Universidade Federal de Goiás
Goiânia, março de 2021

Atividade avaliativa 1 - Métodos iterativos para obter zeros de funções reais

Aluno: Matheus Lázaro Honório da Silva - 201801523
Disciplina: Cálculo Numérico - IME0065

Sumário

Métodos iterativos para obter zeros de funções reais

I. Método da Bissecção

Teoria sobre o método da Bissecção

- Livro Neide - Cap 3. Equações não lineares

- Reduz o comprimento do intervalo que contém a raiz, de maneira sistemática.
- Considere o intervalo [a, b] para o qual f(a) * f(b) < 0.
- No método da bissecção calculamos o valor da função f(x) no ponto médio: x1 = (a + b)/2. Portanto, existem três possibilidades:

  1) O valor da função calculado no ponto x1 é nulo
     - f(x1) = 0
     - Neste caso, x1 é o zero da função, então paramos
  2) f(a) * f(b) < 0
     - a função tem um zero entre (a) e x1.
     - O processo é repetido sobre o novo intervalo [a, x1]
  3) f(a) * f(x1) > 0
     - Segue que, f(b) * f(x1) < 0, desde que seja conhecido que f(a) e f(b) têm sinais opostos.
     - A função tem um zero entre x1 e b, e o processo é repetido com [x1, b]
- A repetição do método é chamado ITERAÇÃO e as aproximações sucessivas são os termos iterados.

Algoritmo do método da Bissecção escrito em Python

```
# Exemplo de função em que a solução é determinada usando
# o método da bissecção
import math

def funcao(x):
    return (math.exp(-x**2) - math.cos(x))
def metodoBisseccao(a, b):
    if(funcao(a) * funcao(b) >= 0):
        print("A condição f(a) * f(b) < 0 deve ser
verdadeira!\n")
        return
    maxIteracoes = 14
    i = 0
    c = a
    erro = 0.000061035
    while((b - a) >= erro):  # Erro tolerável
        # if(i == maxIteracoes + 1): # não converge
          #  break
        c = (a + b)/2 # Ponto médio

        if(funcao(c) == 0.0):#Verificar se o ponto médio é raiz
            break
        if(funcao(c) * funcao(a) < 0): # Verificar a condição
para retornar ao looping
            b = c
        else:
            a = c
        i = i + 1
    print("============================")
    print("Valor da raiz: x = ", "%.8f"%c)
    print("f(x) = ", "%.9f"%(funcao(c)))
    print("Erro em x = ", "%.9f" %erro)
    print("Número de Iterações = ", "%d" %(i-1))
# Principal / Main
a = 1
b = 2
metodoBisseccao(a, b)
```

II. Método da posição falsa

Descrição do método da posição falsa

- Dada uma função f(x), x Real e dois números 'a' e 'b' tais que f(a) * f(b) < 0 e f(x) seja contínua em [a, b].
- f(x) função algébrica

- Semelhanças com o método de bissecção:
  - Mesmas premissas: Este método também assume que a função é contínua em [a, b] e dados dois números 'a' e 'b' são tais que f (a) * f (b) <0.

  - Sempre Converge: como a Bissecção, sempre converge, geralmente consideravelmente mais rápido do que a Bissecção
  - Mas às vezes muito mais lentamente do que a Bissecção.

- Diferenças com o Método da Bissecção:
  - Difere-se no fato de que fazemos um acorde unindo os dois pontos [a, f (a)] e [b, f (b)]. Consideramos o ponto em que a corda toca o eixo x e o denominamos c.

- Etapas:
  - Equação da linha que conecta os dois pontos.
    y - f (a) = ((f (b) -f (a)) / (ba)) * (xa)
  - Temos que encontrar o ponto que toca o eixo x.
  - Para isso, colocamos y = 0.

  - Então x = a - (f (a) / (f (b) -f (a))) * (ba)
    x = (a * f (b) - b * f (a)) / (f (b) -f (a))

  - Este será nosso c que é c = x.
  - Se f (c) == 0, então c é a raiz da solução.
  - Caso contrário, f (c)! = 0
  - Se o valor f (a) * f (c) <0, então a raiz fica entre a e c. Então, recorremos para a e c.
  - Caso contrário, Se f (b) * f (c) <0, então a raiz está entre be c. Portanto, recorremos b e c.
  - Outra função dada não segue uma das suposições.

- Como a raiz pode ser um número de ponto flutuante e pode convergir muito lentamente no pior dos casos, iteramos por um grande número de vezes de forma que a resposta se torna mais próxima da raiz.

Algoritmo do método da posição falsa escrito em Python

```
import math
maximoIteracoes = 6

def funcao(x):
    return (math.exp(-x**2) - math.cos(x))
def metodoPosicaoFalsa(a, b):
    if(funcao(a) * funcao(b) >= 0):
        print("A condição f(a) * f(b) < 0 deve ser
respeitada!")
        return -1
    c = a # Inicialização do resultado
    erro = 0.552885221
    i = 1
    while((b - a) >= erro):
        # if(i == maximoIteracoes + 1):
          #  break

        c = (a * funcao(b) - b * funcao(a)) / (funcao(b) -
funcao(a))
        if(funcao(c) == 0):
            break
        elif(funcao(c) * funcao(a) < 0):
            b = c
        else:
            a = c
        i += 1
    print("====================")
    print("Raiz da função: x = ", "%.4f" %c)
    print("f(x)", "%.9f" %(funcao(c)))
    print("Erro em x = ", "%.9f" %(erro))
    print("Número de Iterações = ", "%d" %(i - 1))
a = 1
b = 2
metodoPosicaoFalsa(a, b)
```

III. Método do ponto fixo

- Descrição do método do ponto fixo

- Encontrar as raízes das equações pelo método de iteração de ponto fixo
- Método clássico entre os métodos iterativos
- Equação de transformação simples
- Geralmente termo de mudança, quadrado, sinal de raiz, ...
- Existem dois teoremas no estudo da convergência:
- Teorema geral:
    - Usado globalmente
    - Quando a equação h(x) é uma função contínua no intervalo fechado e no intervalo correspondente.
    - É necessário satisfazer a compressibilidade e a vedação.
    - Fechamento:
    - Qualquer valor em um determinado intervalo é substituído na função correspondente, e o valor da função obtido também estará neste intervalo.
    - Compressibilidade:
    - Existe uma constante menor que 0, de forma que no intervalo correspondente, o valor absoluto da diferença entre os valores da função entre quaisquer dois pontos seja menor que esta constante.
    - O superior corresponde à diferença de distância entre dois pontos.
- Teorema Local:
    - Intervalo em torno do ponto fixo
    - h(x):
    - Valor absoluto da derivada, que deve ser menor que 1
    - Há uma parte local onde a convergência pode ser alcançada.

É um método para encontrar a raiz real de uma equação não linear por aproximação sucessiva

- Requer uma estimativa inicial para começar
- Por ser um método aberto, sua convergência não é garantida
- Para encontrar a raiz da equação não linear:
    - f(x) = 0
    - Escrevemos f(x) = 0, em que x = g(x)
    - Se x0 é estimativa inicial, a próxima raiz é aproximada neste método é obtida por:
    - x1 = g(x1)
    - A próxima raiz aproximada é obtida usando do valor de x1:
    - x2 = g(x2)
    - O processo é repetido até obtermos raízes com a precisão desejada.
    - Para convergência, os seguintes critérios devem ser satisfeitos:
    - $| g'(x) | < 1$

Algoritmo do método do ponto fixo escrito em Python

```
import math

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

def funcao2(x):
    return (math.cos(x) - math.exp(-x**2) + x)

def metodoPontoFixo():
    etapa = 1

    x0 = 1.5
    erro = 0.00019319
    N = 50
    x1 = x0

    while(abs(funcao(x1)) > erro):
        x1 = funcao2(x0)

        x0 = x1

        print("Etapa: ", "%d" % etapa)
        print("x0: ", "%.6f" % x0)
        print("f(x0): ", "%.6f" % funcao((x0)))
        print("x1: ", "%.6f" % x1)
        print("-----------------")

        # if (etapa >= N): # não converge
          #  break

        etapa += 1



    print("Raiz da funçao: x = ", "%.9f" %x1)
    print("f(x) =  ", "%.9f" %(funcao(x1)))
    print("Número de iterações = ", "%d" %(etapa))


metodoPontoFixo()
```

IV. Método de Newton Raphson

Método de Newton Raphson

- Dada uma função f(x), sendo x pertencente aos Reais.
- f(x) é equação algébrica.
- A derivada é fornecida como entrada.
- No método da bissecção:
- Recebíamos um intervalo.
- Tem garantia para convergir.
- No método de Newton Raphson.
- Recebemos um valor estimado inicial da raiz.
- Pode não convergir em alguns casos.
- Requer derivada, o que pode ser trabalhoso computacionalmente.
- Normalmente, converge mais rápido.

- A fórmula:
    - Começando da estimativa inicial x[1] , o método de Newton Raphson usa a fórmula abaixo para encontrar o próximo valor de x, ou seja, x[n + 1] do valor anterior x[n]
    - $x[n+1] = x[n] - f(x[n])/f'(x[n])$
- Notas:
    - Geralmente usamos esse método para melhorar o resultado obtido pelo método da bissecção ou pelo método da posição falsa.
    - O método babilônico para raiz quadrada é derivado do método de Newton-Raphson.

Algoritmo do método de Newton Raphson escrito em Python

```
import math
def funcao(x):
    y = (math.exp(-x**2) - math.cos(x))
    return y


# x = linspace(0, 1, 1000) # retorna um vetor de 2 elementos,
com números de 0 a 1,

def derivada(x):
    return math.sin(x) - 2 * x * math.exp(-x**2)
    # h = 0.0017072
    # derivada = (funcao(x + h) - funcao(x)) / h # metodo do
quociente
    # return derivada

def metodoNewtonRaphson(x):
    return (x - (funcao(x) / derivada(x)))

def iteracao(p):
    x = p
    erro = 0.00017072
    qtdIteracoes = 0
    # for i in range(n): # range: retorna uma sequência e
numeros (vetor), de 0 a n
    while(abs(funcao(x)) > erro):
        x = metodoNewtonRaphson(x)
        qtdIteracoes += 1


    print("raiz: ", "%.9f" %x)
    print("f(x): ", "%.9f" %funcao(x))
    print("Número de Iterações ", qtdIteracoes)

iteracao(1.5)
```

V. Método da Secante

- Método da Secante para encontrar a raiz de um equação

- Usado para encontrar a raiz de uma equação f(x) = 0
- É iniciado a partir de duas estimativas distintas x1 e x2 para a raiz
- Procedimento iterativo com interpolação linear para uma raiz
- Critério de parada:
    - A diferença entre dois valores intermediários for menor que o fator de convergência
- O método da secante contorna a necessidade de se obter a derivada do método de Newton.
- Substituímos pelo quociente das diferenças
- f1(xk) = (f(x[k]) - f(x[k - 1])) /
  (x[k] - x[k-1])
- x[k] e x[k - 1] são aproximações para a raiz
- φ(x[k]) = (x[k - 1] * f(x[k]) - x[k] * f(x[k - 1]) /
  (f(x[k]) - f(x[k - 1]))

Algoritmo do método da Secante escrito em Python

```
import math

def funcao(x):
    f = (x**3 - x - 1) # 89 Livro Marcia A. Gomes Ruggiero
    return f

def metodoDaSecante(x1, x2, erro):
    numIteracoes = 0
    while True:
        # Resultado da função de iteração,
        # pelo Quociente das diferenças
        x0 = ((x1 * funcao(x2) - x2 * funcao(x1)) /
              (funcao(x2) - funcao(x1)))

        c = funcao(x1) * funcao(x0)

        x1 = x2 # Mudança de valores das variáveis
        x2 = x0
        numIteracoes += 1

        if(c == 0):
            break
        # Resultado da função de iteração,
        # pelo Quociente das diferenças para as novas variáveis
        xm = ((x1 * funcao(x2) - x2 * funcao(x1))/
              (funcao(x2) - funcao(x1)))
        if(abs(xm - x0) < erro):
            break
    raiz = round(x0, 9) # 9 dígitos de precisão
    print("Raiz da equação = ", raiz)
    print("f(x) = ", "%.9f" %funcao(raiz))
    print("Número de iterações = ", numIteracoes)
    return raiz
x1 = 0.0
x2 = 0.5

erro = 0.000008998843
raiz = metodoDaSecante(x1, x2, erro)
```

Aplicação dos algoritmos em exemplos do livro

- RUGGIERO, M. A. G.; LOPES, V. L. R. Cálculo Numérico: Aspectos Teóricos e Computacionais. 2 ed. São Paulo: Makron Books, 1996.

Exemplo 18

$$f(x) = e^{-x^2} - \cos(x)$$

Resultados obtidos aplicando os códigos apresentados neste trabalho:

	Bissecção	Posição Falsa	Método do Ponto Fixo $\varphi(x) = \cos(x) - e^{-x^2} + x$	Newton	Secante
Dados iniciais	[1, 2]	[1, 2]	x0 = 1.5	x0 = 1.5	x0 = 1; x1 = 2
xbarra	1.44741821	1.44735707	1.447717894	1.447416347	1.447413447
f(xbarra)	0.000002507	-0.000036388	0.000193187	0.000001320	-0.000000524
Erro em x	0.000061035	0.552885221	0.00019319	0.00017072	0.000018553
Número de Iterações	14	6	6	2	5

Exemplo 19

$$f(x) = x^3 - x - 1$$

Resultados obtidos aplicando os códigos apresentados neste trabalho:

	Bissecção	Posição Falsa	Método do Ponto Fixo $\varphi(x) = (x+1)^{1/3}$	Newton	Secante
Dados iniciais	[1, 2]	[1, 2]	x0 = 1	x0 = 0	x0 = 0; x1 = 0.5
xbarra	1.32471800	1.32471776	1.324717372	1.324717957	1.324717957
f(xbarra)	0.0000001759	-0.000000829	-0.00000249	0.0000000000027471358 5	-0.000000001 04375152965
Erro em x	0.000000954	0.675282500	0.000003599	0.0000006299186	0.0000000089 98843
Número de Iterações	20	17	9	21	27

Exemplo 20

$$f(x) = 4\text{sen}(x) - e^x$$

	Bissecção	Posição Falsa	Método do Ponto Fixo	Newton	Secante
Dados iniciais	[0, 1]	[0, 1]	x0 = 0.5 $\varphi(x) = x - 2\,\text{sen}(x) + 0.5e^x$	x0 = 0.5	x0 = 0; x1 = 1
xbarra	0.37055779	0.37056282	0.370556114	0.370558084	0.370558098
f(xbarra)	-0.000000708921	0.0000107629384	-0.000004519	-0.000000027834957228 02	0.0000000046 2871629914
Erro em x	0.000000763	0.370562817	0.000011528	0.00013863	0.0000057404
Número de Iterações	18	7	6	3	7

Exemplo 21

$$f(x) = x\log(x) - 1$$

	Bissecção	Posição Falsa	Método do Ponto Fixo	Newton	Secante
Dados iniciais	[2, 3]	[2, 3]	x0 = 2.5 $\varphi(x) = x - 1.3(x\log x - 1)$	x0 = 2.5	x0 = 2.3; x1 = 2.7
xbarra	2.50618413	2.50618403	2.506184170	2.506184146	2.506184181
f(xbarra)	-0.000000012234	-0.000000099 27991906356	0.000000020 5082655302 4	0.000000000 0013780088 2	0.0000000295 0844302241
Erro em x	0.0000000059605	0.493814420	0.000000038426	0.000000398 79	0.0000080561
Número de Iterações	24	5	6	2	3

Exemplo 22

Consideremos

$$f(x) = x^3 - 3.5x^2 + 4x - 1.5 = (x-1)^2 (x - 1.5).$$

	Teste 1	Teste 2	Teste 3
x0	0.5	1.33333	1.33334
xbarra	0.999776466	0.999709005	1.500000005
f(xbarra)	-0.000000024994	-0.00000004236363282632	0.00000000122412771208
Erro em x	0.000000022491	0.000000029079	0.000000035082
Número de iterações	12	35	27

	Teste 4 - Método da Bissecção com Método de Newton	
	Bissecção	Newton
x0	[0.5, 2]	1.53125000
xbarra	1.53125000	1.500000001
f(xbarra)	0.008819580078	0.000000000227
Erro em x	0.01	0.001
Número de iterações	3	2

Análise dos resultados obtidos

Com os resultados obtidos ao se executar os algoritmos de obtenção de zeros de funções reais, escritos em Python, podemos destacar alguns pontos importantes:

- No caso do método de Newton, se já tivermos a função da derivada pronta, podemos ter uma leve vantagem na quantidade de passos e tempo de execução do algoritmo. Entretanto, temos que considerar aspectos como, caso seja necessário calcular a derivada separadamente, o trabalho pode ser ainda maior se comparado com os outros métodos.
- Além disso, também é necessário levar em consideração a distância entre o valor inicial x0 e a raiz. Isso, aliás, pode ser observado no exemplo 19, especialmente nos métodos de Newton e da Secante (de maneira análoga).

- É importante levar em consideração que o "Erro em x" é um critério importante para se obter o zero da função, que é usado como critério de parada para todos os algoritmos. Os resultados finais podem variar devido a aspectos computacionais, especialmente quanto à linguagem Python e seus mecanismos, bem como da arquitetura do computador utilizado.

- Cabe destacar que é possível se mesclar os algoritmos, a fim de se obter melhores resultados e desempenho. Isso foi observado na aplicação do exemplo 22, em que utilizamos o método da Bissecção para se obter uma aproximação inicial e, posteriormente, utilizado no método de Newton, que exige um valor inicial para obtenção do zero da função.

Acesso direto aos algoritmos separados por Exemplo

Os algoritmos utilizados aqui podem ser acessados separadamente via GitHub Gist:

- Cada Gist difere essencialmente nas funções utilizadas, valores iniciais e valor do Erro.
- Os algoritmos seguem a mesma estrutura.
- O Teste 4 do Exemplo 21 mescla o método da Bissecção e o de Newton.

Exemplo 18:
https://gist.github.com/matheusLazaroCC-UFG/f11b2fb9fb16e94243eb09b2bd246b75

Exemplo 19:
https://gist.github.com/matheusLazaroCC-UFG/2cc501fef252a7564e5552f03edc8312

Exemplo 20
https://gist.github.com/matheusLazaroCC-UFG/b6ede23681536c0e5d0eca59eabb85f8

Exemplo 21
https://gist.github.com/matheusLazaroCC-UFG/c111c759d98014a9f664487ab55f3902

Exemplo 22
https://gist.github.com/matheusLazaroCC-UFG/3bcd6a780aaab50134e8564418678ac0