Projeto Prático

SME0206 -
Fundamentos de Análise Numérica

Universidade de São Paulo (USP)

Aluno: Caio Assumpção Rezzadori

Nº USP: 11810481

1 Questões teóricas

1.1 Metodo de Newton

Calcularemos o método de Householder, dado por:

$$x_{k+1} = x_k + d\frac{(1/f)^{(d-1)}(x_k)}{(1/f)^{(d)}(x_k)}$$

para $d = 1$.

Sendo assim, temos:

$$x_{k+1} = x_k + \frac{(1/f)^{(0)}(x_k)}{(1/f)^{(1)}(x_k)}$$

É trivial que $(1/f)^{(0)}(x) = 1/f(x)$. Todavia, calculemos $(1/f)^{(1)}(x)$:

Pela regra da cadeia, temos:

$$(1/f)^{(1)}(x) = \frac{d(1/f(x))}{df(x)} \cdot \frac{df(x)}{dx} = -\frac{1}{[f(x)]^2} \cdot f'(x)$$

$$\Rightarrow x_{k+1} = x_k + \frac{1/f(x_k)}{-\frac{1}{[f(x_k)]^2} \cdot f'(x_k)}$$

$$\therefore x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Que é o método de Newton.

1.2 Método de Halley

Façamos agora o mesmo que fizemos na questão anterior, mas agora para $d = 2$.

Sendo assim, temos:

$$x_{k+1} = x_k + 2\frac{(1/f)^{(1)}(x_k)}{(1/f)^{(2)}(x_k)}$$

Como já calculado na questão anterior:

$$(1/f)^{(1)}(x) = -\frac{f'(x)}{[f(x)]^2}$$

Calculando $(1/f)^{(2)}$:

$$(1/f)^{(2)}(x) = -(\frac{f'(x)}{[f(x)]^2})^{(1)}$$

Utilizando a regra do quociente:

$$(1/f)^{(2)}(x) = -(\frac{f''(x)\cdot[f(x)]^2 - 2f'(x)\cdot f(x)\cdot f'(x)}{[f(x)]^4})$$

$$= -\frac{f''(x)\cdot f(x) - 2[f'(x)]^2}{[f(x)]^3}$$

$$\Rightarrow x_{k+1} = x_k + 2\frac{-\frac{f'(x_k)}{[f(x_k)]^2}}{(-\frac{f''(x_k)\cdot f(x_k) - 2[f'(x_k)]^2}{[f(x_k)]^3})}$$

$$= x_k + 2\frac{f'(x_k)\cdot f(x_k)}{f''(x_k)\cdot f(x_k) - 2f'(x_k)}$$

$$\therefore x_{k+1} = x_k - \frac{f'(x_k)\cdot f(x_k)}{[f'(x_k)]^2 - \frac{1}{2}f''(x_k)\cdot f(x_k)}$$

Que é o método de Halley.

2 Implementação

As implementações de todos os métodos exigidos no projeto foram feitas na linguagem de programação *Python 3.0*, na plataforma *Jupyter Notebook*. Os arquivos *.html* e *.ipynp* estão disponíveis no arquivo *.zip* que inclui esse relatório.

2.1 Método da bissecção

```
1  #f: Funcao estudada
2  #a0: Valor a esquerda na reta real inicial
3  #b0: Valor a direita na reta real inicial
4  #n: Numero de iteracoes maximo
5  #tol: Tolerancia do erro
6
7  def bissec(f, a0, b0, n, tol):
8      a = [a0]
9      b = [b0]
10     i = 0
11
12     while((i < n) & (abs(a[i] - b[i]) / 2 >tol) & (f(a[i]) * f(b[i
       ]) != 0 )):
13
14         if(f(a[i]) * f((a[i] + b[i]) / 2) < 0):
15             a.append(a[i])
16             b.append((a[i] + b[i]) / 2)
17
18         else:
19             a.append((a[i] + b[i]) / 2)
20             b.append(b[i])
21
22         i += 1
23
24     if(i == n):
25         print('O calculo falhou(Bisseccao)')
26
27     return(a, b)
```

Como é possível observar na linha 12, o critério de parada do algoritmo são um dos três:

1. Quando o contador 'i' atinge o número máximo de iterações 'n';
2. Quando a metade da diferença dos dois números da última iteração é menor ou igual à tolerância 'tol';
3. Quando a função estudada 'f' é zero em um dos dois pontos da última iteração.

Dessa forma, analisa-se o sinal da função para saber quais valores devem ser concatenados aos vetores (a ou b) de iterações. Caso a função tenha o mesmo sinal no meio do intervalo que na extremidade esquerda dele, o meio do intervalo da última iteração passa a ser a nova extremidade esquerda do intervalo. Caso tenha sinal oposto, o meio do intervalo da última iteração passa a ser a nova

extremidade direita do intervalo. Isso está explicitado a partir da linha 14 até a linha 20.

Após isso, inicia-se uma nova iteração, como mostra a linha 22.

Por fim, antes de retornar os vetores de iterações(linha 27), o programa analisa se o cálculo foi bem sucedido por meio da contagem de iterações. Caso ela tenha atingido 'n', significa que o programa encerrou antes de atingir a tolerância desejada. Isso está nas linhas 24 e 25.

2.2 Método da secante

```
1  #f: Funcao estudada
2  #x0: Primeiro ponto da reta secante inicial
3  #x1: Segundo ponto da reta secante inicial
4  #n: Numero de iteracoes maximo
5  #tol: Tolerancia do erro
6
7  def sec(f, x0, x1, n, tol):
8
9      x = [x0, x1]
10     i = 1
11
12     while((i < n) & (abs(x[i] - x[i - 1])>tol) & (f(x[i - 1]) != 0)
        & (f(x[i]) != 0) & ((f(x[i]) - f(x[i - 1])) != 0)):
13
14         x.append(x[i] - (f(x[i]) * (x[i] - x[i - 1]))/(f(x[i]) - f(
       x[i - 1])))
15
16         i += 1
17
18     if(i == n):
19         print('O calculo falhou(Secante)')
20
21     return(x)
```

Como é possível observar na linha 12, o critério de parada do algoritmo são um dos quatro:

1. Quando o contador 'i' atinge o número máximo de iterações 'n';
2. Quando a diferença entre as duas últimas iterações é menor ou igual à tolerância 'tol';
3. Quando a função estudada 'f' é zero na última ou na penúltima iteração.
4. Quando a diferença da função na última iteração e na penúltima iteração é nula (problema com divisão por 0 no algoritmo).

Feito isso, o método da secante é aplicado na linha 14 seguindo a fórmula:

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

Após isso, inicia-se uma nova iteração, como mostra a linha 16.

Por fim, antes de retornar o vetor de iterações(linha 21), o programa analisa se o cálculo foi bem sucedido por meio da contagem de iterações. Caso ela tenha atingido 'n', significa que o programa encerrou antes de atingir a tolerância desejada. Isso está nas linhas 18 e 19.

2.3 Método de Newton

```
1  #f: Funcao estudada
2  #df: Primeira derivada da funcao estudada
3  #x0: Primeiro ponto da iteracao
4  #n: Numero de iteracoes maximo
5  #tol: Tolerancia do erro
6
7  def newton(f, df, x0, n, tol):
8
9      x = [x0]
10     i = 2
11
12     if(df(x[0]) == 0):
13         print('O calculo falhou(Newton)')
14         return(x)
15
16     x.append(x[0] - f(x[0]) / df(x[0]))
17
18     while((i < n) & (abs(x[i - 1] - x[i - 2]) > tol) & (f(x[i - 1])
        != 0)):
19
20         if(df(x[i - 1]) == 0):
21             print('O calculo falhou(Newton)')
22             return(x)
23
24         x.append(x[i - 1] - f(x[i - 1]) / df(x[i - 1]))
25
26         i += 1
27
28     if(i == n):
29         print('O calculo falhou(Newton)')
30     return(x)
```

Nesse algoritmo, a primeira iteração é feita separada das demais como pode ser visto a partir da linha 12 até a linha 16, uma vez que o algoritmo precisa de apenas 1 ponto inicial para inciar, mas que ele depende de dois para conferir se as duas últimas iteraões estão fora ou dentro da tolerância.

Como é possível observar na linha 18, o critério de parada do algoritmo são um dos três:

1. Quando o contador 'i' atinge o número máximo de iterações 'n';
2. Quando a diferença entre as duas últimas iterações é menor ou igual à tolerância 'tol';

3. Quando a função estudada 'f' é zero na última iteração.

Além disso, existe outro critério de parada dentro do *loop*, como pode ser visto na linha 20(na primeira iteração foi aplicado na linha 12). Ele serve para caso a derivada da função na penúltima iteração seja zero e ocasione erro no cálculo na próxima iteração(problema com divisão por 0 no algoritmo).

Feito isso, o método de Newton é aplicado na linha 24 seguindo a fórmula:

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

Após isso, inicia-se uma nova iteração, como mostra a linha 26.

Por fim, antes de retornar o vetor de iterações(linha 30), o programa analisa se o cálculo foi bem sucedido por meio da contagem de iterações. Caso ela tenha atingido 'n', significa que o programa encerrou antes de atingir a tolerância desejada. Isso está nas linhas 28 e 29.

2.4 Método de Halley

```
1  #f: Funcao estudada
2  #df: Primeira derivada da funcao estudada
3  #d2f: Segunda derivada da funcao estudada
4  #x0: Primeiro ponto da iteracao
5  #n: Numero de iteracoes maximo
6  #tol: Tolerancia do erro
7
8  def halley(f, df, d2f, x0, n, tol):
9
10     x = [x0]
11     i = 2
12
13     if((df(x[0]) ** 2 - 1/2*(f(x[0]) * d2f(x[0])))== 0):
14         print('O calculo falhou(Halley)')
15         return(x)
16
17     x.append(x[0] - f(x[0]) / df(x[0]))
18
19     while((i < n) & (abs(x[i - 1] - x[i - 2]) > tol) & (f(x[i - 1])
        != 0)):
20
21         if((df(x[i - 1]) ** 2 - 1/2*(f(x[i - 1]) * d2f(x[i - 1])))
       == 0):
22             print('O calculo falhou(Halley)')
23             return(x)
24
25         x.append(x[i - 1] - (f(x[i - 1]) * df(x[i - 1])) / ((df(x[i
        - 1])) ** 2 - 1/2*(f(x[i - 1]) * d2f(x[i - 1]))))
26
27         i += 1
28
```

```
29      if(i == n):
30          print('O calculo falhou(Halley)')
31      return(x)
```

Assim como no método de Newton, a primeira iteração é feita separada das demais como pode ser visto a partir da linha 13 até a linha 17.

Como é possível observar na linha 19, o critério de parada do algoritmo são um dos três:

1. Quando o contador 'i' atinge o número máximo de iterações 'n';
2. Quando a diferença entre as duas últimas iterações é menor ou igual à tolerância 'tol';
3. Quando a função estudada 'f' é zero na núltima iteração.

Além disso, existe outro critério de parada dentro do *loop*, como pode ser visto na linha 21(na primeira iteração foi aplicado na linha 13). Ele serve para evitar que a expressão seja zero e ocasione erro no cálculo na próxima iteração(problema com divisão por 0 no algoritmo).

Feito isso, o método de Halley é aplicado na linha 25 seguindo a fórmula:

$$x_{k+1} = x_k - \frac{f'(x_k) \cdot f(x_k)}{[f'(x_k)]^2 - \frac{1}{2} f''(x_k) \cdot f(x_k)}$$

Após isso, inicia-se uma nova iteração, como mostra a linha 27.

Por fim, antes de retornar o vetor de iterações(linha 31), o programa analisa se o cálculo foi bem sucedido por meio da contagem de iterações. Caso ela tenha atingido 'n', significa que o programa encerrou antes de atingir a tolerância desejada. Isso está nas linhas 29 e 30.

3 Uso das implementações

Em todas as tabelas a primeira linha representa um valor escolhido para iniciar as iterações. No caso do método da bissecção, além da primeira linha que dá os dois primeiros pontos escolhidos, os valores da tabela são dados pela média aritmética das iterações, representando o centro do intervalo que as iterações geram.

3.1 1ª Equação

Precisamos achar a solução da seguinte equação:

$$x - \cos(x) = 0$$

Criando a função $f(x) = x - \cos(x)$ e aplicando os 4 métodos à ela, obtemos os seguintes resultados:

	Bissecção	Secante	Newton	Halley
0	[-1, 2]	[2]	[2]	[2]
1	[0.5]	[4]	[0.7345361688544632]	[0.7345361688544632]
2	[0.75]	[-0.15968742702372118]	[0.7390897242053693]	[0.739085122282906]
3	[0.625]	[0.6628149002247304]	[0.7390851332198145]	[0.7390851332151607]
4	[0.6875]	[0.7638231470880096]	[0.7390851332151607]	
5	[0.71875]	[0.738657163300381]		
6	[0.734375]	[0.7390828251975659]		
7	[0.7421875]	[0.7390851334333132]		
8	[0.73828125]	[0.7390851332151606]		
9	[0.740234375]	[0.7390851332151607]		
10	[0.7392578125]			
11	[0.73876953125]			
12	[0.739013671875]			
13	[0.7391357421875]			
14	[0.73907470703125]			
15	[0.739105224609375]			
16	[0.7390899658203125]			
17	[0.7390823364257812]			
18	[0.7390861511230469]			
19	[0.7390842437744141]			
20	[0.7390851974487305]			
21	[0.7390847206115723]			
22	[0.7390849590301514]			
23	[0.7390850782394409]			
24	[0.7390851378440857]			
25	[0.7390851080417633]			
26	[0.7390851229429245]			
27	[0.7390851303935051]			
28	[0.7390851341187954]			
29	[0.7390851322561502]			
30	[0.7390851331874728]			
31	[0.7390851336531341]			
32	[0.7390851334203035]			
33	[0.7390851333038881]			
34	[0.7390851332456805]			
35	[0.7390851332165767]			
36	[0.7390851332020247]			
37	[0.7390851332093007]			
38	[0.7390851332129387]			
39	[0.7390851332147577]			
40	[0.7390851332156672]			
41	[0.7390851332152124]			
42	[0.739085133214985]			
43	[0.7390851332150987]			
44	[0.7390851332151556]			
45	[0.739085133215184]			
46	[0.7390851332151698]			
47	[0.7390851332151627]			
48	[0.7390851332151591]			
49	[0.7390851332151609]			
50	[0.739085133215 16]			
51	[0.7390851332151605]			
52	[0.7390851332151607]			
53	[0.7390851332151608]			

3.2 2 ª Equação

Precisamos achar a solução da seguinte equação:

$$x^3 - 9x^2 + 27x - 27 = 0$$

Criando a função $g(x) = x^3 - 9x^2 + 27x - 27$ e aplicando os 4 métodos à ela, obtemos os seguintes resultados:

	Bissecção	Secante	Newton	Halley
0	[-1, 6]	[-1]	[2]	[0]
1	[4.25]	[6]	[2.3333333333333335]	[1.0]
2	[3.375]	[3.923076923076923]	[2.5555555555555562]	[2.0]
3	[2.9375]	[3.8607594936708853]	[2.7037037037037006]	[2.5]
4	[3.15625]	[3.5936449643236306]	[2.8024691358024763]	[2.75]
5	[3.046875]	[3.463240354700395]	[2.86831275720167]	[2.875]
6	[2.9921875]	[3.345180069226499]	[2.9122085048011885]	[2.9375]
7	[3.01953125]	[3.2618647588478917]	[2.941472336534064]	[2.96875]
8	[3.005859375]	[3.1972978505748495]	[2.9609815576892418]	[2.984375]
9	[2.9990234375]	[3.149045371555185]	[2.9739877051270898]	[2.9921875]
10	[3.00244140625]	[3.112479157590808]	[2.9826584700817413]	[2.99609375]
11	[3.000732421875]	[3.0849172578452455]	[2.9884389800442834]	[2.998046875]
12	[2.9998779296875]	[3.064099453131603]	[2.9922926533930707]	[2.9990234375]
13	[3.00030517578125]	[3.0483880264555703]	[2.994861768939271]	[2.99951171875]
14	[3.000091552734375]	[3.0365268136567183]	[2.9965745126702417]	[2.999755859375]
15	[2.9999847412109375]	[3.027573341732386]	[2.9977163415697174]	[2.9998779296875]
16	[3.0000381469726562]	[3.0208144807261923]	[2.998477560695579]	[2.99993896484375]
17		[3.015712392196908 5]	[2.9989850401062257]	[2.999969482421875]
18		[3.0118609323489673]	[2.999323358453619]	[2.9999847412109375]
19		[3.0089535533767098]	[2.9995489060054785]	
20		[3.006758837376582]	[2.999699265251814]	
21		[3.0051020955656096]	[2.9997995127752053]	
22		[3.0038514579308484]	[2.999866450950023]	
23		[3.002907379713535]	[2.999911070644811]	
24		[3.0021947155542485]	[2.999940420415613]	
25		[3.001656742012891]	[2.999960437236415]	
26		[3.0012506395999283]	[2.9999755692175545]	
27		[3.0009440759240538]	[2.9999874738141177]	
28		[3.000712662646149]	[3.000017663404747]	
29		[3.000537970158045]	[3.0000100720042853]	
30		[3.0004061173469236]		
31		[3.0003065703226843]		
32		[3.0002314467865214]		
33		[3.000174730142769]		
34		[3.000131835682086]		
35		[3.000099288128781]		
36		[3.000075139944071]		
37		[3.000056564417371]		
38		[3.000044180732904]		
39		[3.0000317970484374]		
40		[3.0000194133639706]		
41		[3.0000169366270772]		
42		[3.000021890100864]		
43		[3.0000268435746507]		

3.3 3ª Equação

Precisamos achar a solução da seguinte equação:

$$e^x - \cos(x) = 0$$

Criando a função $g(x) = e^x - \cos(x)$ e aplicando os 4 métodos à ela, obtemos os seguintes resultados:

	Bissecção	Secante	Newton	Halley
0	[-1, 2]	[1]	[2]	[2]
1	[-0.25]	[4]	[1.0594275224293475]	[1.0594275224293475]
2	[0.125]	[0.8768895984827112]	[0.421823404993508]	[0.16601696200794347]
3	[-0.0625]	[0.7738990336585623]	[0.10520322868058068]	[0.002621834111628807]
4	[0.03125]	[0.2924948669176779]	[0.00942171053193791]	[1.4924651341608614e-08]
5	[-0.015625]	[0.12064619698301429]	[8.739668434868189e-05]	[-6.131574890478334e-17]
6	[0.0078125]	[0.026278803867898312]	[7.637068073467336e-09]	[4.970655355773234e-17]
7	[-0.00390625]	[0.002824473528784517]	[1.010777617761133e-16]	
8	[0.001953125]	[7.24655071231697e-05]		
9	[-0.0009765625]	[2.0418394235941818e-07]		
10	[0.00048828125]	[1.47952822207517e-11]		
11	[-0.000244140625]	[9.135294519214567e-18]		
12	[0.0001220703125]			
13	[-6.103515625e-05]			
14	[3.0517578125e-05]			
15	[-1.52587890625e-05]			
16	[7.62939453125e-06]			
17	[-3.814697265625e-06]			
18	[1.9073486328125e-06]			
19	[-9.5367431640625e-07]			
20	[4.76837158203125e-07]			
21	[-2.384185791015625e-07]			
22	[1.1920928955078125e-07]			
23	[-5.960464477539063e-08]			
24	[2.9802322387695312e-08]			
25	[-1.4901161193847656e-08]			
26	[7.450580596923828e-09]			
27	[-3.725290298461914e-09]			
28	[1.862645149230957e-09]			
29	[-9.313225746154785e-10]			
30	[4.656612873077393e-10]			
31	[-2.328306436538696 3e-10]			
32	[1.1641532182693481e-10]			
33	[-5.820766091346741e-11]			
34	[2.9103830456733704e-11]			
35	[-1.4551915228366852e-11]			
36	[7.275957614183426e-12]			
37	[-3.637978807091713e-12]			
38	[1.8189894035458565e-12]			
39	[-9.094947017729282e-13]			
40	[4.547473508864641e-13]			
41	[-2.2737367544323206e-13]			
42	[1.1368683772161603e-13]			
43	[-5.684341886080802e-14]			
44	[2.842170943040401e-14]			
45	[-1.4210854715202004e-14]			
46	[7.105427357601002e-15]			
47	[-3.552713678800501e-15]			
48	[1.7763568394002505e-15]			
49	[-8.881784197001252e-16]			
50	[4.440892098500626e-16]			
51	[-2.220446049250313e-16]			
52	[1.1102230246251565e-16]			
53	[2.7755575615628914e-16]			

4 Estimativa da ordem de convergência

Sabe-se que a ordem de convergência p pode ser estimada por:

$$\frac{\log \frac{\epsilon_{k+1}}{\epsilon_k}}{\log \frac{\epsilon_k}{\epsilon_{k-1}}} = \frac{\ln \frac{\epsilon_{k+1}}{\epsilon_k}}{\ln \frac{\epsilon_k}{\epsilon_{k-1}}} \approx p$$

Dito isso, o seguinte programa foi feito para auxiliar no cálculo:

```
#Criando a funcao para a estimativa
def ordem(x, raiz):
    aux = []

    for i in range(2, len(x)):
        if ((abs(x[i - 1] - raiz) == 0) | (abs(x[i - 2] - raiz) ==
    0) | (np.log(abs(x[i - 1] - raiz)/abs(x[i - 2] - raiz)) == 0)):

            return(aux)

        aux.append(np.log(abs(x[i] - raiz)/abs(x[i - 1] - raiz))/ \
                np.log(abs(x[i - 1] - raiz)/abs(x[i - 2] - raiz)))
    return(aux)
```

Tal algoritmo foi aplicado para todos os vetores que aproximaram as 3 equações por meio dos 4 métodos. O resultado das estimativas para todas as iterações estão em diferentes tabelas por equação, e em diferentes colunas por método. Eles estão dispostos a partir da próxima página.

4.1 1ª Equação: $x^* = 0.7390851332151607$

	Bissecção	Secante	Newton	Halley
0	[35.327179654055485]	[-1.3563180509063164]	[1.2264774027320344]	[2.300337470312147]
1	[-0.7603018751155418]	[1.9140870666310255]	[2.0006953416353888]	
2	[-0.33820732692475886]	[0.4564479128367176]		
3	[1.1728234984235215]	[3.6032448680934306]		
4	[1.5712320797383341]	[1.287307857851756]		
5	[0.2854787463509074]	[1.7743253855383951]		
6	[3.234256231735667]	[1.5637676499310302]		
7	[-0.264652002004059]			
8	[-5.303307576000234]			
9	[-0.3181591727901333]			
10	[-2.4630388643675154]			
11	[0.23229103278397142]			
12	[4.578787241223198]			
13	[-0.41522198844061814]			
14	[-2.1722078304163612]			
15	[0.38382435525222797]			
16	[1.848048235014479]			
17	[0.13348054811292917]			
18	[19.47988674804202]			
19	[-0.7077298198276057]			
20	[-0.4636490435180523]			
21	[1.3372767199581295]			
22	[2.1457772843615674]			
23	[-0.6843472562682519]			
24	[-0.5292958740067045]			
25	[1.4415473619792851]			
26	[0.881228550086172]			
27	[-0.05223425507528485]			
28	[-59.60159215761488]			
29	[-0.7789100175189784]			
30	[-0.2746855459237045]			
31	[1.1050623790129144]			
32	[1.2732929256701908]			
33	[2.8772241521717814]			
34	[-0.725448931012233]			
35	[-0.3623768663094118]			
36	[1.2013520284906842]			
37	[1.7604745172383436]			
38	[-0.13386162050564324]			
39	[-9.982698054911745]			
40	[-0.5357660234813021]			
41	[-0.852589956419112]			
42	[2.3949243350020297]			
43	[-0.6084286759770127]			
44	[-0.6193014531279519]			
45	[1.6124695682084462]			
46	[0.16573669956458134]			
47	[7.742930488974065]			
48	[-0.5645750340535797]			
49	[-1.0]			
50				

4.2 2ª Equação: $x^* = 3$

	Bissecção	Secante	Newton	Halley
0	[-1.3139637480162702]	[4.097074893463176]	[1.0000000000000029]	[1.7095112913514545]
1	[1.4882059318866432]	[0.05930271354580376]	[0.9999999999999671]	[1.0]
2	[-0.5113915944693856]	[5.315412069697789]	[1.000000000000144]	[1.0]
3	[-1.3139637480162702]	[0.6675986522374209]	[1.0000000000002403]	[1.0]
4	[1.4882059318866432]	[1.1860396986384392]	[1.0000000000017615]	[1.0]
5	[-0.5113915944693856]	[0.9390108220794767]	[0.9999999999952759]	[1.0]
6	[-1.3139637480162702]	[1.0248902229487014]	[0.9999999999941304]	[1.0]
7	[1.4882059318866432]	[0.990640112599392]	[1.0000000000965197]	[1.0]
8	[-0.5113915944693856]	[1.003633538908858]	[0.9999999994874086]	[1.0]
9	[-1.3139637480162702]	[0.9986065412489565]	[0.9999999982462879]	[1.0]
10	[1.4882059318866432]	[1.0005369070910046]	[1.0000000118469037]	[1.0]
11	[-0.5113915944693856]	[0.9997935002229845]	[0.9999999953987793]	[1.0]
12	[-1.3139637480162702]	[1.000079477123442]	[1.000000026655818]	[1.0]
13	[1.4882059318866432]	[0.999969419910599]	[0.9999997410019924]	[1.0]
14	[-0.5113915944693856]	[1.0000117659117715]	[0.9999996588291027]	[1.0]
15		[0.9999954724932858]	[0.9999996997527747]	[1.0]
16		[1.000001739748194]	[0.9999949741261712]	[1.0]
17		[0.9999993421572103]	[1.0000079160764022]	
18		[1.0000002301537687]	[0.9999535414661543]	
19		[0.9999998212106413]	[1.0000765145415496]	
20		[1.0000001835054575]	[1.0019818309550608]	
21		[0.9999997720402946]	[1.0008487491195979]	
22		[1.0000009134887045]	[0.9850053753690621]	
23		[0.9999987044598128]	[1.022208831230006]	
24		[0.9999947068427346]	[1.1773704823222702]	
25		[1.0000208316255499]	[1.3858111834016211]	
26		[0.9999809430176692]	[-0.5144641105832243]	
27		[1.0000236186879772]	[-1.634501956472951]	
28		[0.9998430666652383]		
29		[1.0001210772122555]		
30		[0.9996522013799529]		
31		[1.0000357785244494]		
32		[1.002060377895062]		
33		[1.0065458304866084]		
34		[0.9828710351782418]		
35		[1.0190131226480243]		
36		[0.8701251480418939]		
37		[1.331148749165381]		
38		[1.500117960089156]		
39		[0.27661101876424354]		
40		[-1.8797631066103864]		
41		[0.7951168539264405]		

4.3 3ª Equação: $x^* = 0$

	Bissecção	Secante	Newton	Halley
0	[1.0]	[-1.0947664389636296]	[1.4492766979064213]	[2.9168084166528017]
1	[1.0]	[0.08232343099336736]	[1.5079780287239986]	[2.2381726924800276]
2	[1.0]	[7.787712351225309]	[1.7375170345448028]	[2.9112203454047614]
3	[1.0]	[0.9101644757107819]	[1.939723385790074]	[1.5990105313304377]
4	[1.0]	[1.721009113903347]	[1.996703143266221]	[0.010869875329842718]
5	[1.0]	[1.4634481253016827]	[1.9411453676828414]	
6	[1.0]	[1.6422613670728028]		
7	[1.0]	[1.6030298311321267]		
8	[1.0]	[1.6234196703035069]		
9	[1.0]	[1.4998920755309086]		
10	[1.0]			
11	[1.0]			
12	[1.0]			
13	[1.0]			
14	[1.0]			
15	[1.0]			
16	[1.0]			
17	[1.0]			
18	[1.0]			
19	[1.0]			
20	[1.0]			
21	[1.0]			
22	[1.0]			
23	[1.0]			
24	[1.0]			
25	[1.0]			
26	[1.0]			
27	[1.0]			
28	[1.0]			
29	[1.0]			
30	[1.0]			
31	[1.0]			
32	[1.0]			
33	[1.0]			
34	[1.0]			
35	[1.0]			
36	[1.0]			
37	[1.0]			
38	[1.0]			
39	[1.0]			
40	[1.0]			
41	[1.0]			
42	[1.0]			
43	[1.0]			
44	[1.0]			
45	[1.0]			
46	[1.0]			
47	[1.0]			
48	[1.0]			
49	[1.0]			
50	[1.0]			
51	[-1.3219280948873624]			

5 Conclusão

Teoricamente os métodos que utilizam menos iterações para convergir deveriam ser, em ordem decrescente: bissecção, secante, Newton e Halley. Percebe-se que essa ordem foi cumprida na 1ª e na 3ª equação. Todavia, na 2ª equação, o método da bissecção superou os demais. Isso provavelmente decorreu das escolhas dos pontos iniciais para cada iteração, fazendo com que a escolha dos pontos iniciais dos demais métodos, comparados com o da bisseção, não fossem tão eficientes para a convergência desejada.

Além disso, percebe-se que as estimativas das ordens de convergência oscilam muito conforme as iterações aumentam. Entretanto, na maioria dos casos, os casos que deveriam convergir com menos iterações, tiveram uma ordem superior aos outros métodos que deveriam convergir com mais iterações, principalmente nas últimas iterações de cada método. Isso corrobora com o esperado e mostra que, conforme as iterações tendem ao infinito, mais rápido o método com ordem de convergência superior aos demais, converge comparado aos outros.