UNIVERSIDADE FEDERAL DO CEARÁ - UFC
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA
DISCIPLINA DE MÉTODOS NUMÉRICOS
SEMESTRE 2018.2

RELATÓRIO 01

Métodos: Bisseção, Posição Falsa, Ponto Fixo, Newton-Raphson, Secante.

EQUIPE:	AILSON ALEXANDRE DA SILVA MORAIS	- 397429
	GABRIEL MORAES RAMOS STUDART	- 403083
	ITALO AGUIAR DO NASCIMENTO PAULINO	- 404125
	LUCAS ESTEVES ROCHA	- 404708
CURSO:	ENGENHARIA DE COMPUTAÇÃO	
PROFESSOR(A):	CESAR LINCOLN CAVALCANTE MATTOS	

Fortaleza - CE
2018

Conteúdo

1 Método da Bisseção

O método

O método da bisseção é um método de busca de raízes em que o intervalo é reduzido dividindo-o ao meio até atingir a precisão requerida. O método admite algumas condições iniciais para funcionar corretamente: seja [a,b] o intervalo passado para o método e f(x) a função que se deseja encontrar a raiz, f(x) precisa ser contínua no intervalo [a,b], $f(a) \times f(b) < 0$, isto é, a função corta o eixo x em algum ponto no intervalo, e o intervalo deve possuir apenas uma raiz.
Como o método possui a característica de sempre dividir o intervalo ao meio, podemos calcular quantas iterações serão necessárias para atingir a precisão requerida da seguinte forma:

$$k > \frac{\log(b-a) - \log(\epsilon)}{\log(2)} \tag{1}$$

Onde k é um número inteiro.

Algoritmo

Seja f(x) contínua em [a, b] e tal que $f(a) \times f(b) < 0$.

1. Dados iniciais:
   (a) intervalo inicial [a, b]
   (b) precisão ϵ
2. Se $(b - a) < \epsilon$, então escolha para $\bar{x}$ qualquer x $\in$ [a,b]. FIM.
3. k = 1
4. M = f(a)
5. x = $\dfrac{a+b}{2}$
6. Se $M \times f(x) > 0$, faça a = x. Vá para o passo 8.
7. b = x
8. Se $(b - a) < \epsilon$, escolha para $\bar{x}$ qualquer x $\in$ [a,b]. FIM.
9. k = k + 1. Volte para o passo 5.

Implementação

```
def bissecao(f, a, b, epsilon, maxIter = 50):
    """Executa o método da bisseção para achar o zero de f no intervalo
       [a,b] com precisão epsilon. O método executa no máximo maxIter
       iterações.
       Retorna uma tupla (houveErro, raiz), onde houveErro é booleano.
    """
    ## Inicializar as variáveis Fa e Fb
    Fa = f(a)
    Fb = f(b)

    ## Teste para saber se a função muda de sinal. Se não mudar, mostrar
    ## mensagem de erro
    if (Fa * Fb) > 0:
        ## Mostrar mensagem
        print("Erro! A função não muda de sinal.")
        return (True, None)

    ## Mostra na tela cabeçalho da tabela
    print("k\t  a\t\t  fa\t\t  b\t\t  fb\t\t  x\t\t  fx\t\tintervX")

    ## Inicializa tamanho do intervalo intervX usando a função abs, x e Fx
    intervX = abs(b - a)
    x = (b + a)/2.0
    Fx = f(x)

    ## Mostra dados de inicialização
    print("-\t%e\t%e\t%e\t%e\t%e\t%e\t%e" % (a, Fa, b, Fb, x, Fx, intervX))

    ## Teste se intervalo já é do tamanho da precisão e retorna a raiz sem erros
    if(intervX <= epsilon):
        return (False, x)

    ## Iniciliza o k
    k = 0

    while k <= maxIter:
        ## Testes para saber se a raiz está entre a e x ou entre x e b e atualiza
        ## as variáveis apropriadamente

        if(f(a) * f(x) < 0):
            b = x
        else:
            a = x

        ## Atualiza intervX, x, e Fx
        intervX = abs(b - a)
        x = (b + a)/2.0
```

```
        Fx = f(x)

        ## Mostra valores na tela
        print("%d\t%e\t%e\t%e\t%e\t%e\t%e\t%e"%(k, a, Fa, b, Fb, x, Fx,
            intervX))

        ## Teste do critério de parada (usando apenas o tamanho do
            intervalo)
        if(intervX <= epsilon):
            return (False, x)

        ## Atualiza o k
        k = k+1

    ## Se chegar aqui é porque o número máximo de iterações foi atingido
    ## Mostrar uma mensagem de erro e retorna que houve erro e a última
        raiz encontrada
    print("ERRO! número máximo de iterações atingido.")
    return (True, x)
```

Testes

O método foi testado com as seguintes funções:

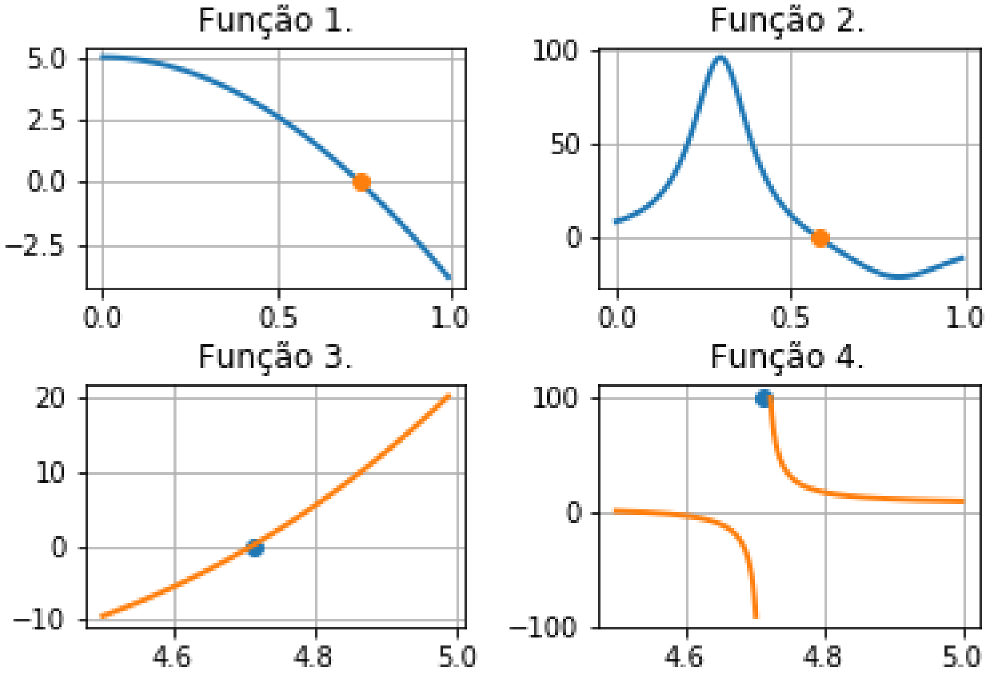
1. $f(x) = x^3 - 10x^2 + 5$
2. $f(x) = \dfrac{1}{(x-0.3)^2 + 0.01} - \dfrac{1}{(x-0.8)^2 + 0.04}$
3. $f(x) = cosh(x)cos(x)$
4. $f(x) = x - tan(x)^*$

Tabela 1: Testes para o Método da Bisseção

$f(x)$	a	b	ϵ	k	Time (s)	$\bar{x}$
1.	0	1	10^{-4}	13	$6.29925e^{-4}$	0.734588623046875
2.	0	1	10^{-2}	6	$6.32075e^{-4}$	0.58203125
3.	4	5	10^{-5}	16	$1.48355e^{-3}$	4.712390899658203
4.*	4.5	5	10^{-3}	16	$2.74667e^{-4}$	4.71240234375

* A função 4. é um caso em que o método da bisseção falha. Foi passado um intervalo em que a função é descontínua e portanto o método encontrou uma raiz errônea.

Figura 1: Comportamento das funções testadas.



2 Método da Posição Falsa

O método

O método da Posição Falsa pode aumentar a velocidade de convergência da sequência xi para a raiz ξ de uma equação f(x) = 0 reduzindo a amplitude do intervalo [a,b] usando-se um esquema diferente da bisseção Ao invés de selecionar o ponto médio de cada intervalo, esses métodos usam o ponto onde a reta secante intersecta o eixo das abscissas Se o intervalo for pequeno, essa aproximação é válida para a maioria das funções.
A equação da reta secante que passa pelos pontos de coordenadas (a, f(a)) e (b,f(b)) é dada por:

$$x0 = \frac{a(f(b)) - b(f(a))}{f(b) - f(a)} \tag{2}$$

Algoritmo

Seja f(x) contínua em [a, b] e tal que $f(a) \times f(b) < 0$.

1. Dados iniciais:
   (a) intervalo inicial [a, b]
   (b) precisão $\epsilon 1$ e $\epsilon 2$
2. Se $(b - a) < \epsilon 1$, então escolha para $\bar{x}$ qualquer x $\in$ [a,b]. FIM.
3. Se $|f(a)| < \epsilon 2$ ou se $|f(b)| < \epsilon 2$, então escolha a ou b como $\bar{x}$.FIM
4. k = 1
5. M = f(a)
6. x = $\dfrac{a(f(b)) - b(f(a))}{f(b) - f(a)}$
7. se $|f(a)| < \epsilon 2$, escolha $\bar{x} = x$.FIM
8. Se $M \times f(x) > 0$, faça a = x. Vá para o passo 10.
9. b = x
10. Se $(b - a) < \epsilon 1$, escolha para $\bar{x}$ qualquer x $\in$ (a,b). FIM.
11. k = k + 1. Volte para o passo 5.

Implementação

```
def false_pos(f, a, b, epsilon, maxIter = 50):

    """Executa o método da Posição Falsa para achar o zero de f no
       intervalo
      [a,b] com precisão epsilon. O método executa no máximo maxIter
      iterações.
      Retorna uma tupla (houveErro, raiz), onde houveErro é booleano.
    """
    ## Inicializar as variáveis Fa e Fb
    Fa = f(a)
    Fb = f(b)

    ## Teste para saber se a função muda de sinal. Se não mudar, mostrar
    ## mensagem de erro
    if Fa*Fb > 0 :
        print("Erro! A função não muda de sinal.")
        return (True, None)

    ## Inicializa o tamanho do intervalo intervX usando a função abs
    intervX = abs(b - a)

    ## Teste se intervalo já é do tamanho da precisão e retorna a raiz
       sem erros
    if intervX < epsilon:
        x = (a*f(b) - b*f(a))/(f(b) - f(a))
        return (False,x)

    ## Testes se raiz está nos extremos dos intervalos

    ## Teste se a é raiz, se for, retorna o próprio a sem erros
    if Fa == 0:
        return (False,a)

    ## Teste se b é raiz, se for, retorna o próprio b sem erros
    if Fb == 0:
        return (False,b)

    ## Mostra na tela cabeçalho da tabela
    print("k\t  a\t\t  Fa\t\t  b\t\t  Fb\t\t  x\t\t  Fx\t\tintervX")

    ## Iniciliza o k, dessa vez usaremos um for
    for k in range(1, maxIter+1):
        ## Calcula x, Fx
        x = (a*f(b) - b*f(a))/(f(b) - f(a))
        Fx = f(x)

        ## Mostra valores na tela
        print("%d\t%e\t%e\t%e\t%e\t%e\t%e\t%e"%(k,a, Fa, b, Fb, x, Fx,
            intervX))

        ## Teste do critério de parada módulo da função
```

```
        if abs(Fx) < epsilon:
            return(False,x)

        ## Testes para saber se a raiz está entre a e x ou entre x e b e
            atualiza
        ## as variáveis apropriadamente

        if Fa * Fx > 0 :
            a = x
            Fa = Fx
        else:
            b = x
            Fb = Fx

        ## Atualiza intervX e checa o outro critério de parada: tamanho
            do intervalo
        intervX = abs(b - a)
        if intervX < epsilon:
            return(False,x)

    ## Mostrar uma mensagem de erro e retorna que houve erro e a última
        raiz encontrada
    print("ERRO! número máximo de iterações atingido.")

    return (True, x)
```

Testes

O método foi testado com as seguintes funções:

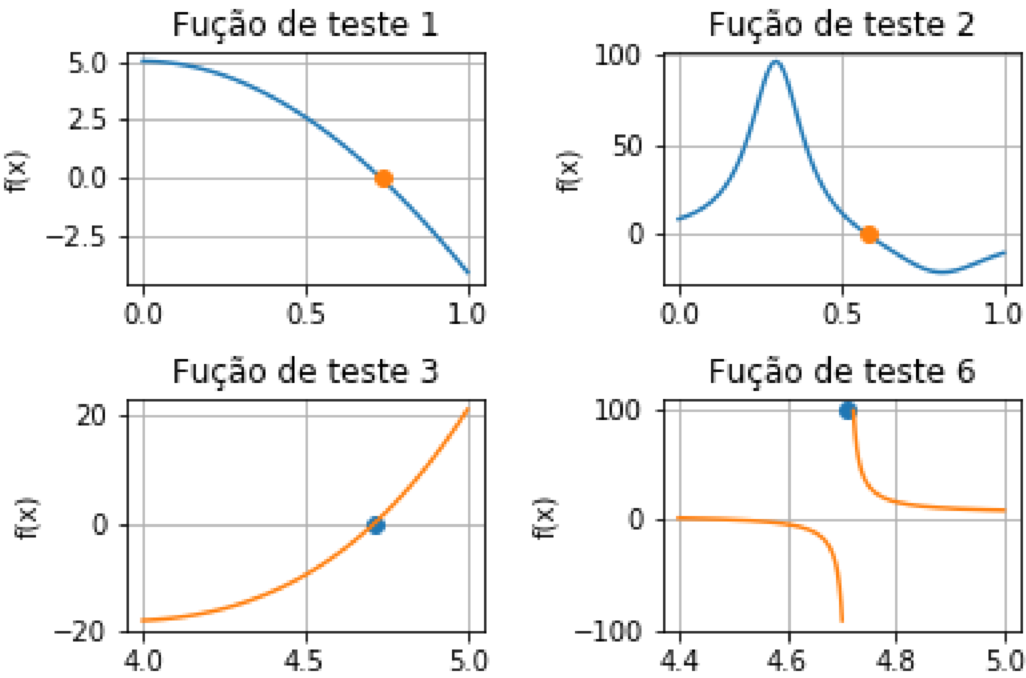
1. $f(x) = x^3 - 10x^2 + 5$
2. $f(x) = \dfrac{1}{(x-0.3)^2 + 0.01} - \dfrac{1}{(x-0.8)^2 + 0.04}$
3. $f(x) = cosh(x)cos(x)$
4. $f(x) = x - tan(x)$*

Tabela 2: Testes para o Método da Posição Falsa

$f(x)$	a	b	ϵ	k	Time (s)	$\bar{x}$
1.	0	1	10^{-4}	7	$2.950682e^{-4}$	0.7346024
2.	0	1	10^{-2}	10	$2.611405e^{-4}$	0.5800261
3.	4	5	10^{-5}	12	$8.898311e^{-4}$	4.712389
4.*	4.5	5	10^{-3}	16	$2.74667e^{-4}$	4.71240234375

* A função 4. é um caso em que o método da posição falsa falha. Foi passado um intervalo em que a função é descontínua e portanto o método encontrou uma raiz errônea.

Figura 2: Comportamento das funções testadas.



3 Método da Ponto Fixo

O método

O método do ponto fixo consiste em, dada uma função contínua f(x) contínua em um intervalo [a, b] que contém uma raiz da equação f(x)=0, encontrando uma equação equivalente $x = \phi(x)$, tal que $f(\xi) = 0$ se, e somente se, $\phi(\xi) = \xi$ e assim a parir de um x_0, utilizando-se da aproximação $x_{k+1} = \phi(x_k)$ para encontrar o ponto fixo.

Para que a função $\phi(x)$ convirja para um ponto fixo num intervalo I partindo de um ponto x_0, teremos as seguintes condições:

1. $\phi(x)$ e $\phi'(x)$ são contínuas em I;
2. $|\phi'(x)| \leq M < 1, \forall x \in I$;
3. $x_0 \in I$

Algoritmo

Considere a equação f(x)=0 e a equação equivalente $x = \phi(x)$. Supondo que as condições para convergência de $\phi(x)$ estejam válidas.

1. Dados iniciais:
   (a) x_0: aproximação inicial
   (b) ϵ: precisão
2. Se $|f(x_0)| < \epsilon$, retorne x_0. FIM.
3. k=1
4. $x_1 = \phi(x_0)$
5. Se $|f(x_1)| < \epsilon$ ou se $|x_1 - x_0| < \epsilon$, retorne x_1. FIM.
6. $x_0 = x_1$
7. k=k+1 volte ao passo 4

Implementação

```
def MPF(f, phi, x0, epsilon, iterMax=50):
    """Executa o método de Newton-Raphson para achar o zero de f
       a partir da derivada de f flin, aproximação inicial x0
       e tolerancia epsilon.
       Retorna uma tupla (houveErro, raiz), onde houveErro é booleano.
    """
    Fx0=f(x0)

    Flinx0=phi(x0)
    ## Teste se x0 já é logo a raiz
    if Fx0==0:
        return (False,x0)

    ## Escreva o cabeçalho da tabela e o valor da aproximação inicial
    print("k \t x\t\t  f(x)\t\t ")
    print("0\t%e\t%e"%( x0, Fx0))
    ## Inicie as iterações (pode ser um for)
    for k in range(1,iterMax+1):
        ## Em cada iteração:
        ##    Calcule x1 a partir de x0
        x1=phi(x0)
        Fx1=f(x1)        ##    Escreva os valores de k, x1, f(x1)
        print("%d\t%e\t%e"%(k, x1, Fx1))
        ##    Teste para o critério de parada usando módulo da função
        if ((Fx1>0) and (Fx1<epsilon)) or ((Fx1<0) and (-(Fx1)<epsilon))
            :
            return (False,x1)
        ##    Atualize o valor de x0
        x0=x1
        Fx0=Fx1
    ## Se atingir o número máximo de iterações mostra mensagem de erro e
        retorna
    ## a última raiz encontrada
    print("ERRO! número máximo de iterações atingido.")
    return (True, x0)
```

Testes

O método foi testado com as seguintes funções:

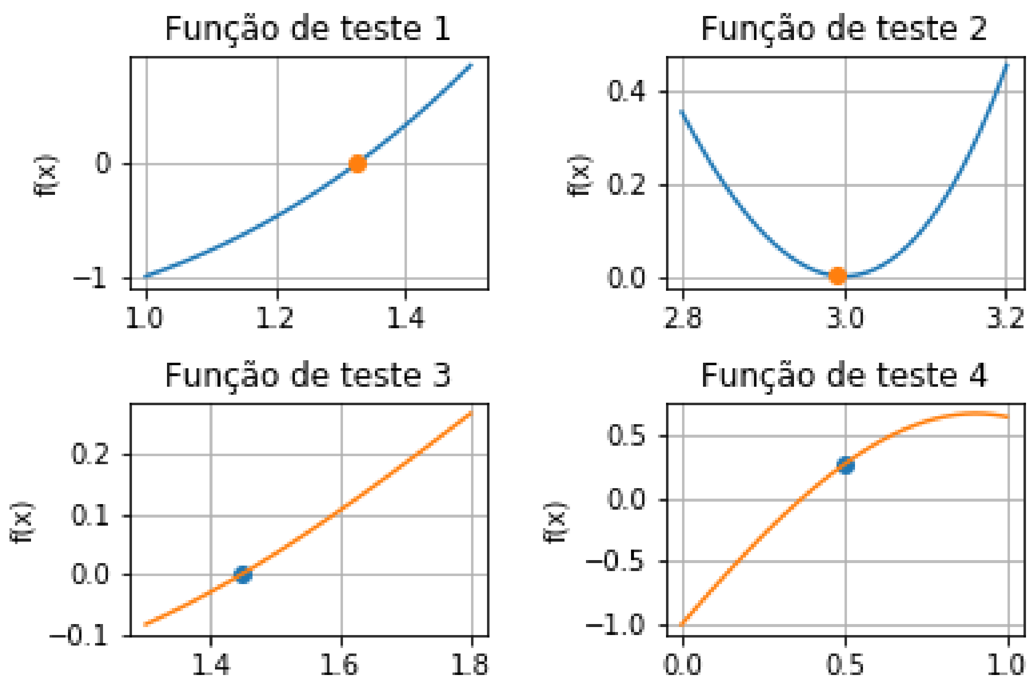
1. (a) $f(x) = x^3 - x - 1$
   (b) $\phi(x) = (3x - 1)^{1/3}$
2. (a) $f(x) = x^4 - 6x^3 + 10x^2 - 6x + 9$
   (b) $\phi(x) = x + 0.19 * f(x)$
3. (a) $f(x) = exp(-x^2) - cos(x)$
   (b) $\phi(x) = cos(x) - exp(-x^2)$

4. (a) $f(x) = 4sin(x) - exp(x)$
   (b) $\phi(x) = x - 2sin(x) + 0.5exp(x)$

Tabela 3: Testes para o Método do Ponto Fixo

$f(x)$	$\phi(x)$	x_0	e	k	Time (s)	ξ
1.(a)	1.(b)	1	10^{-6}	9	$5.735720e^{-4}$	1.324718
2.(a)	2.(b)	2	10^{-3}	41	$3.65963e^{-3}$	2.990003
3.(a)	3.(b)	1.5	10^{-4}	3	$4.915989e^{-4}$	$3.111228e^{-3}$
4.(a)	4.(b)	0.5	10^{-5}	5	$6.157360e^{-4}$	0.3705561

Figura 3: Resultado para as funções testadas.



4 Método de Newton-Raphson

O método

Ao analisarmos a ordem de convergência do método do ponto fixo na secção anterior, pudemos notar que a convergência do método será mais rápida quanto menor for $|\varphi'(\xi)|$. Assim, o método de Newton nos permite escolher uma $\varphi(x)$ tal que $\varphi'(\xi) = 0$.

De modo sucinto, da forma geral para $\varphi(x)$, temos que obter $A(x)$ tal que $\varphi'(\xi) = 0$.

$$\varphi(x) = x + A(x)f(x) \Rightarrow \varphi'(x) = 1 + A'(x)f(x) + A(x)f'(x)$$

$$\varphi'(\xi) = 1 + A'(\xi)f(\xi) + A(\xi)f'(\xi) \Rightarrow \varphi'(\xi) =' +A(\xi)f'(\xi)$$

Dito isso, para $\varphi'(\xi) = 0$ temos que

$$\varphi'(\xi) = 0 \Leftrightarrow 1 + A(\xi 0 f'(\xi) = 0 \Rightarrow A(\xi) = \frac{-1}{f'(\xi)} \Rightarrow$$

$$A(x) = \frac{-1}{f'(x)} \tag{3}$$

Logo, chegamos na nossa $\varphi(x)$ ideal com $\varphi'(\xi) = 0$.

$$\varphi(x) = x - \frac{f(x)}{f'(x)} \tag{4}$$

Apesar de ausentar-se neste relatório a desmonstração, é sabido que $\varphi(x)$ atende aos critérios de convergência vistos anteriormente. Omite-se, ainda, a prova da ordem de convergência do método, que é quadrática.

Algoritmo

A abordagem do algoritmo do método de newton é praticamente a mesma do método do ponto fixo, sendo o primeiro apenas um refinamento na escolha da função de iteração $\varphi(x)$ ideal.

Supondo que a $f(x)$, $f'(x)$ e $f''(x)$ são contínuas num intervalo I que contém a raiz $x = \xi$ de $f(x) = 0$ temos:

Dados iniciais:

1. (a) x_0, aproximação inicial;
   (b) $f(x)$, $f'(x)$;
   (c) ϵ, precisão de parada ;
   (d) iterMax, máximo de iterações.
2. Verificar se x_0 já é raiz por meio do critério de parada $|f(x)| < \epsilon$;
3. em cada interação;
4. Calcular o x_k a partir do x_0 anterior com a função de iteração $\varphi(x)$;
5. Testar o critério de parada;
6. Atualize x_0;

Implementação

```
def newton(f, flin, x0, epsilon, iterMax=50):
    """Executa o método de Newton-Raphson para achar o zero de f
       a partir da derivada de f flin, aproximação inicial x0
       e tolerância epsilon.
       Retorna uma tupla (houveErro, raiz), onde houveErro é booleano.
    """
    ## Teste se x0 já é logo a raiz
    Fx = f(x0)
    if Fx < epsilon:
        return (False, Fx)
    ## Cabeçalho da Tabela com os valores iniciais
    print ("k\t x\t\t f(x)\t\t ")
    print("0\t %e\t %e\t" % (x0,Fx))
    ## Inicia as iterações
    for i in range(1, iterMax+1):
        ## Em cada iteração:
        ##    Calcula xk a partir de x0
        xk = x0 - f(x0)/flin(x0)
        Fxk = f(xn)
        ##    Escreve os valores de k, x1, f(x1)
        print ("%d\t %e\t %e\t " % (i, xk, Fxk))
        ##    Teste para o critério de parada usando módulo da função
```

```
        if abs(Fxk) < epsilon:
            return (False, xk)
        ##    Atualiza o valor de x0
        x0 = xk
    ## Se atingir o número máximo de iterações mostra mensagem de erro e
        retorna
    ## a última raiz encontrada
    print("Erro! Número máximo de iteraçcoes atingido!")
    return (True, None)
```

Testes

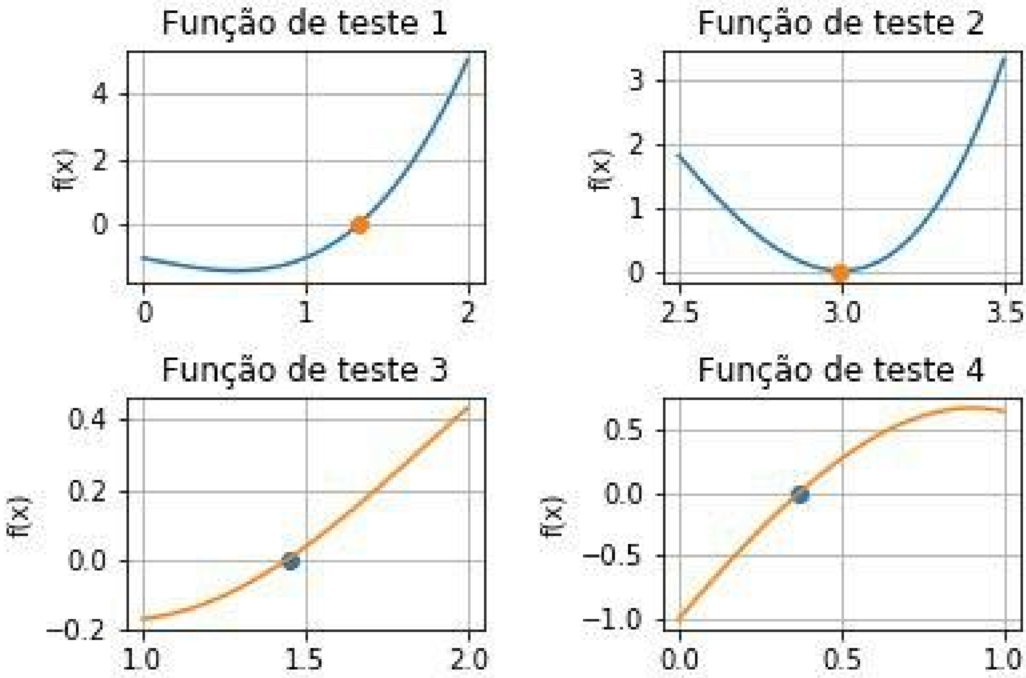
O método foi testado com as seguintes funções:

1. (a) $f(x) = x^3 - x - 1$
   (b) $f'(x) = 3x^2 - 1$
2. (a) $f(x) = x^4 - 6x^3 + 10x^2 - 6x + 9$
   (b) $f'(x) = 4x^3 - 18x^2 + 20x - 6$
3. (a) $f(x) = exp(-x^2) - cos(x)$
   (b) $f'(x) = -2xexp(-x^2) - cos(x)$
4. (a) $f(x) = 4sin(x) - exp(x)$
   (b) $f'(x) = 4cos(x) - exp(x)$

Tabela 4: Testes para o Método de Newton-Raphson

$f(x)$	$f'(x)$	x_0	e	k	Time (s)	ξ
$1(a)$	$1(b)$	0	10^{-6}	21	$4.77e^{-3}$	1.324718
$2(a)$	$2(b)$	2	10^{-3}	5	$6.178e^{-3}$	2.990569
$3(a)$	$3(b)$	1.5	10^{-4}	5	$2.19e^{-3}$	1.447416
$4(a)$	$4(b)$	0.5	10^{-5}	3	$1.44e^{-3}$	0.3705581

Figura 4: Comportamento das funções testadas no Método de Newton.



5 Método da Secante

O método

Como podemos observar na definição do método de Newton, é necessário calcularmos previamente a $f'(x)$ para fornecer ao programa.

Uma alternativa a esse inconveniente é, ao invés de calcularmos a derivada exatamente, podemos aproximá-la por meio de uma reta secante, ou:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k_1})}{x_k - x_{k-1}} \tag{5}$$

onde $x_k e x_{k-1}$ são aproximações bastante próximas das raízes.

Desse modo, temos que a nova função iterativa $\varphi(x_k)$ será:

$$\varphi(x_k) = x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1}}{x_k-x_{k-1}}}$$

$$\Rightarrow \varphi(x_k) = \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})} \tag{6}$$

Basicamente, a partir das duas aproximações x_{k-1} e x_k, o ponto x_{k+1} é tido como a abcisssa da intersecção do eixo x com a reta secante que passa por $(x_{k-1}, f(x_{k-1})$ e $(x_k, f(x_k))$.

Algoritmo

Como já se deve suspeitar, a formulação do algoritmo para o método da secante é exatamente igual ao método de Newton-Raphson, exceto pela aproximação feita sobre $\varphi(x_k)$, que acarreta na possibilidade de divergência se $f(x_k) \approx f(x_{k-1})$

/par Como parâmetros da função, pasamos x_0 e x_1 para calcularmos a aproxiamação de $f'(x)$ inicial

Implementação

```
def secante(f, x0, x1, epsilon, iterMax=50):
    """Executa o método da Secante para achar o zero de f
       a partir das aproximações x0 e x1, e da tolerância
       epsilon.
       Retorna uma tupla (houveErro, raiz), onde houveErro é booleano.
    """
    ## Testa se x0 e x1 já são raízes
    Fx0 = f(x0)
    Fx1 = f(x1)
    if Fx0 == 0:
        return (False, x0)
```

```
    elif Fx1 == 0:
        return (False, x1)
    ## Escreve o cabeçalho da tabela e as linhas para x0 e x1
    print ("k\t x2\t\t f(x2)\t\t ")
    print("-\t %e\t %e\t" % (x0,Fx0))
    print("-\t %e\t %e\t" % (x1,Fx1))
    ## Inicia as iterações (pode ser um for)
    for k in range (1, iterMax+1):
        ## Em cada iteração:
        ##    Calcula x2 a partir de x0 e x1
        x2 = (x0*Fx1 - x1*Fx0)/(Fx1 - Fx0)
        Fx2 = f(x2)
        ##    Escreve os valores de k, x2, f(x2)
        print("%d\t %e\t %e\t" % (k, x2, Fx2))
        ##    Testa para o critério de parada usando módulo da função
        if abs(Fx2) < epsilon:
            return (False, x2)
        ##    Atualiza os valores de x0 e x1
        x0 = x1
        Fx0 = f(x0)
        x1 = x2
        Fx1 = f(x1)

    ## Se atingir o número máximo de iterações mostra mensagem de erro e
       retorna
    ## a última raiz encontrada
    print("ERRO! Número máximo de interações atingido")
    return (True, x2)
```

Testes

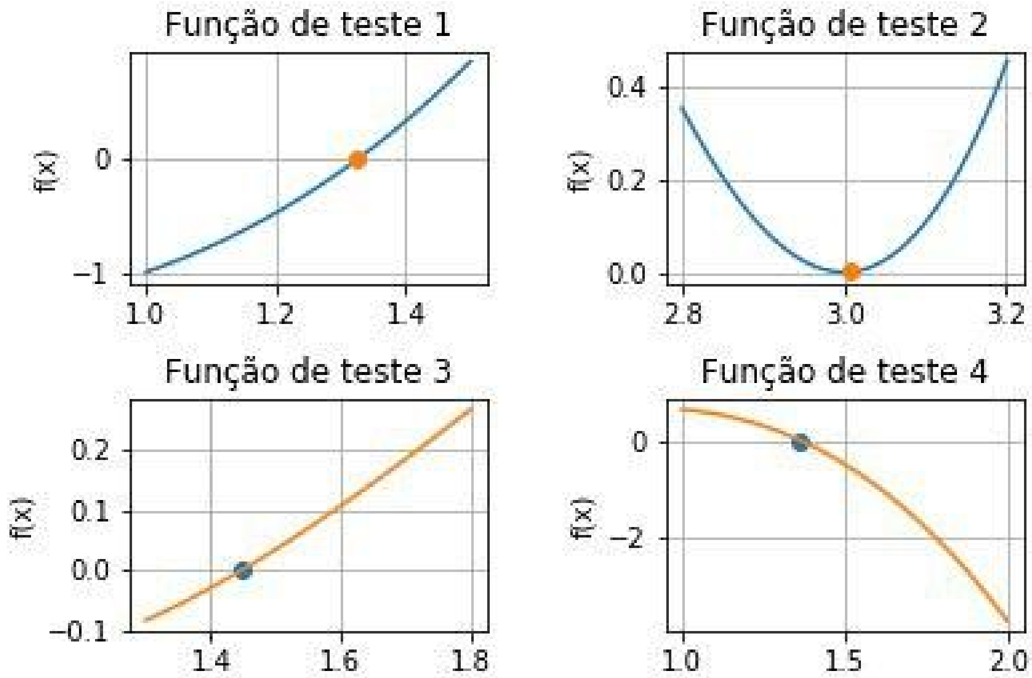
Os testes realizados para o método da Secante foram:

1. $f_1(x) = x^3 - x - 1$
2. $f_2(x) = x^4 - 6x^3 + 10x^2 - 6x + 9$
3. $f_3(x) = exp(-x^2) - cos(x)$
4. $f_4(x) = 4sin(x) - exp(x)$

Tabela 5: Testes para o Método da Secante

$f(x)$	x_0	x_1	e	k	Time (s)	ξ
1	0	0.5	10^{-6}	26	$6.31e^{-3}$	1.324718
2	2	1.5	10^{-3}	5	$2.60e^{-3}$	3.007243
3	1	2	10^{-4}	5	$2.27e^{-3}$	1.447413
4	1	1.5	10^{-5}	3	$5.05e^{-4}$	1.364934

Figura 5: Comportamento das funções testadas no Método da Secante.



6 Comparação dos Métodos

A tabela 6 contém os resultado das funções (1-4) e nota-se que não há uma consistência de qual método é mais rápido. Isso se deve ao comportamento da função, figura 2 abaixo. Quando a função tem um comportamento mais linear, como na função 1 e função 3, o método da posição falsa se sobressai, pois, o intervalo é reduzido através de uma reta secante sobre os extremos do intervalo. Já quando a função tem um comportamento mais ondulado, não linear, o método da bisseção se sobressai por sempre reduzir o intervalo ao meio, ao contrário da posição falsa, que por vezes a redução do intervalo se torna bem pequena, causando maiores iterações e assim se tornando mais lento. Na função 4, como a função não é continua no intervalo dado, os dois métodos falham em encontrar a raiz.

Tabela 6: Comparação dos métodos para $f_1(x)$ a $f_4(x)$

$f(x)$	Bisseção	Posição Falsa
Número de iterações para $f_1(x)$	13	7
Tempo de execução para $f_1(x)$	$6.29925e^{-4}$	$2.950682e^{-4}$
Número de iterações para $f_2(x)$	6	10
Tempo de execução para $f_2(x)$	$6.32075e^{-4}$	$2.611405e^{-4}$
Número de iterações para $f_3(x)$	16	12
Tempo de execução para $f_3(x)$	$1.48355e^{-3}$	$8.898311e^{-4}$
Número de iterações para $f_4(x)$	16	16
Tempo de execução para $f_4(x)$	$2.74667e^{-4}$	$2.74667e^{-4}$

A tabela 7 contém os resultado das funções (1-4) e nota-se que o MPF tem seu tempo de código variado de acordo com a complexidade da função de entrada e do $\phi(x)$, o método de Newton geralmente tem mais velocidade que o método da secante, visto que ele aproxima a derivada pela secante e isso pode ser mais custoso em tempo de algoritmo. Em suma, o tempo para convergência e o número de interação dos métodos geralmente dependerá das funções passadas como referência e da aproximação inicial, além disso dependendo do método podemos lidar com operações mais custosas em tempo para o algoritmo, como no método da secante em que a derivada é obtada a partir de uma aproximação, e isso definirá a diferença entre os métodos, que tem suas "regiões"de operação ótima.

Tabela 7: Comparação dos métodos para $f_1(x)$

$f(x)$	MPF	Newton-Raphson	Secante
Número de iterações para $f_1(x)$	9	21	26
Tempo de execução para $f_1(x)$	$5.736e^{-3}$	$4.77e^{-3}$	$6.31e^{-3}$
Número de iterações para $f_2(x)$	41	5	5
Tempo de execução para $f_2(x)$	$3.660e^{-3}$	$6.178e^{-3}$	$2.60e^{-3}$
Número de iterações para $f_3(x)$	3	5	5
Tempo de execução para $f_3(x)$	$4.915e^{-3}$	$2.19e^{-3}$	$2.27e^{-3}$
Número de iterações para $f_4(x)$	5	3	3
Tempo de execução para $f_4(x)$	$6.157e^{-3}$	$1.44e^{-3}$	$5.05e^{-4}$

7 Conclusão

Do que foi apresentado anteriormente, podemos tirar conclusões importantes acerca da utilização dos variados métodos de aproximação de raízes não-lineares estudados.

Basicamente, implementamos e analisamos métodos que, em suas respectivas abordagens, eram eficientes na resolução dos nossos problemas. O primeiro grupo, consistia dos métodos da bisseção, que nos prevê o número de iterações antes de sua execução; e o da posição falsa, que refina por uma aproximação linear a busca binária do método anterior.

No segundo grupo, trabalhamos com os métodos de uma função iterativa $\varphi(x)$ que nos proporcionava uma conversão mais rápida. O método do ponto fixo foi o primeiro, seguido pelo método de Newton-Raphson, que melhorava esse, e, por fim, o método da secante, que retirava uma barreira computacional do método de Newton.

Dito isso, apesar da eficácia geral de todos os métodos, cabe ao usuários destes a responsabilidade de escolher sabiamente a ferramenta, a depender da função em questão, o número de raízes e o intervalo dado.

8 Anexo 1

8.1 Código de teste do método da bisseção

```python
from timeit import default_timer as timer
import numpy as np


def f1(x):
    return x**3 - 10*(x**2) + 5

def f2(x):
    return ((1/((x - 0.3)**2 + 0.01)) - (1/((x - 0.8)**2 + 0.04)))

def f3(x):
    return np.cosh(x)*np.cos(x)

def f4(x):
    return x - np.tan(x)


print("### F1(x) ###")

a = 0
b = 1

epsilon = 10**(-4)
maxIter = 20

start = timer()
```

```
(houveErro, raiz) = bissecao(f1, a, b, epsilon, maxIter)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Bisseção retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)


print("### F2(x) ###")

a = 0
b = 1

epsilon = 10**(-2)
maxIter = 100


start = timer()

(houveErro, raiz) = bissecao(f2, a, b, epsilon, maxIter)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Bisseção retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)

print("### F3(x) ###")

a = 4
b = 5

epsilon = 10**(-5)
maxIter = 20

start = timer()

(houveErro, raiz) = bissecao(f3, a, b, epsilon, maxIter)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Bisseção retornou um erro.")
if raiz is not None:
```

```
    print("Raiz encontrada: %s" % raiz)

print("### F4(x) ###")

a = 4.5
b = 5

epsilon = 10**(-3)
maxIter = 20

(houveErro, raiz) = bissecao(f4, a, b, epsilon, maxIter)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Bisseção retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)
```

8.2 Código de teste do método da Posição Falsa

```
import numpy as np

def f(test,x = None,a = None,b = None,plot=False):

    if a != None and b != None:
        intervaX = np.arange(a,b+0.01,0.01)
    fx = []
    aux2 = []
    aux = 0

    if test == 1:

        if plot == True:
            for x in intervaX:
                y  = (x**3 - 10*x**2 + 5)
                aux2.append(y)
                if aux != 0 :
                    dif = abs(aux2[aux - 1] - y)
                    if  dif > 100:
                        y = np.inf
                fx.append(y)
                aux = aux + 1
        else:
            return (x**3 - 10*x**2 + 5)

    elif test == 2:

        if plot == True:
            for x in intervaX:
                y  = (1/((x - 0.3)**2 + 0.01) - 1/((x - 0.8)**2 + 0.04))
```

```
                aux2.append(y)
                if aux != 0 :
                    dif = abs(aux2[aux - 1] - y)
                    if  dif > 100:
                        y = np.inf
                fx.append(y)
                aux = aux + 1
        else:
            return (1/((x - 0.3)**2 + 0.01) - 1/((x - 0.8)**2 + 0.04))

    elif test == 3:

        if plot == True:
            for x in intervaX:
                y  = (np.cosh(x)*np.cos(x))
                aux2.append(y)
                if aux != 0 :
                    dif = abs(aux2[aux - 1] - y)
                    if  dif > 100:
                        y = np.inf
                fx.append(y)
                aux = aux + 1
        else:
            return (np.cosh(x)*np.cos(x))

    elif test == 4:

        if plot == True:
            for x in intervaX:
                y  = (x*np.sin(x) + 3*np.cos(x) - x)
                aux2.append(y)
                if aux != 0 :
                    dif = abs(aux2[aux - 1] - y)
                    if  dif > 100:
                        y = np.inf
                fx.append(y)
                aux = aux + 1
        else:
            return (x*np.sin(x) + 3*np.cos(x) - x)

    elif test == 5:

        if plot == True:
            for x in intervaX:
                y  = np.cos(x) - 3*np.sin(np.tan(x) - 1)
                aux2.append(y)
                if aux != 0 :
                    dif = abs(aux2[aux - 1] - y)
                    if  dif > 100:
                        y = np.inf
                fx.append(y)
                aux = aux + 1
        else:
```

```
            return np.cos(x) - 3*np.sin(np.tan(x) - 1)

    elif test == 6:

        if plot == True:
            intervaX = np.arange(a,b+0.001,0.001)
            for x in intervaX:
                y  = x - np.tan(x)
                aux2.append(y)
                if aux != 0 :
                    dif = abs(aux2[aux - 1] - y)
                    if  dif > 10:
                        y = np.inf
                fx.append(y)
                aux = aux + 1
        else:
            return x - np.tan(x)

def f1(x):
    return f(1,x)
def f2(x):
    return f(2,x)
def f3(x):
    return f(3,x)
def f6(x):
    return f(6,x)

from timeit import default_timer as timer

start = timer()
(houveErro, raiz) = false_pos(f1,0,1,0.0001)
end = timer()
print("Tempo de execucao total: %e segundos" % (end - start))

if houveErro:
    print("O Método da Posição Falsa retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)

start = timer()
(houveErro, raiz) = false_pos(f2,0,1,0.01)
end = timer()
print("Tempo de execução total: %e segundos" % (end - start))

if houveErro:
    print("O Método da Posição Falsa retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)

start = timer()
(houveErro, raiz) = false_pos(f3,4,5,0.00001)
end = timer()
print("Tempo de execução total: %e segundos" % (end - start))
```

```
if houveErro:
    print("O Método da Posição Falsa retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)

start = timer()
(houveErro, raiz) = false_pos(f6,4,5,0.001)
end = timer()
print("Tempo de execução total: %e segundos" % (end - start))

if houveErro:
    print("O Método da Posição Falsa retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)
```

8.3 Código de teste do método do ponto fixo

```
import sys
import numpy as np
from timeit import default_timer as  timer

def f1(x):
     return x**3 - x - 1

def phi1(x):
    return (x**3-1)**(1/3)

def f2(x):
    return x**4 - 6*x**3 + 10*x**2 - 6*x + 9

def phi2(x):
    return x+0.19*(x**4 - 6*x**3 + 10*x**2 - 6*x + 9)

def f3(x):
    return np.exp(-x**2) - np.cos(x)

def phi3(x):
    return np.cos(x)-np.exp(-(x**2))

def f4(x):
    return 4*np.sin(x) - np.exp(x)

def phi4(x):
    return x-2*np.sin(x)+0.5*np.exp(x)

print("### F1(x) ###")

x0 = 1

epsilon = 10**(-6)
maxIter = 50

start = timer()
```

```
(houveErro, raiz) = MPF(f1, phi1, x0, epsilon, iterMax)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método do ponto fixo retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)


print("### F2(x) ###")

x0 = 2

epsilon = 10**(-3)
maxIter = 50


start = timer()

(houveErro, raiz) = newton(f2, phi2, x0, epsilon, iterMax)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método do ponto fixo retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)


print("### F3(x) ###")

x0 = 1.5

epsilon = 10**(-4)
maxIter = 50

start = timer()

(houveErro, raiz) = newton(f3, phi3, x0, epsilon, iterMax)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método do ponto fixo retornou um erro.")
if raiz is not None:
```

```
    print("Raiz encontrada: %s" % raiz)


print("### F4(x) ###")

x0 = 0.5

epsilon = 10**(-5)
maxIter = 50

start = timer()

(houveErro, raiz) = newton(f4, phi4, x0, epsilon, iterMax)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método do ponto fixo retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)
```

8.4 Código de teste do método de Newton-Raphson

```
from timeit import default_timer as timer
import numpy as np


def f1(x):
     return x**3 - x - 1

def flin1(x):
    return 3*x**2 - 1

def f2(x):
    return x**4 - 6*x**3 + 10*x**2 - 6*x + 9

def flin2(x):
    return 4*x**3 - 18*x**2 + 20*x - 6

def f3(x):
    return math.exp(-x**2) - cos(x)

def flin3(x):
    return -2*math.exp(-x**2) - cos(x)

def f4(x):
    return 4*math.sin(x) - math.exp(x)

def flin4(x):
    return 4*math.cos(x) - math.exp(x)
```

```python
print("### F1(x) ###")

x0 = 0

epsilon = 10**(-6)
maxIter = 50

start = timer()

(houveErro, raiz) = newton(f1, flin1, x0, epsilon, iterMax)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método de Newton retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)


print("### F2(x) ###")

x0 = 2

epsilon = 10**(-3)
maxIter = 50


start = timer()

(houveErro, raiz) = newton(f2, flin2, x0, epsilon, iterMax)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método de Newton retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)


print("### F3(x) ###")

x0 = 1.5

epsilon = 10**(-4)
maxIter = 50

start = timer()

(houveErro, raiz) = newton(f3, flin3, x0, epsilon, iterMax)
```

```python
end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método de Newton retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)


print("### F4(x) ###")

x0 = 0.5

epsilon = 10**(-5)
maxIter = 50

start = timer()

(houveErro, raiz) = newton(f4, flin1, x0, epsilon, iterMax)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método de Newton retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)
```

8.5 Código de teste do método da Secante

```python
from timeit import default_timer as timer
import numpy as np


def f1(x):
     return x**3 - x - 1

def f2(x):
    return x**4 - 6*x**3 + 10*x**2 - 6*x + 9

def f3(x):
    return math.exp(-x**2) - cos(x)

def f4(x):
    return 4*math.sin(x) - math.exp(x)

print("### F1(x) ###")

x0 = 0
x1 = 0.5
```

```
epsilon = 10**(-6)
maxIter = 50

start = timer()

(houveErro, raiz) = secante(f1, x0, x1, epsilon, iterMax=50)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Secante retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)


print("### F2(x) ###")

x0 = 2
x1 = 1.5

epsilon = 10**(-3)
maxIter = 50


start = timer()

(houveErro, raiz) = secante(f2, x0, x1, epsilon, iterMax=50)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Secante retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)

print("### F3(x) ###")

x0 = 1
x1 = 2

epsilon = 10**(-4)
maxIter = 50

start = timer()

(houveErro, raiz) = secante(f3, x0, x1, epsilon, iterMax=50)

end = timer()
```

```
print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Secante retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)

print("### F4(x) ###")

x0 = 1
x1 = 1.5

epsilon = 10**(-5)
maxIter = 50

start = timer()

(houveErro, raiz) = secante(f4, x0, x1, epsilon, iterMax=50)

end = timer()

print("Tempo de execução total: %e segundos" %(end - start))

if houveErro:
    print("O Método da Secante retornou um erro.")
if raiz is not None:
    print("Raiz encontrada: %s" % raiz)
```