Prova 1 Cálculo Numérico

6

a

Qual o número de iteracões necessárias para encontrar a raíz da função f(x) = (x)^2 − 14x +3 no intervalo [0; 1] com erro de 2*10^{-}3?
É possível pois estamos procurando à raiz da equação em um intervalo coerente.

b

- Fácil de implementar
- Não utiliza derivadas
- Convergência Garantida
- Não é tão sensível a condições iniciais

c

```
# Cálculo numérico para engenharia elétrica com PYTHON
# Capítulo 4: Raízes
# Método intervalar: Bissecção

from math import log, e,sin  # importa função logarítimo neperiano
e a cte de euler

def f(x):  # Resistência (Ohms)
   return x**2 - 14*x + 3

def bisseccao(x1,  # x1 início do intervalo
              x2,  # x2 fim do intervalo
              TOL,  # erro tolerado
              iter=10):  # número máximo de iterações

   hp = (x1 + x2)/2  # Ponto médio entre os valores x1 e x2
   if f(x1) * f(x2) > 0:
       print("Nenhuma raíz encontrada.")  # nenhuma raíz.
   else:
       c = 0  # variável contador
       ERRO = abs(f(x2) - f(x1))  # diferença entre os valores de y
       while ERRO > TOL or c < iter:  # loop iterativo com
critérios de parada
           hp = (x1 + x2) / 2.0
           if f(hp) == 0:
```

```
                return [hp, c]
            elif f(x1) * f(hp) < 0:
                x2 = hp
                c += 1  # contagem
            else:
                x1 = hp
            ERRO = abs(f(x2) - f(x1))
        return {"hp": hp, "iteração": c}  # raíz da função; número
de iterações

resp = bisseccao(x1=0, x2=1, TOL=0.0002, iter=10)
print(f'raíz aprox {resp["hp"]:.4f}')
print(f'O número de iterações foi {resp["iteração"]}')
```

```
/home/salgado/Desktop/CN/venv/bin/python /home/salgado/Desktop/CN/Prova1/6.py

raíz aprox 0.2177
O número de iterações foi 10

Process finished with exit code 0
```

d

Com base nas condições iniciais e nos critérios de parada escolhidos, o método da bissecção é uma escolha apropriada para encontrar a raiz da função fornecida. A raiz aproximada encontrada (0.2177) está dentro da faixa esperada e foi obtida após 10 iterações, atendendo ao critério de convergência definido.

7

a

Supondo que por malhas de um determinado circuito chegamos que as equações são dadas por:
{ 10i1 + 10(i1+i2) = 50
{ 10i2 + 10(i1+i2) = 50

b

- Convergência Rápida
- Precisão Ajustável
- Ampla Aplicabilidade

Para sistemas não lineares.

c

```python
# Cálculo numérico para engenharia elétrica com PYTHON
# Capítulo 6: Sistemas Não Lineares
# Método iterativo: Newton-Raphson
# Circuito magnético


from math import pi, exp
from numpy import array, divide, linalg

x = array([5,  # aproximações iniciais
           5])
iter = 0
maxit = 50
es = 0.001
while True:
   f1 = 10 * x[0] + 10 * x[0] * x[1] - 50
   f2 = 10 * x[1] + 10 * x[0] * x[1] - 50
   F = array([f1, f2])  # Matriz de funções

   J = array([[10 + 10 * x[1], 10 * x[0]],
              [10 * x[1], 10 + 10 * x[0]]])  # Matriz Jacobiana

   dx = linalg.lstsq(J, F, rcond=None)[0]
   x = x - dx
   iter += 1
   ea = max(abs(divide(dx, x)))

   # print(f'inter{iter}')
   # print(f'x{x}')
```

```
    # print(f'err{ea/100}')

    if iter >= maxit or ea <= es:
        break
print(f'Com {iter} iterações: \n x = \n {x}')
```

```
/home/salgado/Desktop/CN/venv/bin/python /home/salgado/Desktop/CN/Prova1/7.2.py

Com 5 iterações:
 x =
 [1.79128785 1.79128785]

Process finished with exit code 0
```

d

Com base nas condições iniciais e nos critérios de parada escolhidos, Método de Newton-Raphson nos entrega convergência rápida e controle de precisão, mas sua eficácia depende da escolha adequada de condições iniciais e da computação das derivadas. Afim de se adequar ao Ea escolhido