

Universidade Federal do Rio Grande do Norte
Instituto Metr pole Digital
Bacharelado em Tecnologia da Informa  o
C culo Numerico para Ci ncia da Computa  o

Relat rio Tarefa 2

Discente: Gabriel Martins Sp nola

Docente: Dr. Rafael Beserra Gomes

Natal, RN
2020

Introdução

Tarefa realizada pela matéria de Cálculo Numérico onde por meio da linguagem python foi criado algoritmos para os métodos de aproximação de raízes. O resultado dos algoritmos podem ser visto por meio da ferramenta GNUPLOT onde plotamos gráficos dos pontos.

Desenvolvimento

Exercício 1

O exercício 1 nos dá a função $f(x) = x^3 - 1.7x^2 - 12.78x - 10.08$. Utilizando-a precisamos utilizar os métodos iterativos para calcularmos uma aproximação ao valor da raiz dessa função. O primeiro método implementado foi o método da bisseção, onde o código podemos ver abaixo

```
1 def bissecao(a, b, TOL):
2     i = 1
3     N = 500
4     fa = a**3 - 1.7*a**2 - 12.27*a - 10.08
5     while (i <= N):
6         p = a + (b-a)/2
7         fp = p**3 - 1.7*p**2 - 12.27*p - 10.08
8         if((fp == 0) or ((b-a)/2 < TOL)):
9             return p;
10        i = i+1
11        if(fa*fp>0):
12            a = p
13            fa = fp
14        else:
15            b = p
```

O próximo algoritmo implementado foi o método da iteração por ponto fixo. Segue o código

```
1 def itrPontoFixo(x, a):
2     for i in range(100):
3         x = x-a*(x**3 - 1.7*x*x - 12.78*x - 10.08)
4     return x
```

Para visualizar-mos o resultado desses 2 algoritmos foi plotado os pontos gerados pelos algoritmos onde no método da bisseção estimamos as 3 raízes passando os parâmetros que pode ser visualizado no código abaixo

```
17 arq = open("ptsBissecao.pts", "w")
18 erro = 0.000000005
19 arq.write(str(bissecao(-2.2, -2, erro)) + " " + "0\n")
20 arq.write(str(bissecao(-1.10, -0.9, erro)) + " " + "0\n")
21 arq.write(str(bissecao(4.65, 4.90, erro)) + " " + "0\n")
```

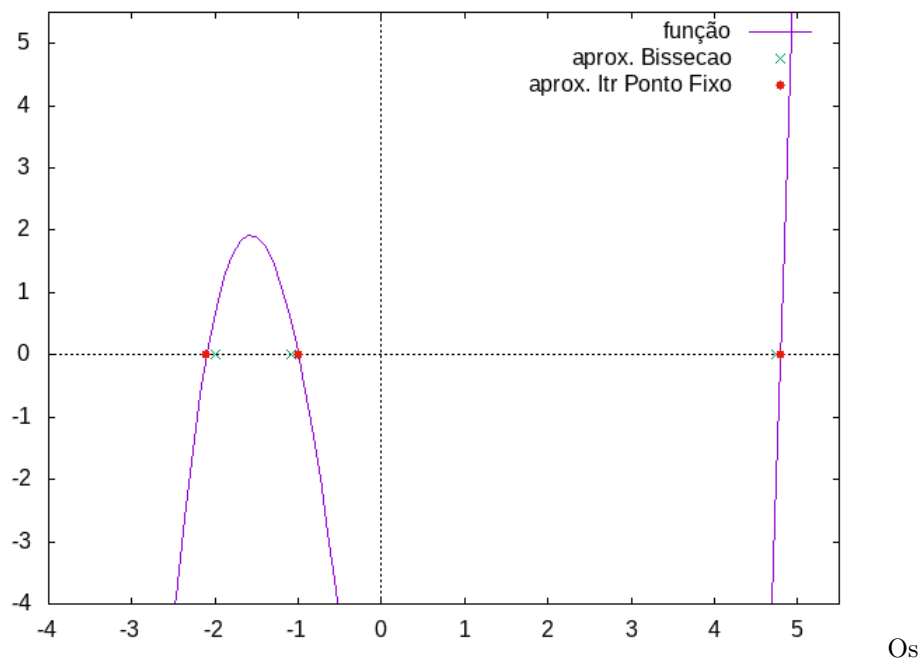
O algoritmo que utiliza o método da iteração por ponto fixo estimou as raízes com os parâmetros

```

5  arq = open("ptsItrPontoFixo.pts", "w")
6  arq.write(str(itrPontoFixo(-2.50, 0.005)) + " 0\n")
7  arq.write(str(itrPontoFixo(-1, 0.0005)) + " 0\n")
8  arq.write(str(itrPontoFixo(-0.7, 0.005)) + " 0\n")

```

O resultado dos pontos estimados com os parametros acima pode ser visto abaixo:



Próximos algoritmos implementam os métodos de Newton-Raphson, secante e regula falsi. Primeiro, vamos visualizar o método do do Newton:

```

1  def newthonRaphson(x):
2      for n in range(0, 100):
3          x = x - ((x**3 - 1.7*(x**2) - 12.78*x - 10.08)/(3*x**2 - 3.4*x - 12.78))
4      return x

```

Continuando, Podemos ver a implementação do método da secante:

```

1  def secante(x, y):
2      #x = Xn
3      #y = Xn-1
4      for i in range(0, 5):
5          aux = x
6          x = (y*f(x) - x*f(y))/(f(x) - f(y))
7          y = aux
8      return x

```

```

9
10 def f(x):
11     return x**3 - 1.7*(x**2) - 12.78*x - 10.08

```

Finalmente, a implementação do método regula falsi:

```

1 def regulaFalsi(a, b):
2     c = 0
3     for i in range(0, 20):
4         c = b - (f(b)*(b-a))/(f(b) - f(a))
5         if ((f(a) > 0 and f(c) > 0) or (f(a) < 0 and f(c) < 0)):
6             a = c
7         else:
8             b = c
9     return c
10
11 def f(x):
12     return x**3-1.7*(x**2)-12.78*x-10.08

```

Estimamos as 3 raízes passando os valores a seguir como parâmetro dos métodos: Newton-Raphson:

```

7 arq.write(str(newthonRaphson(-3)) + " 0\n")
8 arq.write(str(newthonRaphson(-1.5)) + " 0\n")
9 arq.write(str(newthonRaphson(4)) + " 0 \n")

```

Secante:

```

14 arq.write(str(secante(-3, -3.2)) + " 0\n")
15 arq.write(str(secante(-1.4, -1.5)) + " 0\n")
16 arq.write(str(secante(3, 2)) + " 0\n")

```

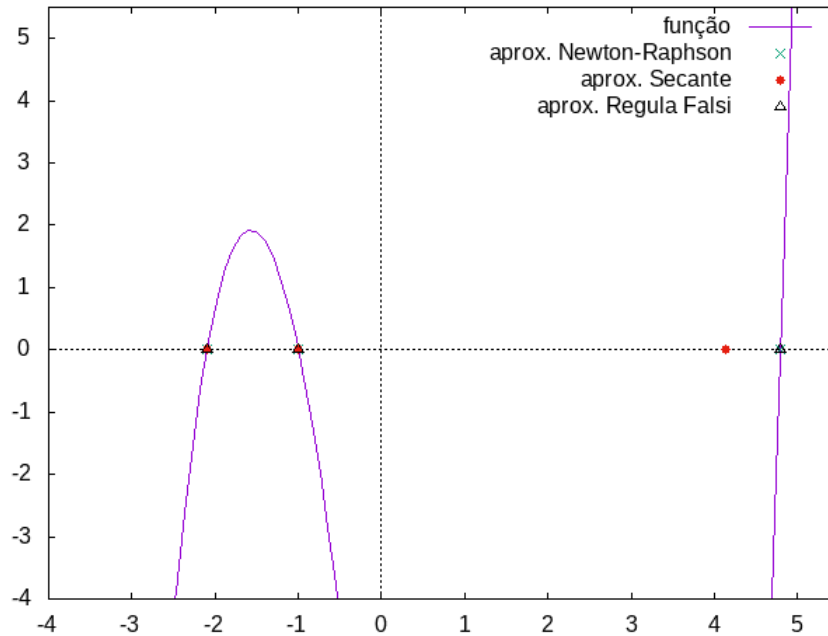
Regula Falsi:

```

15 arq.write(str(regulaFalsi(-2.2, -1.9)) + " 0\n")
16 arq.write(str(regulaFalsi(-1.5, 1)) + " 0\n")
17 arq.write(str(regulaFalsi(5, 1)) + " 0\n")

```

Novamente podemos ver o resultado com o gráfico plotado no gnuplot abaixo:



Exercício 2

Comparando os métodos que foram mostrados acima pôde ser observado que os algoritmos que Newton-Raphson, Secante e Regula Falsi foram os que apresentaram melhor desempenho, onde o método Newton-Raphson precisou apenas de 3 iterações para apresentar uma aproximação razoável para as 3 raízes, e os métodos da secante e Regula Falsi 5 iterações para apresentar um resultado satisfatório. Em contrapartida, o método da iteração por ponto fixo foi o algoritmo que apresentou pior desempenho, onde foram necessários 100 iterações para conseguir uma aproximação satisfatória para a raiz 4.8, e por fim, o algoritmo da bisseção precisou de exatas 26 iterações para apresentar um valor razoavelmente próximo para cada uma das raízes.

Exercício 3

Para a resolução dessa questão era necessário achar um valor de t (tempo) que fosse raiz da equação dada. Para isso, foi feita uma aproximação com o método da aproximação de raízes feito por Newton e Raphson. Primeiro, substituímos todos os valores que foram dados na questão, deixando a função somente em função de t

$$s(t) = 300 - \frac{0.25 * 32.17}{0.1} * t + \frac{0.25^2 * 32.17}{0.1^2} * (1 - e^{\frac{(-0.1 * t)}{0.25}})$$

. Após isso, derivamos essa função, com ajuda de calculadoras de derivadas online, para ser possível a realização do método anteriormente dito.

$$s'(t) = -80.425 + 80.425 * e^{\frac{-0.1*t}{0.25}}$$

Podemos ver a implementação da resolução da questão abaixo, ela contém a função original em função de t, e também a função derivada em função de t.

```

1 def newthonRaphson(t):
2     while True:
3         t = t - (s(t)/sd(t))
4         if s(t) <= 0.000001 and s(t) >= -0.000001:
5             return t
6
7 def s(t):
8     return 300 - ((0.25*32.17)/0.1)*t + (((0.25**2)*32.17)/0.1**2)*(1 - 2.71828182846**((-0.1*t)/0.25))
9
10 def sd(t):
11     return -80.425 + 80.425*2.71828182846**((-0.1*t)/0.25)

```

Executando o algoritmo acima, estimamos o ponto t, e com a condição "if s(t) <= 0.000001 and s(t) >= -0.000001:" garantimos que a aproximação tem precisão de 0,000001s. O ponto estimado pode ser visto abaixo

```

gabriel@gabriel-Nitro-AN515-51: ~/Documentos/calcul-numerico/...
gabriel@gabriel-Nitro-AN515-51:~/Documentos/calcul-numerico/tarefa2$ python newton
RaphsonQ3.py
6.00372631275
gabriel@gabriel-Nitro-AN515-51:~/Documentos/calcul-numerico/tarefa2$

```

Exercício 4

O maior desafio da tarefa até então, pois não foi dado nenhuma fórmula previamente, na verdade era necessário achar a formula através das informações que foi dada. Para começar a resolver a questão, precisamos primeiramente observar a imagem para entender nosso problema. Foi notado que era necessário utilizar conceitos de semelhança de triângulos e o Teorema de Pitágoras. Após muitas substituições dos valores utilizando Pitágoras e semelhança de triângulos, deduzimos uma fórmula em função de L:

$$f(L) = L^2 + \frac{25600}{\left(\frac{\left(\frac{240}{\sqrt{900-L^2}}\right)L}{30} - L\right)^2} - 20^2$$

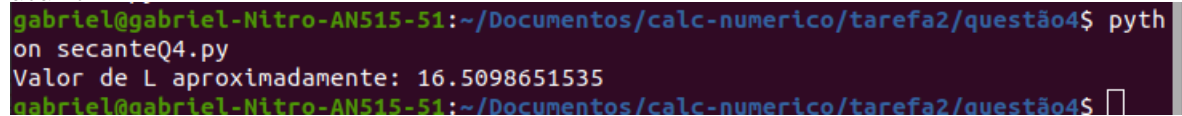
Com a fórmula acima, implementamos o algoritmo da secante para encontrarmos a raiz dessa equação

```

1 import math
2 def secante(x, y):
3     #x = Xn
4     #y = Xn-1
5     for i in range(6):
6         aux = x
7         x = (y*f(x) - x*f(y))/(f(x) - f(y))
8         y = aux
9     return x
10 def f(x):
11     return x**2 + 25600/(((240/math.sqrt(900 - x**2)*x)/30) - x)**2 - 20**2
12
13 print("Valor de L aproximadamente: " + str(secante(5, 4)))

```

Executando o algoritmo achamos uma aproximação para L com o valor abaixo



```

gabriel@gabriel-Nitro-AN515-51:~/Documentos/calc-numerico/tarefa2/questão4$ python secanteQ4.py
Valor de L aproximadamente: 16.5098651535
gabriel@gabriel-Nitro-AN515-51:~/Documentos/calc-numerico/tarefa2/questão4$

```

Nota: o algoritmo consegue executar o algoritmo com apenas 6 iterações com o parâmetro informado, após isso, o valor para de se aproximar e depois o código quebra, eu acredito que o problema está acontecendo por causa da função "math.sqrt" pois quando o código quebra, informa que é problema de domínio nessa função.

Conclusão

A tarefa feita tinha por finalidade a utilização de técnicas de encontrar aproximações quase que perfeitas para achar o resultado das raízes das funções. Tais técnicas ajudam a solucionar problemas reais como por exemplo a questão 4 dessa tarefa. Aprender essas técnicas serão essenciais para a continuação do semestre na disciplina de cálculo numérico. Além disso, foi possível rever e relembrar de assuntos de outras disciplinas como física na questão 3 e semelhança de triângulos e o Teorema de Pitágoras na Questão 4. Também foi possível observar qual dos métodos são mais eficaz para a resolução desses problemas.