TRABALHO 2

April 2, 2020

Análise Numérica (M2018)

Francisco Gonçalves
201604505

Departamento de Ciência de Computadores
Faculdade de Ciências de Universide do Porto

Contents

0.1 Primeiro Exercício

Pretende-se usar um método iterativo para determinar um valor aproximado de um zero de $F(x) = x^2 - x - sen(x + 0.15)$.

0.1.1 Alínea a)

Separem graficamente as raízes de $F(x) = 0$ *e determinem um intervalo* I *de amplitude* 10^{-1} *que contenha a maior delas.*

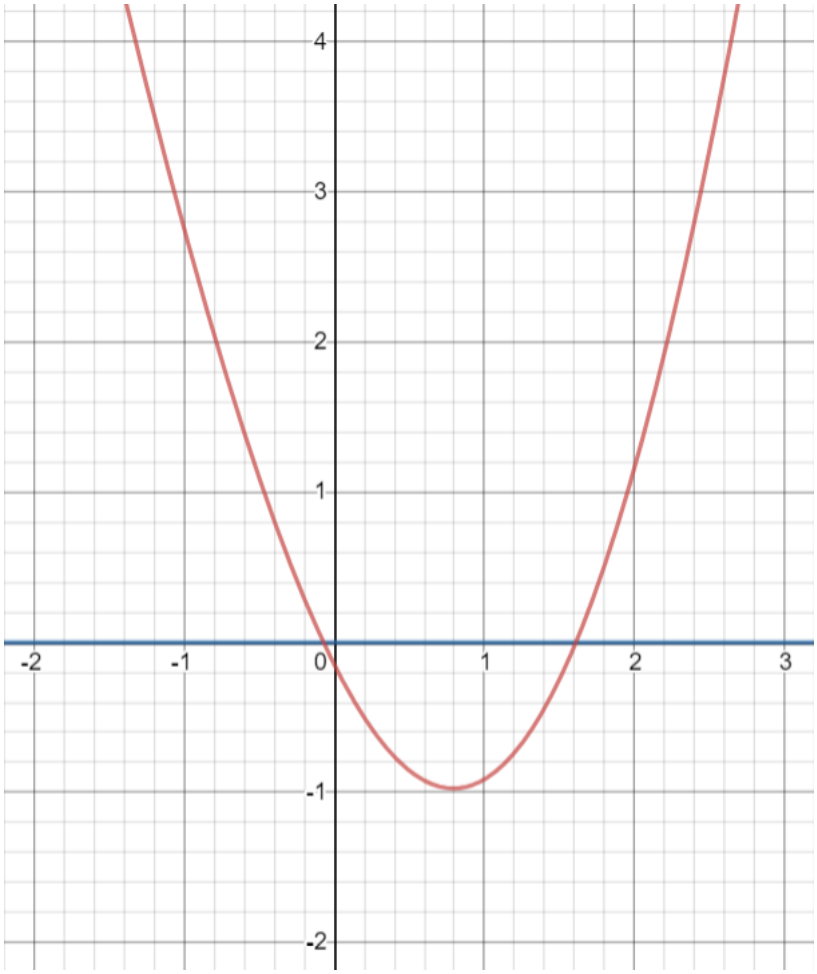

Figure 1: Representação gráfica da função F **(exercício 2.a)**

Usando o método gráfico para separar as raízes da função basta encontrar os pontos de interseção de $F(x)$ com o eixo $y = 0$. Assim sendo, os pontos de interseção encontrados são $(-0.0723, 0)$ e $(1.61, 0)$.

Consequentemente, de forma a obter o intervalo I, adiciona-se e subtrai-se 10^{-1} (0.1) ao maior valor de x dos pontos encontrados, para obter os extremos do intervalo pedido.

Assim, tem-se o intervalo $\mathbf{I} = \mathbf{[1.51, 1.71]}$.

0.1.2 Alínea b)

i. *Mostrem que as condições de aplicabilidade do método são satisfeitas no intervalo I.*

0.1.2.1 Método das bisseções sucessivas

Para que este método possa ser corretamente aplicado para obter a raíz da função é necessário verificar que satisfaz um certo conjunto de condições.

Primeiramente, é necessário verificar que F é contínua no intervalo I previamente calculado, e é possível verificar que é, pois não ocorre nenhuma interrupção, visível através da representação gráfica anterior.

Seguidamente, a seguinte condição deve ser verdadeira: $F(a)F(b) < 0$, sendo a,b os limites do intervalo I. Uma vez que a raíz será um ponto com ordenada $y = 0$, tem lógica que um dos limites seja negativo e o outro positivo, logo:

- $F(a) = 1.51^2 - 1.51 - sen(1.51 + 0.15) \approx -0.2259$
- $F(b) = 1.71^2 - 1.71 - sen(1.71 + 0.15) \approx 0.2556$

Por último, obviamente é necessário que $a < b$, visto que se trata de um intervalo, o menor valor é sempre o da esquerda. Sendo que, neste exemplo, $a = 1.51$ e $b = 1.71$, a condição comprova-se.

0.1.2.2 Método iterativo simples

Para que o método iterativo simples possa ser usado corretamente as seguintes condições devem ser verificadas. Primeiramente, a função F admite duas raízes, contudo, devido ao primeiro exercício, apenas foi considerada a maior ao determinar o intervalo I. Assim sendo, para determinar a função f é considerada apenas uma das equações, tendo-se:

- $f(x) = sin(x + 0.15)$

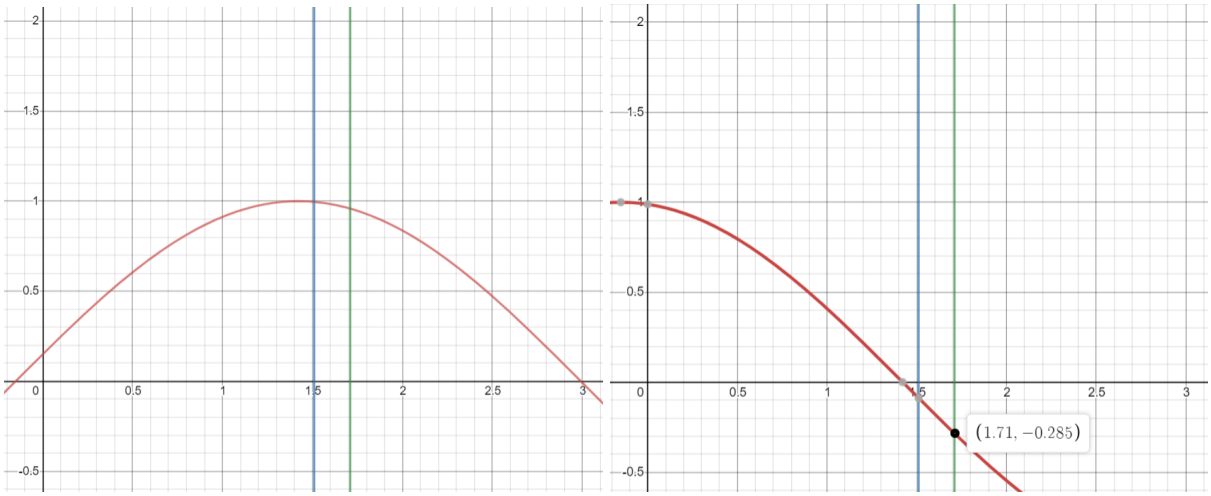

Figure 2: Representações gráficas das funções f (esquerda) e f' (direita)

A função f deve ser contínua no intervalo I. Visto que f é uma função sinusoidal, conclui-se que a condição é satisfeita.

Seguidamente, a seguinte propriedade deve existir: $f(x) \in [a,b], \forall x \in [a,b]$ ou $f([a,b]) \subseteq [a,b]$. Deste modo, sabendo que f é uma função estritamente decrescente no intervalo I, então esta condição é satisfeita para $[1.51, 1.71]$.

Ultimamente, a propriedade $|f(x_1) - f(x_2)| \leq L|x_1 - x_2|, \forall x_1, x_2 \in [a,b], 0 < L < 1$ tem de ser satisfeita. Ou seja, $|f'(x)| \leq L < 1, \forall x \in]a,b[$. Assim sendo, $f'(x) = cos(x + 0.15)$, logo tem-se $max|f'(x)| \leq 0.2852$, concluindo a análise da propriedade e permitindo a sua aplicabilidade.

0.1.2.3 Método de Newton

Para mostrar a aplicabilidade do Método de Newton no intervalo I, é necessário determinar a derivada da função F, assim como a sua segunda derivada, logo:

- $F(x) = x^2 - x - sen(x + 0.15)$
- $F'(x) = 2x - 1 - cos(x + 0.15)$
- $F''(x) = 2 + sen(x + 0.15)$

De seguida, é necessário verificar que F,F' e F'' são contínuas no intervalo I previamente calculado, e é possível verificar que o são, pois não ocorre nenhuma interrupção. É também necessário que $F(a)F(b) < 0$, sendo a, b os limites do intervalo I, ou seja, com **a=1.51** e **b=1.71**:

- $F(a) = 1.51^2 - 1.51 - sen(1.51 + 0.15) \approx -0.2259$
- $F(b) = 1.71^2 - 1.71 - sen(1.71 + 0.15) \approx 0.2556$
- $F(a)F(b) \approx -0.0577$ (logo verifica-se a condição)

Deve ainda ser respeitada a seguinte condição: $F'(x) \neq 0, \forall x \in [a, b]$. Por conseguinte, se assumirmos uma contrariedade e resolvermos $F'(x) = 0$, cuja única solução é $x = 0.793$ e, neste caso, $x \notin [a, b]$, logo esta condição também se verifica.

Seguidamente, deve ser tido em conta que $F''(x) \geq 0$ ou $F''(x) \leq 0, \forall x \in [a, b]$. A função F"(x) é um sinusoide, tendo sofrido deslocações na vertical e na horizontal. Contudo, todos os seus pontos apresentam y positivo, confirmando que respeita a primeira opção.

Por último, deve-se mostrar que $F(x_0)F''(x_0) > 0, x_0 \in [a, b]$. Sabendo que, $F(x_0)$ toma sempre um valor positivo então, para que a condição se verifique $F(x_0) > 0$, o que só é possível para $x_0 > 1.61$ e $x_0 \leq 1.71$.

Como todas estas propriedades se verificam, o método de Newton pode ser aplicado na função F de forma a convergir para a sua raíz no intervalo I.

0.1.2.4 Programa

ii. *Escrevam um programa que, usando este método, calcule um valor aproximado da raiz de* $F(x) = 0$ *que pertence a* I*, com erro absoluto (estimado) inferior a um valor* ϵ *dado e escreva o número de iterações foi necessário efetuar. Usem o vosso programa para calcular um valor aproximado daquela raiz com erro absoluto estimado inferior a* $5 * 10^{-8}$.

A seguinte implementação engloba todos os métodos necessários para a resolução do exercício:

Listing 1: *p1b.py* (PYTHON)

```
import math

def main():
    f = lambda x: x**2 - x - math.sin(x + 0.15)
    df = lambda x: 2*x - 1 - math.cos(x + 0.15)
    f2 = lambda x: math.sqrt(x + math.sin(x + 0.15))
    eps = epsilon()
    errorAbs = 5*10**(-8)
    intervalA = 1.51
    intervalB = 1.71

    print("METODO DAS BISSECOES SUCCESSIVAS:")
    FirstZeroSuccBi = SuccessiveBisectionMethod(f,intervalA,intervalB,eps)
    getApproximation(FirstZeroSuccBi,errorAbs)
    print("METODO ITERATIVO SIMPLES:")
    FirstZeroSimpIt = SimpleIterativeMethod(f2,intervalA,eps,100)
    getApproximation(FirstZeroSimpIt,errorAbs)
    print("METODO DE NEWTON:")
    FirstZeroNewton = NewtonMethod(f,df,intervalA,intervalB,eps)
    getApproximation(FirstZeroNewton,errorAbs)

def epsilon():
    eps = 1
    while (eps + 1 > 1):
        eps = eps / 2
    eps  = 2 * eps
    return eps

def getApproximation(zero,error):
    print("Valor aproximado -> %0.8f" % zero, "+/-", error)

main()
```

SuccessiveBisectionMethod() in *p1b.py* (PYTHON)

```python
def SuccessiveBisectionMethod(f,a,b,eps):
    i = 0
    x = a
    vfa = f(a)
    error = abs(b - a)
    while True:
        if (f(x) == 0):
            error = 0
        if (error <= eps):
            break
        x = (a + b) / 2
        i += 1
        if((f(x) * vfa) < 0):
            b = x
        else:
            a = x
        error = error / 2
        print(i, "Iteracao | x ->", x)
        print("[ %0.20f" % a, ", %0.20f" % b, "]")
    print("Valor de x -> %0.20f" % x, "| Valor de a ->", a, "| Valor de b ->", b)
    print("Erro: %0.20f" % error, "| N de iteracoes:", i)
    return x
```

NewtonMethod() in *p1b.py* (PYTHON)

```python
def NewtonMethod(f,df,a,b,eps):
    i = 0
    x = a
    while True:
        xn = x - (f(x) / df(x))
        i += 1
        print(i, "Iteracao | x ->", x)
        error = abs(xn - x)
        if (error <= eps):
            break
        x = xn
    print("Valor de x -> %0.20f" % x, "| Valor de a ->", a, "| Valor de b ->", b)
    print("Erro: %0.20f" % error, "| N de iteracoes:", i)
    return x
```

SimpleIterativeMethod() in *p1b.py* (PYTHON)

```
def SimpleIterativeMethod(f,x0,eps,nMax):
    x1 = f(x0)
    error = abs(x1 - x0)
    i = 0
    while (error > eps and i <= nMax):
        x0 = x1
        x1 = f(x0)
        error = abs(x1 - x0)
        i += 1
        print(i, "Iteracao | x ->", x1)
    if i > nMax:
        print("Nao foi possivel ao fim de %d iteracoes enocntrar a solucao com o erro
            pretendido." % nMax)
        return -500
    else:
        print("Valor de x -> %0.20f" % x1)
        print("Erro: %0.20f" % error, "| N de iteracoes:", i)
        return x1
```

0.1.3 Alínea c)

Comparem o comportamento dos três métodos neste problema.

Número de iterações	**Método das bisseções sucessivas** Valor de x	**Método de Newton** Valor de x	**Método iterativo simples** Valor de x
1	1.6099999999999999	1.6171194120931818	1.6030937438790354
2	1.66	1.6100534601769825	1.6082672461392857
3	1.6349999999999998	1.6100225565073827	1.6095794418279536
4	1.6224999999999998	1.6100225559160435	1.6099107954661334
5	1.61625	1.6100225559160437	1.6099943745251886
6	1.613125	-	1.610015450133565
7	1.6115624999999998	-	1.6100207642579267
8	1.6107812499999998	-	1.6100221041674623
9	1.610390625	-	1.6100224420122848
10	1.6101953125	-	1.6100225271963888
11	1.6100976562499998	-	1.61002254867468
12	1.6100488281249998	-	1.6100225540902087
13	1.6100244140625	-	1.6100225554556782
14	1.61001220703125	-	1.6100225557999672
15	1.610018310546875	-	1.6100225558867762
16	1.6100213623046875	-	1.610022555908664
17	1.6100228881835936	-	1.610022555914183
18	1.6100221252441407	-	1.6100225559155743
19	1.6100225067138672	-	1.6100225559159251
20	1.6100226974487304	-	1.6100225559160137
21	1.6100226020812989	-	1.610022555916036
22	1.610022554397583	-	1.6100225559160417
23	1.610022578239441	-	1.610022555916043
24	1.610022566318512	-	1.6100225559160435
25	1.6100225603580474	-	1.6100225559160435
[...]	[...]	[...]	[...]
47	1.6100225559160428	-	-
48	1.6100225559160435	-	-
49	1.6100225559160437	-	-
50	1.6100225559160437	-	-

Número de iterações	**Método das bisseções sucessivas** Intervalo [a,b]
1	[1.61000000 , 1.71000000]
2	[1.61000000 , 1.66000000]
3	[1.61000000 , 1.63500000]
4	[1.61000000 , 1.62250000]
5	[1.61000000 , 1.61625000]
6	[1.61000000 , 1.61312500]
7	[1.61000000 , 1.61156250]
8	[1.61000000 , 1.61078125]
9	[1.61000000 , 1.61039062]
10	[1.61000000 , 1.61019531]
11	[1.61000000 , 1.61009766]
12	[1.61000000 , 1.61004883]
13	[1.61000000 , 1.61002441]
14	[1.61001221 , 1.61002441]
15	[1.61001831 , 1.61002441]
16	[1.61002136 , 1.61002441]
17	[1.61002136 , 1.61002289]
18	[1.61002213 , 1.61002289]
19	[1.61002251 , 1.61002289]
20	[1.61002251 , 1.61002270]
21	[1.61002251 , 1.61002260]
22	[1.61002255 , 1.61002260]
23	[1.61002255 , 1.61002258]
24	[1.61002255 , 1.61002257]
25	[1.61002255 , 1.61002256]
[...]	[...]
46	[1.61002256 , 1.61002256]
47	[1.61002256 , 1.61002256]
48	[1.61002256 , 1.61002256]
49	[1.61002256 , 1.61002256]
50	[1.61002256 , 1.61002256]

Ao longo da execução deste método, o intervalo em que a raíz se encontra mais sendo encolhido, reduzindo o número de possibilidades. O programa termina quando o zero for encontrado ou quando o erro for demasiado reduzido para continuar.

Resultados Finais

```
# METODO DAS BISSECOES SUCCESSIVAS:
Valor final de x -> 1.61002255591604370721
Valor de a -> 1.6100225559160435
Valor de b -> 1.6100225559160437
Erro: 0.00000000000000017764 | N de iteracoes: 51
Valor aproximado -> 1.61002256 +/- 5e-08

# METODO ITERATIVO SIMPLES:
Valor final de x -> 1.61002255591604348517
Erro: 0.00000000000000000000 | N de iteracoes: 25
Valor aproximado -> 1.61002256 +/- 5e-08

# METODO DE NEWTON:
Valor final de x -> 1.61002255591604348517
Valor de a -> 1.51
Valor de b -> 1.71
Erro: 0.00000000000000022204 | N de iteracoes: 5
Valor aproximado -> 1.61002256 +/- 5e-08
```

Ao analisar os resultados obtidos, é possível confirmar que todos os métodos atingiram relativamente o mesmo valor, dependendo do erro a considerar. Para o erro absoluto em causa, o valor aproximado obtido pelos três métodos foi exatamente igual, embora o seu valor final, antes do cálculo da aproximação ser feito, mostre algumas irregularidades.

Também se pode afirmar que, dado que cada método usa um procedimento diferente, o Método de Newton, por exemplo, necessitou de menos iterações para convergir no valor da sua raíz, enquanto que o Método das Bisseções Sucessivas foi o mais demorado.

Comparando os três erros obtidos, é possível verificar que, para este exercício, embora a diferença seja incrivelmente mínima, o Método Iterativo Simples apresenta o menor valor, nem podendo ser representado com precisão usando 20 casas decimais. Assim, embora este método não tenha sido tão rápido como o Método de Newton em termos de iterações, conseguiu obter um valor mais preciso. Por contrariedade, o Método de Newton, embora o mais rápido, apresenta o maior erro dos três.

0.2 Segundo Exercício

0.2.1 Alínea a)

Mostre que a função $F(x) = 2 + 6x - 4x^2 + 0.5x^3$ *tem um zero no intervalo* $[3.5, 6.5]$.

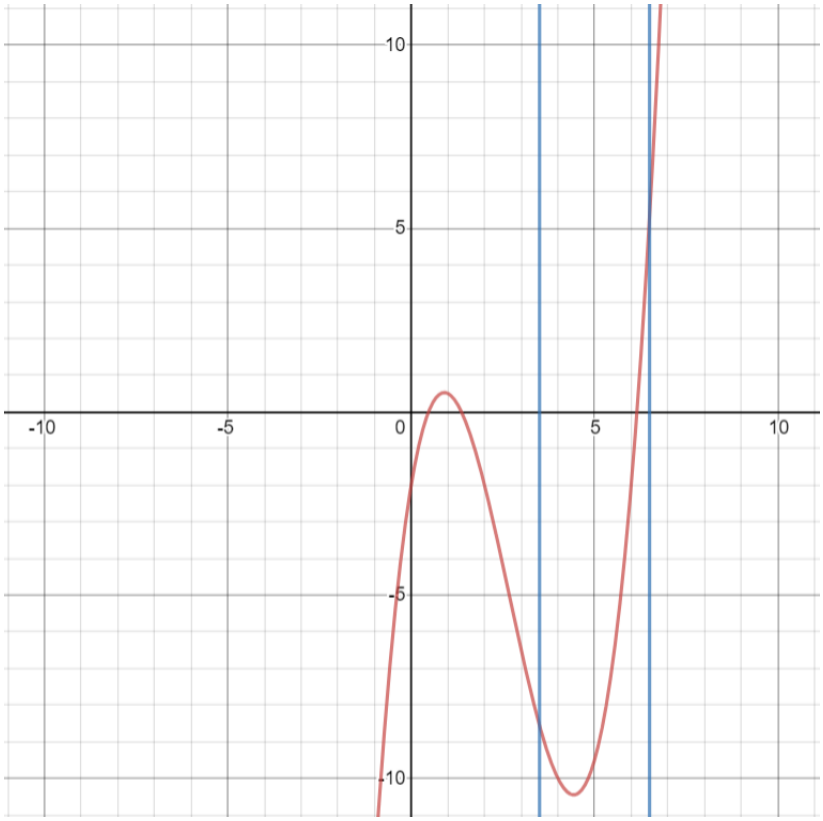

Figure 3: Representação gráfica da função F **(exercício 2.a)**

Tomando $F(x) = 0$, ou seja, $2 + 6x - 4x^2 + 0.5x^3 = 0$ e resolvendo para x, obtêm-se todas as soluções possíveis cujos pontos tenham ordenada $y = 0$:

- $x \approx 0.475$
- $x \approx 1.369$
- $x \approx 6.156$

Após restringir a função ao intervalo $[3.5, 6.5]$, verifica-se que existe apenas um ponto que respeite esse limite, sendo ele **(6.156,0)**, logo comprova-se a existência de um zero da função F nesse intervalo.

0.2.2 Alínea b)

Usem o vosso programa para aplicar o método de Newton (sem verificar se são satisfeitas as condições do método) para obter um valor aproximado daquele zero com erro absoluto estimado inferior a 10^{-6}*, considerando os seguintes valores para* x_0*:*

- $x_0 = 3.5$.
- $x_0 = 6.5$.
- $x_0 = 4.4$.

A seguinte implementação permite resolver para todos os valores de x_0 em simultâneo:

Listing 2: *p2b.py* (PYTHON)

```python
import math

def main():
    f = lambda x: -2 + 6*x - 4*x**2 + 0.5*x**3
    df = lambda x: 6 - 8*x + 1.5*x**2
    eps = epsilon()
    errorAbs = 10**(-6)
    intervalA = 3.5
    intervalB = 6.5
    x1 = 3.5
    x2 = 6.5
    x3 = 4.4

    print("X0 ->", x1, ":")
    FirstZeroNewtonX1 = NewtonMethod(f,df,x1,intervalB,eps)
    getApproximation(FirstZeroNewtonX1,errorAbs)
    print("X0 ->", x2, ":")
    FirstZeroNewtonX2 = NewtonMethod(f,df,x2,intervalB,eps)
    getApproximation(FirstZeroNewtonX2,errorAbs)
    print("X0 ->", x3, ":")
    FirstZeroNewtonX3 = NewtonMethod(f,df,x3,intervalB,eps)
    getApproximation(FirstZeroNewtonX3,errorAbs)

def epsilon():
    eps = 1
    while (eps + 1 > 1):
        eps = eps / 2
    eps  = 2 * eps
    return eps
```

NewtonMethod() / getApproximation() in *p2b.py* (PYTHON)

```
def NewtonMethod(f,df,a,b,eps):
    i = 0
    x = a
    while True:
        if(df(x) == 0):
            print("Impossivel encontrar uma solucao")
            return
        xn = x - (f(x) / df(x))
        error = abs(xn - x)
        x = xn
        i += 1
        print(i, "Iteracao | x ->", xx)
        if (error <= eps):
            break
    print("Valor de x -> %0.20f" % x, "| Valor de a ->", a, "| Valor de b ->", b)
    print("Erro: %0.20f" % error, "| N de iteracoes:", i)
    return x

def getApproximation(zero,error):
    print("Valor aproximado -> %0.6f" % zero, "+/-", error)

main()
```

Esta implementação do Método de Newton é semelhante à do programa anterior, tendo apenas sido feita reorganização e limpeza de dados, uma vez que estes mudaram com o exercício. Tudo o resto permaneceu inalterado. O número de casas decimais do erro absoluto ao efetuar a aproximação do valor de x também foi recalibrado.

0.2.3 Alínea c)

Comentem e justifiquem os resultados obtidos.

Após executar o programa *p2b.py*, são imprimidos os resultados para cada valor de x_0:

Número de iterações	$x_0 = 3.5$ **Valor de** x	$x_0 = 6.5$ **Valor de** x	$x_0 = 4.4$ **Valor de** x
1	1.1379310344827585	6.194244604316546	-60.90000000000135
2	1.4693221745502059	6.156866719227977	-39.74294512512855
3	1.3764715896011954	6.156325287500599	-25.653555579807282
4	1.3691509152320513	6.156325174658666	-16.28286182080067
5	1.3691023883254085	6.156325174658662	-10.067466823324072
6	1.3691023861848552	6.156325174658661	-5.96801390233247
7	1.369102386184855	6.156325174658662	-3.294140026526098
8	-	6.156325174658661	-1.586491421795651
9	-	-	-0.5368207740912427
10	-	-	0.0645665686675827
11	-	-	0.361328727676477
12	-	-	0.46137513541757913
13	-	-	0.47435367021897357
14	-	-	0.47457237726118506
15	-	-	0.47457243915647807
16	-	-	0.4745724391564829
17	-	-	0.47457243915648295

Resultados Finais

```
# X0 -> 3.5 :
Valor final de x -> 1.3691023861848524675 | Valor de a -> 3.5 | Valor de b -> 6.5
Erro: 0.0000000000000022204 | N de iteracoes: 7
Valor aproximado -> 1.369102 +/- 1e-06

# X0 -> 6.5 :
Valor final de x -> 6.15632517465866069273 | Valor de a -> 6.5 | Valor de b -> 6.5
Erro: 0.0000000000000088818 | N de iteracoes: 8
Valor aproximado -> 6.156325 +/- 1e-06

# X0 -> 4.4 :
Valor final de x -> 0.47457243915648289478 | Valor de a -> 4.4 | Valor de b -> 6.5
Erro: 0.0000000000000005551 | N de iteracoes: 17
Valor aproximado -> 0.474572 +/- 1e-06
```

Analisando os resultados obtidos, embora o limite máximo (b) seja o mesmo para todas as iterações de diferentes x_0 (a), o método encontrou 3 raízes, sendo os zeros visíveis na representação gráfica anterior da função F. Deste modo, o zero previamente encontrado, apenas pôde ser aproximado com $x_0 = 6.5$. É de notar que, sem verificar as condições de aplicabilidade do Método de Newton para esta função, toda a implementação está propensa a erros ou falhas quanto a encontrar uma solução, confirmando que o valor inicial é de extrema importância para obter a raíz correta no intervalo definido, visto que, existindo mais do que uma raíz, um dado valor irá apenas convergir para uma delas, o que está fortemente dependente da derivada da função em causa.

Os programas aqui exibidos assim como os reusltados obtidos podem ser consultados em detalhe aqui: `https://github.com/1Skkar1/Numerical-Analysis/tree/master/Trabalho2`