,df,0.1,1e-16,%t)
1       0.1000000000    1.0e+00
2       0.1000000000    8.7e-01
3       0.7679487179    8.2e-02
4       0.8369692023    9.0e-04
5       0.8377222502    1.1e-07
6       0.8377223398    1.5e-15
7       0.8377223398    1.3e-16
8       0.8377223398    0.0e+00
 xr  =

   0.8377223
```

```
--> ps=poly([6,-8,1],'s','coeff')
 ps  =

  6 -8s +s²

--> xr=NewtonRaphson_pol(ps,0.1,%t)
1       0.1000000000    1.0e+00
2       0.1000000000    8.7e-01
3       0.7679487179    8.2e-02
4       0.8369692023    9.0e-04
5       0.8377222502    1.1e-07
6       0.8377223398    1.5e-15
7       0.8377223398    1.3e-16
8       0.8377223398    0.0e+00
 xr  =

   0.8377223
```

Método da Secante

- Seja uma função não-linear $y = f(x)$
- A equação recursiva de Newton-Raphson para encontrar uma raiz será:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad \text{(I)}$$

- Um dos problemas relacionados com o método de Newton Raphson está na necessidade de se avaliar a derivada da função.
- Algumas funções podem ter derivadas de difícil avaliação analítica.
- Neste caso, podemos substituir a derivada por uma diferença finita:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \qquad \text{(II)}$$

Substituindo (II) em (I)

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}}$$

Métodos Abertos - Método da Secante

Algoritmo:

- Escolha duas estimativas iniciais x_1 e x_2 para a raiz da função $f(x)$
- Defina uma tolerância (precisão) desejada

1 – Faça uma estimativa para a raiz da equação

$$x_o = x_1\,;\quad x_1 = x_2$$

$$x_2 = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$$

2 – Se $(x_2 > 0)$ erro = $|(x_2 - x_1)/x_2|$

3 – Se erro $<$ tolerância

Raiz $= x_2$

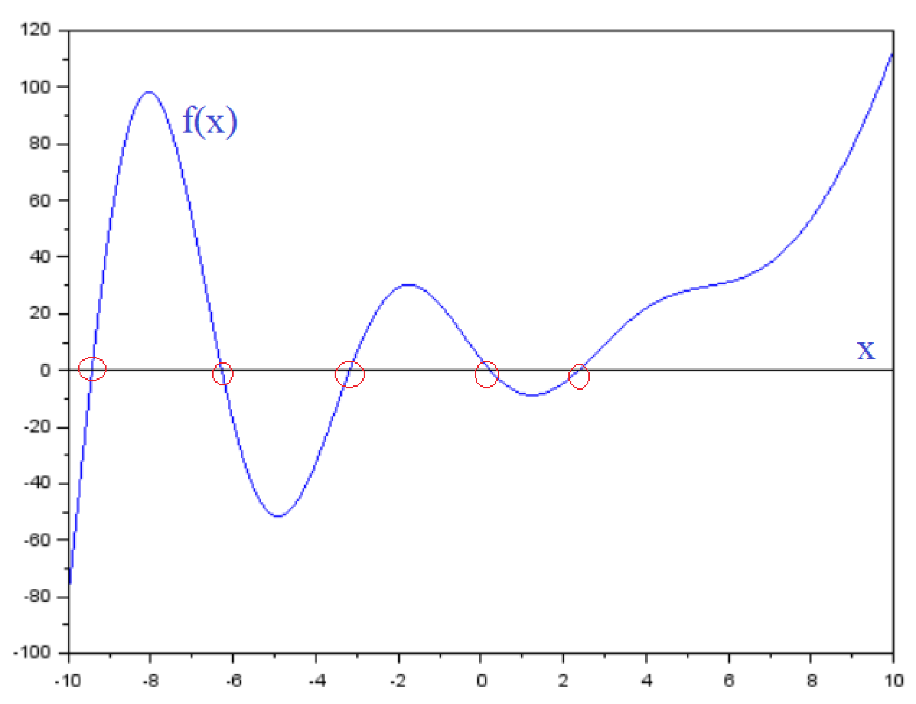
Pare

4 – Repita passo 1

```
function x1=Secante(f,x1,x2,tol,prt)
  erro = 1;
  if (prt)
    printf ( '%i\t%.10f\t%.1e\n',1,x1,erro)
    printf ( '%i\t%.10f\t%.1e\n',1,x2,erro)
  end
  for(k=2:100)
     x0=x1;
     x1=x2;
     f0=f(x0)
     f1=f(x1)
     df1 = (f1-f0)/(x1-x0);
     x2 = x1 - f1/df1;
     if(x2<>0) erro =abs((x2-x1)/x2) end
     if (prt)
        printf ( '%i\t%.10f\t%.1e\t%.1e\n',k,x2,erro,abs(x2-x1))
     end
     if ( (erro<tol) | (f(x2)==0) ) break end
  end
endfunction
```

- Notar que o algoritmo não exige um intervalo inicial [a,b] que contenha a raiz no qual f(a)f(b)<0
- Sendo assim, pode ser utilizado (com algumas ressalvas) em raízes múltiplas

Exemplo $y = 4e^{\frac{x}{3}} - 20e^{-\left(\frac{x}{5}\right)}\sin(x)$




```
--> deff('y=f(x)','y=4*exp(x/3)-20*exp(-x/5)*sin(x)');

--> deff('y=df(x)','y=4/3*exp(x/3)+4*exp(-x/5)*sin(x)-20*exp(-x/5)*cos(x)');

--> xr=NewtonRaphson(f,df,2,1e-15,%T)
1        2.0000000000     1.0e+00
2        2.4144932997     1.7e-01
3        2.3608200577     2.3e-02
4        2.3602450897     2.4e-04
5        2.3602450181     3.0e-08
6        2.3602450181     5.6e-16
 xr  =

   2.36024501810
```

```
--> deff('y=f(x)','y=4*exp(x/3)-20*exp(-x/5)*sin(x)');

--> xr=Secante(f,2,3,1e-14,%t)
1         2.0000000000     1.0e+00
1         3.0000000000     1.0e+00
2         2.3205762170     2.9e-01
3         2.3572775493     1.6e-02
4         2.3602715537     1.3e-03
5         2.3602450010     1.1e-05
6         2.3602450181     7.2e-09
7         2.3602450181     4.2e-14
8         2.3602450181     1.9e-16
 xr  =

   2.3602450
```

- Notar a convergência do método da secante não é exatamente quadrática nas estimativas iniciais, mas se torna quadrático quando as duas estimativas se aproximam.

- Tente usar os métodos estudados para encontrar as outras raízes da função.

Método da Secante Modificado

- Seja uma função não-linear $y = f(x)$
- A equação recursiva de Newton-Rapson para encontrar uma raiz será:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad \text{(I)}$$

- Um dos problemas relacionados com o método de Newton Raphson está na necessidade de se avaliar a derivada da função.
- Algumas funções podem ter derivadas de difícil avaliação analítica.
- Neste caso, podemos substituir a derivada por uma diferença finita:

$$f'(x_i) \approx \frac{f(x_i + \delta_x) - f(x_i)}{\delta_x} \qquad \text{(II)}$$

Substituindo (II) em (I)

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)\delta_x}{f(x_i + \delta_x) - f(x_i)}}$$

Métodos Abertos - Método da Secante Modificado

Algoritmo:

- Escolha uma estimativa inicial x_1 a raiz da função $f(x)$
- Defina uma tolerância (precisão) desejada
- Escolho o incremento d_x para o cálculo da derivada (10^{-8} para e 64 bits, ou 10^{-4} para 32 bits)

1 – Faça uma estimativa para a raiz da equação

$$x_o = x_1\ ;$$

$$x_1 = x_0 - \frac{f(x_0)d_x}{f(x_0+d_x)-f(x_0)}$$

2 – Se $(x_1 > 0)$ erro = $|(x_1-x_0)/x_1|$

3 – Se erro $<$ tolerância

Raiz $=x_1$

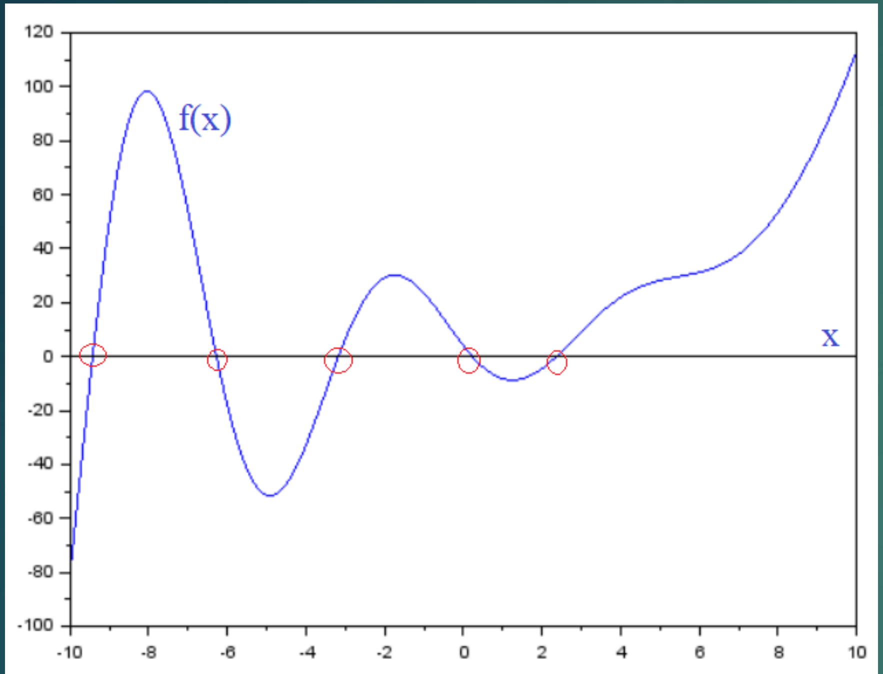
Pare

4 – Repita passo 1

```
function x1=SecanteModificado(f,x0,tol,prt)
  erro = 1;
  dx=1e-8;
  if (prt)
     if(imag(x0)<>0) printf ( '%i\t%.10f+%.10fi\t%.1e\n',1,real(x0),imag(x0),erro)
     else            printf ( '%i\t%.10f\t%.1e\n',1,x0,erro)  end
  end
  for (k=2:100)
    f0=f(x0)
    f1=f(x0+dx)
    df0=(f1-f0) /dx  // Secante Modificado
    x1 = x0 - f0/df0
    if(x1<>0)  erro =abs((x1-x0)/x1)  end
    if (prt)
      if(imag(x1)<>0) printf ('%i\t%.10f+%.10fi\t%.1e\n',k,real(x1),imag(x1),erro)
      else            printf ( '%i\t%.10f\t%.1e\n',k,x0,erro)   end
    end
    if  ( (erro<tol) | (f(x1)==0) )  break  end
    x0=x1
  end
endfunction
```

- O número de iterações necessária para a convergência depende de dx.
- Se dx for muito grande, a convergência pode ser lenta, se for muito pequeno, pode haver erros de arredondamento. Em ambos os casos pode haver divergência (assim como no método de Newton Rapshon)
- Um bom compromisso seria ser fixar δ_x em 10^{-8} para uma máquina de 64 bits, ou em 10^{-4} para uma máquina de 32 bits.

Exemplo $y = 4e^{\frac{x}{3}} - 20\mathrm{e}^{-\left(\frac{\mathrm{x}}{5}\right)}\sin(\mathrm{x})$




```
--> deff('y=f(x)','y=4*exp(x/3)-20*exp(-x/5)*sin(x)');

--> deff('y=df(x)','y=4/3*exp(x/3)+4*exp(-x/5)*sin(x)-20*exp(-x/5)*cos(x)');

--> xr=NewtonRaphson(f,df,2,1e-14,%T)
1       2.0000000000     1.0e+00
2       2.4144932997     1.7e-01
3       2.3608200577     2.3e-02
4       2.3602450897     2.4e-04
5       2.3602450181     3.0e-08
6       2.3602450181     5.6e-16
 xr  =

   2.36024501810

--> xr=SecanteModificado(f,2,1e-14,%T)
1       2.0000000000     1.0e+00
2       2.4144932901     1.7e-01
3       2.3608200574     2.3e-02
4       2.3602450897     2.4e-04
5       2.3602450181     3.0e-08
6       2.3602450181     7.5e-16
 xr  =

   2.36024501810
```

- Notar que o método da secante modificado tem desempenho comparável ao de Newton-Rapahson
- Assim como no método de Newton Raphson, só precisa de uma estimativa para a raiz (secante precisa de duas)
- No entanto, não precisamos da fórmula analítica para a derivada da função!!!!

Método do Ponto Fixo

- Seja uma função não-linear $y = f(x)$
- Na raiz temos que

$$f(x) = 0 \qquad \text{(I)}$$

- Equação (I) pode ser rearranjada na forma:

$$f(x) + x = 0 + x$$
$$g(x) = f(x) + x$$
$$x = g(x)$$

- De modo que podemos usar a seguinte equação recursiva:

$$\boxed{x_{i+1} = g(x_i)}$$

- Exemplo:

$$f(x) = \mathrm{e}^{-\left(\frac{\mathrm{x}}{5}\right)}\sin(\mathrm{x})$$

$$g(x) = x + \mathrm{e}^{-\left(\frac{\mathrm{x}}{5}\right)}\sin(\mathrm{x})$$

$$x_{i+1} = g(x_i) = x_i + \mathrm{e}^{-\left(\frac{\mathrm{x_i}}{5}\right)}\sin(\mathrm{x_i})$$

Métodos Abertos - Ponto Fixo

Algoritmo:

- Escolha uma estimativa inicial x_1 para a raiz da função $f(x)$
- Defina uma tolerância (precisão) desejada
- $g(x) = x + f(x)$

1 – Faça uma estimativa para a raiz da equação

$$x_o = x_1$$

$$x_1 = g(x_o)$$

2 – Se $(x_1 > 0)$ erro = $|(x_1 - x_0)/x_1|$

3 – Se erro $<$ tolerância

Raiz $= x_1$

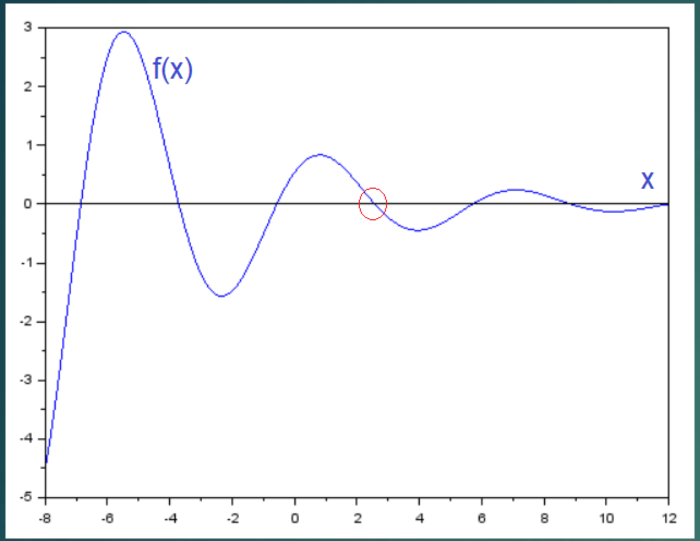
Pare

4 – Repita passo 1

```
function x1=PontoFixo(f,x0,tol,prt)
  deff ('y=g(x)','y=f(x)+x')
  erro = 1;
  if (prt)  printf ( '%i\t%.10f\t%.1e\n',0,x0,erro)  end
  for (k=1:200)
     x1 = g(x0);
     if(x1<>0)  erro =abs((x1-x0)/x1)  end
     if (prt) printf ( '%i\t%.10f\t%.1e\n',k,x1,erro)  end
     if ( (erro<tol) | (f(x1)==0) ) break end
     x0=x1
  end
endfunction
```

- Notar que o algoritmo não exige um intervalo inicial [a,b] que contenha a raiz no qual f(a)f(b)<0
- Sendo assim, pode ser utilizado (com algumas ressalvas) em raízes múltiplas

Algoritmo Scilab para o método do Ponto Fixo

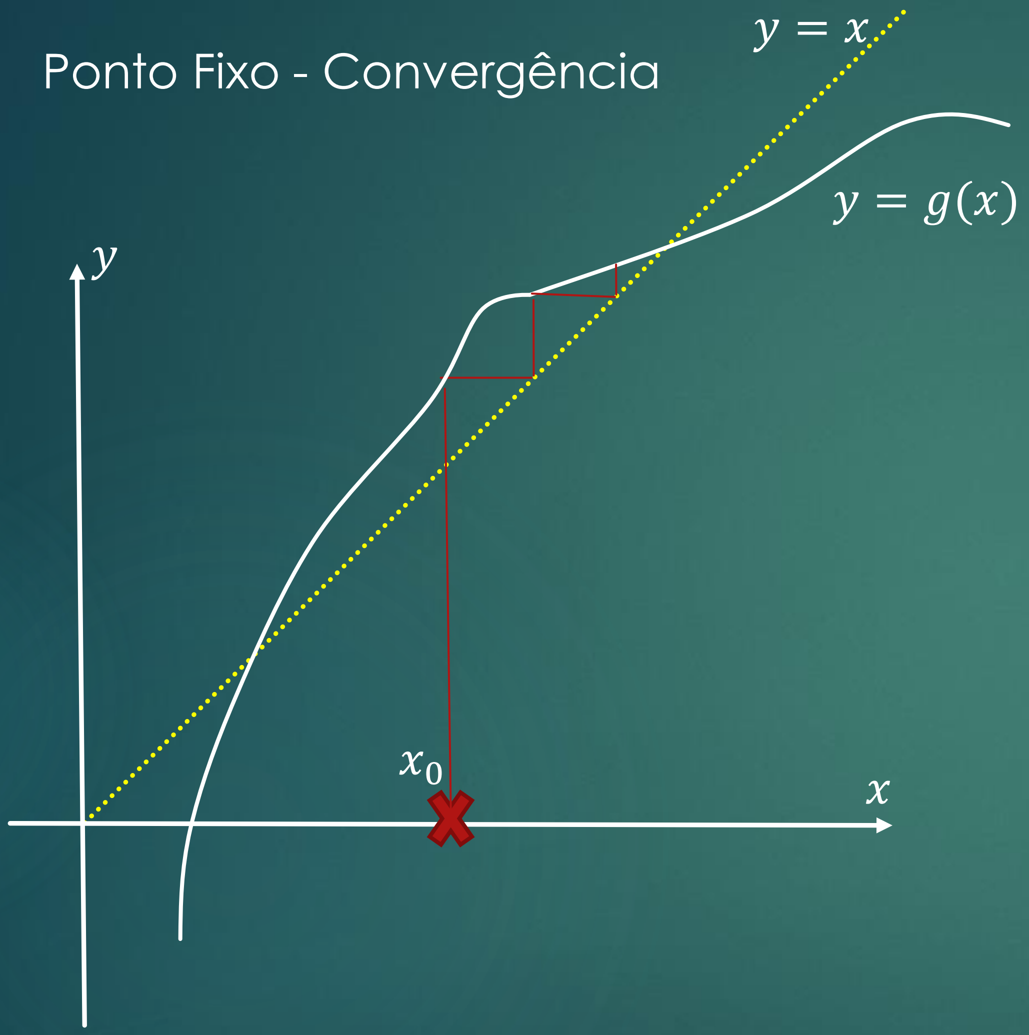


- f é a função para a qual se que calcular a raiz
- x_{ini} é a estimativa inicial da raiz

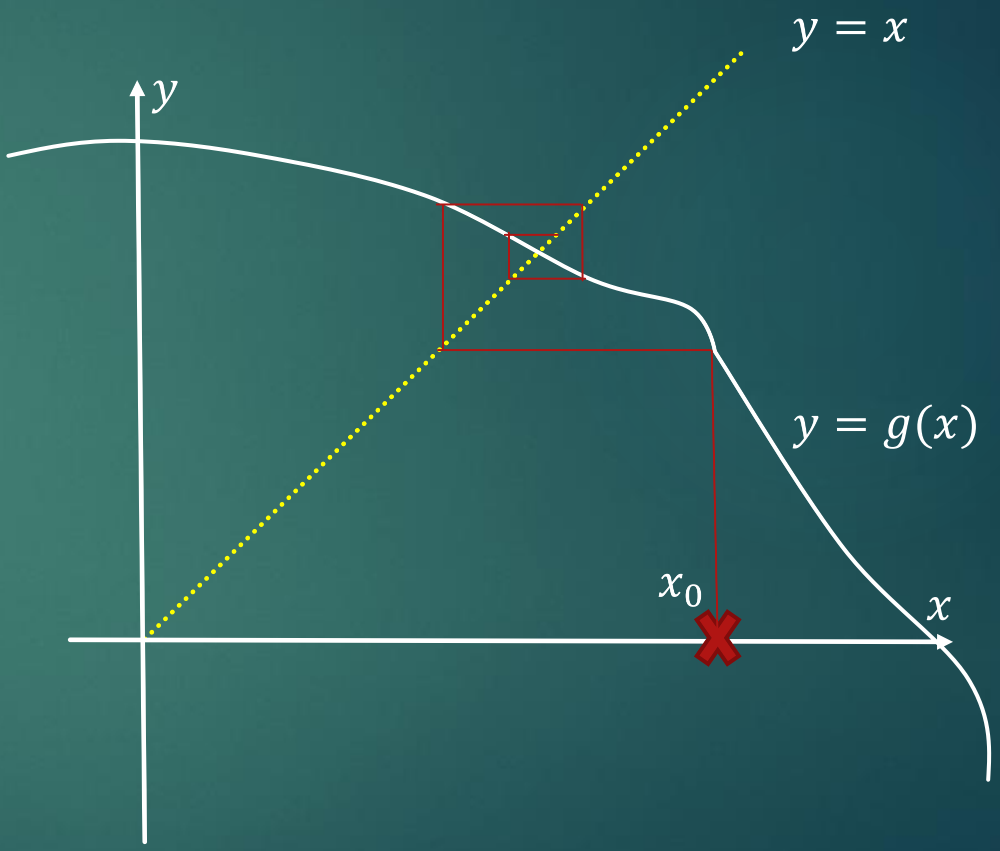
```
--> deff('y=f(x)','y=exp(-x/5)*cos(x-1)');

--> xr=SecanteModificado(f,2,1e-4,%T)
1        2.0000000000     1.0e+00
2        2.5690199200     2.2e-01
3        2.5707956978     6.9e-04
4        2.5707963268     2.4e-07
 xr  =

   2.57079632679

--> xr=PontoFixo(f,2,1e-4,%T)
0        2.0000000000     1.0e+00
1        2.3621754665     1.5e-01
2        2.4913053977     5.2e-02
3        2.5395521483     1.9e-02
4        2.5583503260     7.3e-03
5        2.5658114306     2.9e-03
6        2.5687953753     1.2e-03
7        2.5699924287     4.7e-04
8        2.5704732396     1.9e-04
9        2.5706664592     7.5e-05
 xr  =

   2.57066645922
```

- Notar que o método do Ponto Fixo tem desempenho inferior a Newton Raphson, mas com um algoritmo extremamente simpes
- Assim como no método de Newton Raphson, só precisa de uma estimativa para a raiz (secante precisa de duas)
- Há várias situações onde o método diverge, dependendo da inclinação de g(x) em relação à reta f(x)=x;

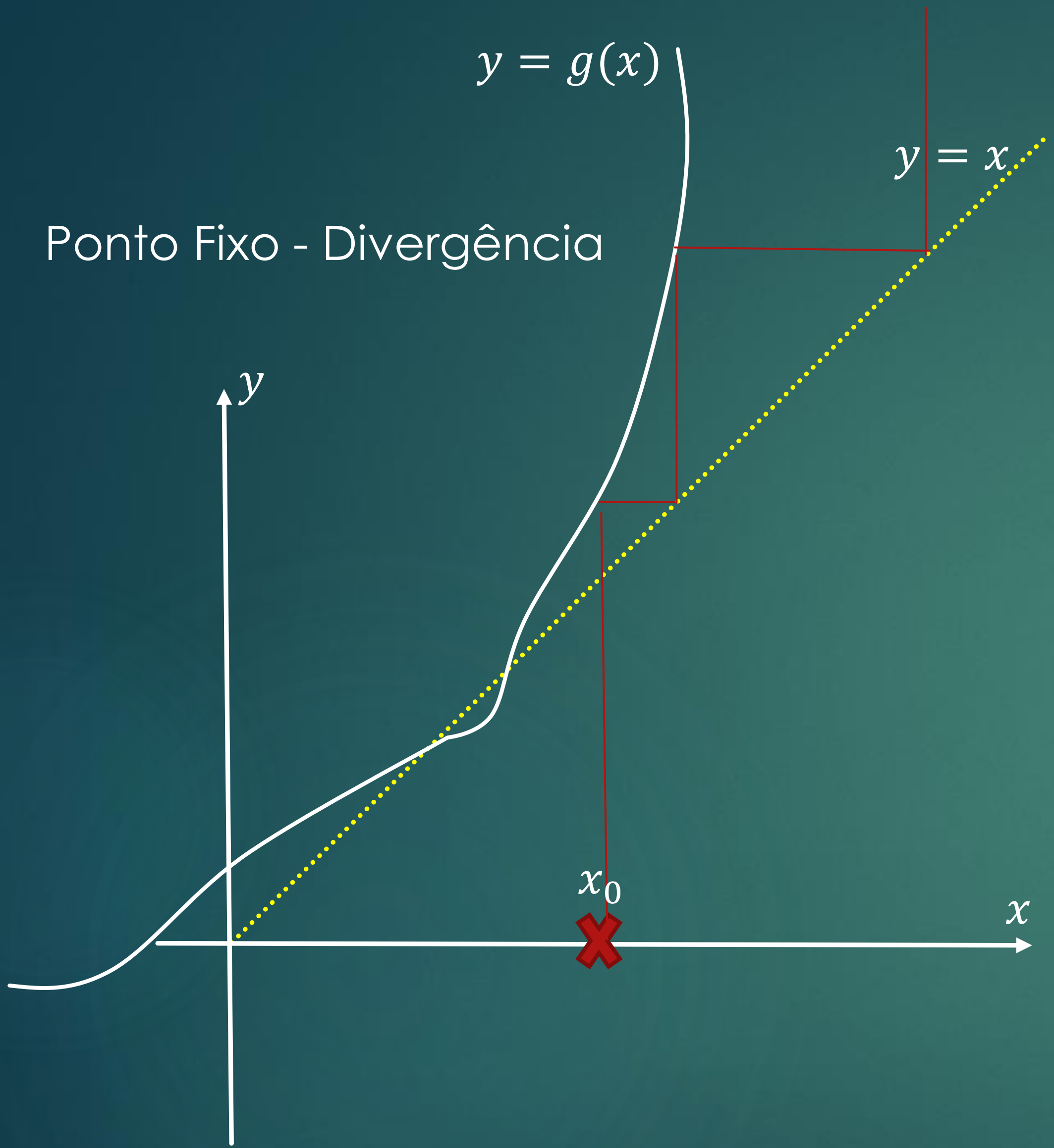
Ponto Fixo - Convergência



Ponto Fixo - Convergência



Ponto Fixo - Divergência



Ponto Fixo - Divergência