Universidade do Algarve

Faculdade de Ciências e Tecnologia

Licenciatura em Engenharia Informática

Análise Numérica I

Trabalho 2 – Equações Não Lineares

Método da Bissecção, Método de Newton e Método da Secante

Realizado por: João Andréz nº61066, Carlos Ferreira nº 71319, Diogo Freitas nº 90147, Diogo Carvalho nº 90247

Docente: Hermenegildo Borges de Oliveira

Ano Letivo: 2025/2026

Introdução

O presente trabalho foi desenvolvido no âmbito da unidade curricular Análise Numérica I e tem como objetivo principal a implementação de três programas que calculem valores aproximados de raízes de equações não lineares, e que são executados por um único que possibilita ao utilizador a escolha de qual dos três usar.

Os três programas visam implementar os Métodos da Bissecção, de Newton e da Secante, com alguns parâmetros definidos, no Método da Bissecção, o programa deve ser capaz de ir até às 20 iterações e de trabalhar com tolerância relativa inferior a 10^(-5), e em ambos os Métodos o de Newton e da Secante, devem ser capazes de ir até às 10 iterações e de trabalhar com tolerância relativa inferior a 10^(-9) .

Neste trabalho, irão estar presentes testes que verificam o funcionamento de todos os programas, bem como uma verificação de resultados utilizando o Geogebra.

Este trabalho, como um todo visa consolidar a utilização destes três métodos, Bissecção, Newton e Secante, com as suas condições de aplicabilidade, convergência e ordem, lidar com o erro relativo e garantir a robustez e eficiência dos programas.

Código

Main.py

```
# TRABALHO 2
# João Andréz - 61066
# Carlos Ferreira - 71319
# Diogo Freitas - 90147
# Diogo Carvalho - 90247

import bissection, newton, secant

if __name__ == '__main__':
    while(True):
        user_selection = str(input("Escolha um método:\n  1) M√©todo da Bisseção\n  2) Método de Newton\n  3) Método da Secante\n  4) Sair\n-->"))

        if user_selection == "1":
            bissection.userInput()
        elif user_selection == "2":
            newton.userInput()
        elif user_selection == "3":
            secant.userInput()
        elif user_selection == "4":
```

```
            break
        else:
            print("Selecione uma opção válida! (1,2,3 ou 4)")
```

bissection.py

```
# TRABALHO 2 - Método da Bisseção
# João Andréz - 61066
# Carlos Ferreira - 71319
# Diogo Freitas - 90147
# Diogo Carvalho - 90247

from sympy import * # type: ignore

MAX_ITERATIONS = 20
MAX_RELATIVE_TOLERANCE = 10**(-5)

# Function for finding f(x) = 0 where:
#  f: mathematical expression for the given function, where the
variable is x
# a: lower bound for x
# b: upper bound for x
# x: symbol for expression variable
# returns an aproximation of x where f(x) = 0
def solve(f , a: float, b: float, x):
    r = 0       # the median point for the given interval
    k = int(0)  # iteration counter
    while(k < MAX_ITERATIONS):
        r = (a + b) / 2     # get the median point of [a,b]
        relative_tolerance = (r - a) / r       # check relative
tolerance
        if relative_tolerance < MAX_RELATIVE_TOLERANCE:
            break

        fa = f.evalf(subs={x: a})
        fb = f.evalf(subs={x: b})
        fr = f.evalf(subs={x: r})
        if fr == 0:     # check for possible solution
            break

        if fa * fr < 0: # check if f(x)=0 is in [a,r]...
            b = r
        else:           # ... or in [r,b]
            a = r

        k += 1

    return r

# Basic method for checking if the given interval [a,b] contains a
solution to f(x)
#  f: mathematical expression for the given function, where the
variable is x
```

```
# a: lower bound for x
# b: upper bound for x
# x: symbol for expression variable
def bolzanoTheorem(f, a: float, b: float, x):
    fa = f.evalf(subs={x: a})
    fb = f.evalf(subs={x: b})

    if fa*fb < 0:
        return true
    else:
        return false

def userInput():
    while True:
        x = symbols("x")
        expression = sympify(str(input("Introduza a função (f(x)): 
")))
        lower_bound = float(input("Introduza o limite inferior de x 
(a): "))
        upper_bound = float(input("Introduza o limite superior de x 
(b): "))

        # Input checks
        if lower_bound > upper_bound:
            print("O limite inferior tem de ser menor que o 
superior!")
            continue

        if not bolzanoTheorem(expression, lower_bound, upper_bound, 
x):
            user_continue = input(f"O intervalo pode não conter uma 
solução para função!\nPretende continuar? (Y/N): ")
            if user_continue == "N" or user_continue == "n":
                continue

        # Main dish method call
        print(solve(expression, lower_bound, upper_bound, x))

        # Loop if user doesn't input "N"
        user_continue = input(f"Pretende continuar com este método? 
(Y/N): ")
        if user_continue == "N" or user_continue == "n":
            break
```

newton.py

```
# TRABALHO 2 - Método de Newton
# João Andréz - 61066
# Carlos Ferreira - 71319
# Diogo Freitas - 90147
# Diogo Carvalho - 90247

from sympy import * # type: ignore
```

```
MAX_ITERATIONS = 10
MAX_RELATIVE_TOLERANCE = 10**(-9)

# Function for finding f(x) = 0 where:
#  f: mathematical expression for the given function, where the
variable is x
# r0: initial aproximation of x where f(x) = 0
# x: symbol for expression variable
# returns an aproximation of x where f(x) = 0
def solve(f , r0: float, x):
    r = r0              # aproximation of solution
    diff_f = f.diff(x)  # f'(x)
    k = int(0)          # iteration counter
    while(k < MAX_ITERATIONS):
        r_old = r       # auxiliary temp of r
        fr_old = f.evalf(subs={x: r_old})                      # aux
f(r_old)
        diff_fr_old = diff_f.evalf(subs={x: r_old})          # aux
f'(r_old)
        r = r_old - (fr_old / diff_fr_old)                # get the
new solution aproximation

        relative_tolerance = abs(r - r_old) / r            # check
relative tolerance
        if relative_tolerance < MAX_RELATIVE_TOLERANCE:
            break

        fr = f.evalf(subs={x: r})
        if fr == 0:     # check for possible solution
            break

        k += 1

    return r

def userInput():
    while True:
        x = symbols("x")
        expression = sympify(str(input("Introduza a função (f(x)):
")))
        initial_aproximation    =    float(input("Introduza    uma
aproximação inicial de x (r0): "))

        # Main dish method call
        print(solve(expression, initial_aproximation, x))

        # Loop if user doesn't input "N"
        user_continue = input(f"Pretende continuar com este método?
(Y/N): ")
        if user_continue == "N" or user_continue == "n":
            break
```

secant.py

```
# TRABALHO 2 - Método da Secante
# João Andréz - 61066
# Carlos Ferreira - 71319
# Diogo Freitas - 90147
# Diogo Carvalho - 90247

from sympy import * # type: ignore

MAX_ITERATIONS = 10
MAX_RELATIVE_TOLERANCE = 10**(-9)

# Function for finding f(x) = 0 where:
#  f: mathematical expression for the given function, where the
variable is x
# a: lower bound for x
# b: upper bound for x
# r_minus1: initial aproximation of x where f(x) = 0
# r0: initial aproximation of x where f(x) = 0
# x: symbol for expression variable
# returns an aproximation of x where f(x) = 0
def solve(f , a: float, b: float, r_minus1: float, r0: float, x):
    r = float()
    k = int(0)          # iteration counter
    while(k < MAX_ITERATIONS):
        fr_minus1 = f.evalf(subs={x: r_minus1})
        fr0 = f.evalf(subs={x: r0})
        r = r0 - fr0 * (r0 - r_minus1) / (fr0 - fr_minus1)   # new
solution aproximation

        relative_tolerance = abs(r - r0) / r      # check relative
tolerance
        if relative_tolerance < MAX_RELATIVE_TOLERANCE:
            break

        fr = f.evalf(subs={x: r})
        if fr == 0:     # check for possible solution
            break

        r_minus1 = r0       # value update
        r0 = r              # value update

        k += 1

    return r

def userInput():
    while True:
        x = symbols("x")
        expression = sympify(str(input("Introduza a função (f(x)):
")))
        lower_bound = float(input("Introduza o limite inferior de x
(a): "))
        upper_bound = float(input("Introduza o limite superior de x
(b): "))
```

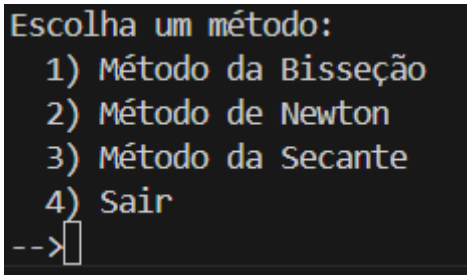
```
        initial_aproximation_minus1 = float(input("Introduza uma
aproximação inicial de x (r-1): "))
        initial_aproximation0 = float(input("Introduza uma
aproximação inicial de x (r0): "))

        # Input checks
        if lower_bound > upper_bound:
            print("O limite inferior tem de ser menor que o
superior!")
            continue

        #IDEIA:
        #se falhar, perguntar ao utilizador se pretende continuar ou
redefinir intervalo
        #if not bolzanoTheorem(expression, lower_bound, upper_bound,
x):
        #       print("O intervalo não contem uma solução para
função!")
        #    continue

        # Main dish method call
        print(solve(expression, lower_bound, upper_bound,
initial_aproximation_minus1, initial_aproximation0, x))

        # Loop if user doesn't input "N"
        user_continue = input(f"Pretende continuar com este método?
(Y/N): ")
        if user_continue == "N" or user_continue == "n":
            break
```

Resultados e Testes

Foram realizados diversos testes para demostrar a correta implementação de cada um dos programas.

Main.py

Ao Executarmos os Programa Main.py, é no apresentado um pequeno Menu, onde podemos escolher o Método que Pretendemos.



bissection.py

```
Introduza a função (f(x)): (x - 1)*(x - 2)
Introduza o limite inferior de x (a): 1.5
Introduza o limite superior de x (b): 2.5
1.9999847412109375
```

```
Introduza a função (f(x)): cos(x) - x
Introduza o limite inferior de x (a): 0
Introduza o limite superior de x (b): 1
0.7390861511230469
```

```
Introduza a função (f(x)): x**3 - x - 2
Introduza o limite inferior de x (a): 1
Introduza o limite superior de x (b): 2
1.5213851928710938
```

```
Introduza a função (f(x)): exp(x) - 3
Introduza o limite inferior de x (a): 0
Introduza o limite superior de x (b): 2
1.0986099243164062
```

```
Introduza a função (f(x)): 12*x**6 - 300*x**5 + 9*x**4 - 10*x**3 + 6*x**2 + 21*x + 13
Introduza o limite inferior de x (a): 0
Introduza o limite superior de x (b): 1
0.6224632263183594
```

newton.py

```
Introduza a função (f(x)): (x - 1)*(x - 2)
Introduza uma aproximação inicial de x (r0): 2.2
2.00000000000000
```

```
Introduza a função (f(x)): x**3 - x - 2
Introduza uma aproximação inicial de x (r0): 1.5
1.52137970680457
```

```
Introduza a função (f(x)): cos(x) - x
Introduza uma aproximação inicial de x (r0): 0.5
0.739085133215161
```

```
Introduza a função (f(x)): exp(x) - 3
Introduza uma aproximação inicial de x (r0): 1
1.09861228866811
```

```
Introduza a função (f(x)): 12*x**6 - 300*x**5 + 9*x**4 - 10*x**3 + 6*x**2 + 21*x + 13
Introduza uma aproximação inicial de x (r0): 0.7
0.622460804565664
```

secant.py

```
Introduza a função (f(x)): (x - 1)*(x - 2)
Introduza o limite inferior de x (a): 1.5
Introduza o limite superior de x (b): 2.5
Introduza uma aproximação inicial de x (r-1): 1.8
Introduza uma aproximação inicial de x (r0): 2.2
2.00000000000000
```

```
Introduza a função (f(x)): x**3 - x - 2
Introduza o limite inferior de x (a): 1
Introduza o limite superior de x (b): 2
Introduza uma aproximação inicial de x (r-1): 1.5
Introduza uma aproximação inicial de x (r0): 1.7
1.52137970680457
```

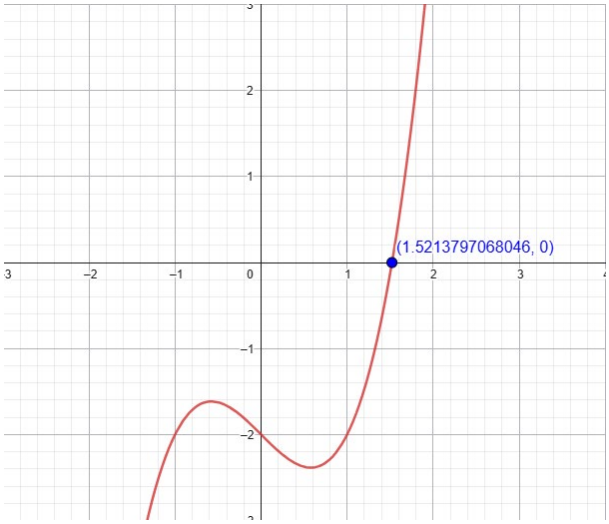
```
Introduza a função (f(x)): cos(x) - x
Introduza o limite inferior de x (a): 0
Introduza o limite superior de x (b): 1
Introduza uma aproximação inicial de x (r-1): 0
Introduza uma aproximação inicial de x (r0): 1
0.739085133215161
```

```
Introduza a função (f(x)): exp(x) - 3
Introduza o limite inferior de x (a): 0
Introduza o limite superior de x (b): 2
Introduza uma aproximação inicial de x (r-1): 0.5
Introduza uma aproximação inicial de x (r0): 1.5
1.09861228866811
```

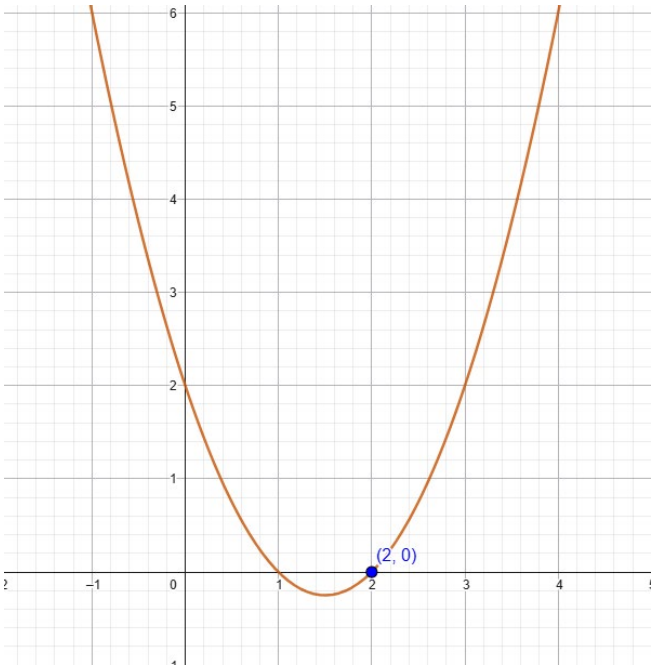
```
Introduza a função (f(x)): 12*x**6 - 300*x**5 + 9*x**4 - 10*x**3 + 6*x**2 + 21*x + 13
Introduza o limite inferior de x (a): 0
Introduza o limite superior de x (b): 1
Introduza uma aproximação inicial de x (r-1): 0.5
Introduza uma aproximação inicial de x (r0): 0.8
0.622460804565664
```

Verificação de Resultados no Geogebra

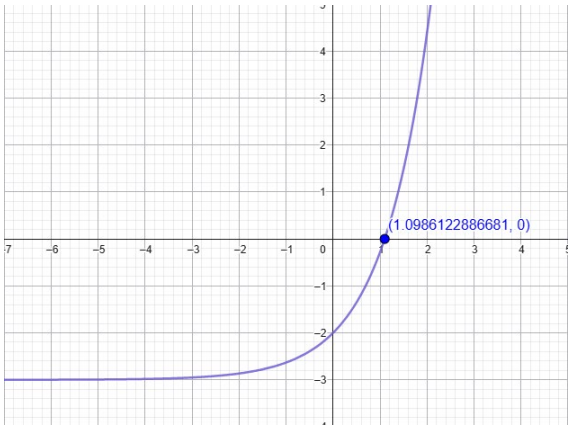
f(x) = x^3 - x – 2



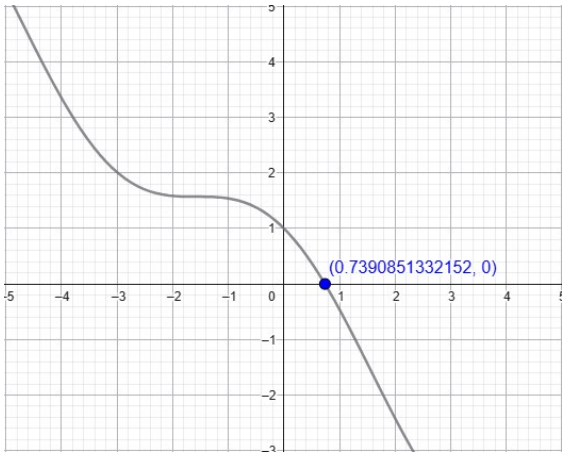
f(x) = (x - 1)*(x - 2)



f(x) = exp(x) – 3



f(x) = cos(x) – x



f(x) = 12*x^6 - 300*x^5 + 9*x^4 - 10*x^3 + 6*x^2 + 21*x + 13