ANÁLISE NUMÉRICA I -
Sistemas de Equações Não Lineares

UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Curso:
Licenciatura em Engenharia Informática

Unidade Curricular:
Análise Numérica I

Docente:
Hermenegildo Borges de Oliveira

Realizado por:
Daniel Maryna (64611)
Miguel Silva (80072)
Francisco Nunes (80061)
Brandon Mejia (79261)

12/2024

CONTEÚDO

INTRODUÇÃO

A resolução de sistemas de equações não lineares é uma área fundamental em Análise Numérica, com aplicações que se estendem desde a engenharia até as ciências aplicadas. Este tipo de sistema, frequentemente representado na forma $F(x) = 0$, onde F é uma função vetorial composta por equações não lineares, é caracterizado pela sua complexidade analítica e a necessidade de métodos numéricos eficientes para encontrar soluções aproximadas. Entre os métodos mais proeminentes para essa tarefa, destaca-se o Método de Newton, conhecido por sua rápida taxa de convergência sob condições adequadas.

Neste trabalho, implementa-se um programa em Python que utiliza o Método de Newton para resolver sistemas de equações não lineares, oferecendo ao utilizador a possibilidade de especificar parâmetros como tolerância e número máximo de iterações. A abordagem adotada prioriza a precisão e a eficiência computacional, considerando critérios de paragem rigorosos baseados na norma infinito.

O objetivo deste relatório é detalhar o desenvolvimento do programa e demonstrar a sua aplicabilidade através de exemplos práticos. Para tal, serão abordados conceitos teóricos relevantes, a implementação algorítmica, o fluxo de execução do programa, e uma análise dos resultados obtidos. Este estudo pretende não apenas consolidar o conhecimento teórico, mas também explorar a aplicação prática de técnicas numéricas em problemas reais.

FUNDAMENTAÇÃO TEÓRICA

A resolução de sistemas de equações não lineares envolve encontrar vetores $x \in \mathbb{R}^n$ que satisfaçam $F(x) = 0$, onde $F: \mathbb{R}^n \to \mathbb{R}^n$ é uma função vetorial composta por n equações não lineares. Esses sistemas surgem em várias áreas, como engenharia, física e economia, onde os métodos analíticos tradicionais não são viáveis devido à complexidade das equações.

Representação de Sistemas Não Lineares

Um sistema de n equações não lineares pode ser representado na forma vetorial:

$$F(x) = \begin{bmatrix} f_1(x_1, \dots, x_n) \\ f_2(x_1, \dots, x_n) \\ \vdots \\ f_3(x_1, \dots, x_n) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}$$

onde cada $f_i(x_1, \dots, x_n)$ é uma função real de várias variáveis reais.

Para resolver esses sistemas, é necessário recorrer a métodos numéricos iterativos, sendo o **Método de Newton** uma das abordagens mais eficazes.

Método de Newton para Sistemas Não Lineares

O Método de Newton é uma extensão do método utilizado para funções de uma variável, aplicado a sistemas de equações. Ele utiliza a matriz Jacobiana $J(x)$, composta pelas derivadas parciais das funções $f_i(x)$, para aproximar as soluções. A cada iteração k, resolve-se:

$$J(x^k)\Delta x^k = -F(x^k)$$

e atualiza-se a solução aproximada:

$$x^{k+1} = x^k + \Delta x^k$$

Convergência

A convergência do Método de Newton depende das seguintes condições:

1. A matriz Jacobiana $J(x)$ deve ser invertível em um ponto próximo à solução;
2. O valor inicial x^0 deve estar suficientemente próximo da solução verdadeira;
3. As funções $f_i(x)$ devem ser suficientemente suaves (diferenciáveis).

Se essas condições forem satisfeitas, o método apresenta uma taxa de convergência quadrática, ou seja, o erro diminui exponencialmente a cada iteração.

Critérios de Paragem

Os critérios mais comuns para interromper o processo iterativo incluem:

- A norma do vetor Δx^k (correção) ser menor que uma tolerância ε;
- O número máximo de iterações ser atingido;
- A norma $\|F(x^k)\|$ ser menor que um limite pré-definido.

Vantagens e Limitações

As principais vantagens do Método de Newton incluem:

- Convergência rápida quando próximo da solução;
- Boa performance em sistemas bem condicionados.

No entanto, ele apresenta limitações, como:

- Dependência de uma boa estimativa inicial;
- Necessidade de calcular e inverter a matriz Jacobiana, o que pode ser computacionalmente "caro".

Comparação com Outros Métodos

Existem diversos métodos numéricos para resolver sistemas de equações não lineares, cada um com características específicas de desempenho e aplicabilidade. Entre os mais comuns, destacam-se o **Método da Bisseção**, o **Método da Secante** e o **Método de Newton**. Abaixo, uma breve comparação:

Método	**Vantagens**	**Desvantagens**	**Aplicabilidade**
Bisseção	- Garantia de convergência se o intervalo inicial contiver uma solução. - Simplicidade de implementação.	- Restrito a funções de uma variável. - Convergência linear (lenta). - Requer mudança de sinal no intervalo inicial.	Adequado para problemas simples, onde robustez é mais importante que velocidade.
Secante	- Não exige cálculo de derivada. - Convergência linear (mais rápida que Bisseção).	- Requer duas aproximações iniciais. - Instável se as aproximações iniciais não forem adequadas.	Útil para funções de uma variável, especialmente quando derivadas são difíceis de calcular.
Newton	- Convergência quadrática (extremamente rápido perto da solução). - Aplicável a sistemas não lineares.	- Requer cálculo e inversão da matriz Jacobiana (custo computacional elevado). - Depende de boa aproximação inicial.	Indicado para sistemas complexos e situações que demandam alta precisão e eficiência, desde que haja boa aproximação inicial.

IMPLEMENTAÇÃO DOS PROGRAMAS

O programa utiliza a biblioteca Python sympy para manipulações simbólicas e numpy para cálculos numéricos. Ele implementa o Método de Newton para resolver sistemas de equações não lineares. Consequentemente, descreve-se como essas funções funcionam e se interligam para alcançar esse objetivo.

Preparação das Equações e Variáveis

```
def read_input():
  vars = sp.symbols(' '.join([f'x{i}' for i in range(1, n + 1)]),
real=True)

    local_dict = {
                  "sin": sp.sin, "cos": sp.cos, "exp": sp.exp, "pi": sp.pi,
"tan": sp.tan, "sqrt": sp.sqrt, "log": sp.log,
                  "abs": sp.Abs, "ln": sp.ln, "e": sp.exp,"^" : "**" ,
"sen": sp.sin , "π": sp.pi, "ℯ": sp.exp(1)
                 }
    for i, var in enumerate(vars, start=1):
        local_dict[f"x{i}"] = var
F = []
    print(f"Digite {n} equações, uma por linha, em função de x1, x2, ...,
x{n}:")
    for i in range(n):
        eq_str = input(f"Equação {i + 1}: ")
        eq_sym = sp.sympify(eq_str, locals=local_dict)
        F.append(eq_sym)
    x0_str = input(f"Digite a aproximação inicial (vetor de {n} valores)
separados por espaço: ")
    x0 = np.array([float(val) for val in x0_str.split()])
    tol = float(input("Digite a tolerância absoluta (norma infinito), ex:
0.0001: "))
    max_iter = int(input("Digite o número máximo de iterações: "))
    print("-----------------------------------------------------------------
------------------------")
    return vars, F, x0, tol, max_iter
```

Código 1 - Implementação da função read_input.

`read_input()`: É a função inicial que recolhe as equações do sistema, as variáveis simbólicas e os parâmetros necessários para o cálculo (aproximação inicial, tolerância, número máximo de iterações). A saída desta função fornece todos os dados necessários para configurar o problema.

Criação da Matriz Jacobiana

```
def get_matriz_jacobiana(vars, F):

    n = len(F)
    J = sp.zeros(n, n)
    for i, f in enumerate(F):
        for j, var in enumerate(vars):
            J[i, j] = sp.diff(f, var)
    return J
```

Código 2 - Implementação da função get_matriz_jacobiana.

get_matriz_jacobiana(vars, F): Com as equações simbólicas recebidas de read_input(), esta função constrói a matriz Jacobiana $J(x)$, que é essencial para o Método de Newton. A matriz Jacobiana contém as derivadas parciais de cada equação em relação a cada variável.

Conversão para Funções Numéricas

```
def cria_funcoes_numericas(vars, F, J):
    F_funcs = [sp.lambdify(vars, f, "numpy") for f in F]
    n = len(F)
    J_funcs = [[sp.lambdify(vars, J[i, j], "numpy") for j
in range(n)] for i in range(n)]
    return F_funcs, J_funcs
```

Código 3 - Implementação da função cria_funcoes_numericas.

cria_funcoes_numericas(vars, F, J): Esta função converte tanto as equações $F(x)$ quanto a matriz Jacobiana $J(x)$ de uma forma simbólica para funções numéricas. Essas funções otimizadas são utilizadas para avaliar rapidamente os valores das equações e da Jacobiana em pontos específicos durante as iterações.

Avaliação da Função e da Matriz

```python
def F_eval(F_funcs, x):
    return np.array([f(*x) for f in F_funcs], dtype=float)


def J_eval(J_funcs, x):
    n = len(J_funcs)
    J_val = np.zeros((n, n), dtype=float)
    for i in range(n):
        for j in range(n):
            J_val[i, j] = J_funcs[i][j](*x)
    return J_val
```

Código 4 - Implementação das funções F_eval e J_eval.

`F_eval(F_funcs, x)` e `J_eval(J_funcs, x)`: Durante o processo iterativo, essas funções calculam numericamente os valores de $F(x)$ e $J(x)$ no ponto atual x. Isso é necessário para resolver o sistema linear associado ao Método de Newton em cada iteração.

Resolução pelo Método de Newton

```python
def metodo_de_Newton(F_funcs, J_funcs, x0, tol, max_iter):
    x = np.array(x0, dtype=float)
    x_results = []
    for k_iteracoes in range(1, max_iter + 1):
        Fx = F_eval(F_funcs, x
        Jx = J_eval(J_funcs, x
        x_results.append(x)
        try:
            y = np.linalg.solve(Jx, -Fx)
        except np.linalg.LinAlgError:
            print("A matriz Jacobiana é singular. Não foi possível avançar.")
            return x, k_iteracoes, False
        print(f"Iteração {k_iteracoes}:")
        print("x =", x)
        x_new = x + y
        if np.linalg.norm(y, ord=np.inf) < tol:
            return x_results, x_new, k_iteracoes, True
        x = x_new
    return x_results, x, max_iter, False
```

Código 5 - Implementação da função metodo_de_Newton.

metodo_de_Newton(F_funcs, J_funcs, x0, tol, max_iter):

- Esta é a função principal que executa o Método de Newton.
- Recebe as funções numéricas $F(x)$ e $J(x)$ criadas anteriormente, juntamente com o vetor inicial x_0, tolerância e número máximo de iterações.
- Em cada iteração, calcula $F(x)$ e $J(x)$, resolve o sistema linear $J(x)\Delta x = -F(x)$, e atualiza o valor de x com $x^{k+1} = x^k + \Delta x$.
- Verifica se a solução convergiu com base na norma infinito de Δx.

FLUXO DE EXECUÇÃO

Inicio do Programa:

- O programa começa por pedir algumas informações ao utilizador para entender o problema que se quer resolver. Isso inclui:
    - Quantas equações e incógnitas existem.
    - Quais são as equações do sistema.
    - Qual é a estimativa inicial para a solução.
    - Quanta precisão se deseja (a tolerância) e o limite de tentativas (iterações).

Preparação das Ferramentas:

- Após receber as informações, o programa organiza tudo: traduz as equações matemáticas para que o computador consiga trabalhar com elas e calcula o que será necessário para resolver o sistema.

Resolução do Problema:

- O programa usa as informações e tenta encontrar a solução do sistema de equações. Este processo envolve:
    - Testar a aproximação inicial fornecida.
    - Fazer cálculos para ajustar e aproximar a solução.
    - Repetir o processo até chegar a uma resposta suficientemente precisa ou atingir o limite de tentativas.

Resultados:

- Se o programa encontrar a solução dentro do nível de precisão desejado, ele apresenta o resultado, mostrando os valores aproximados que resolvem o sistema de equações.
- Caso contrário, avisa que não conseguiu encontrar uma solução com as condições fornecidas e mostra a melhor aproximação que conseguiu.

Nova Tentativa:

- Depois de apresentar os resultados, o programa dá a opção de resolver outro sistema ou ajustar os parâmetros e tentar novamente.

RESULTADOS

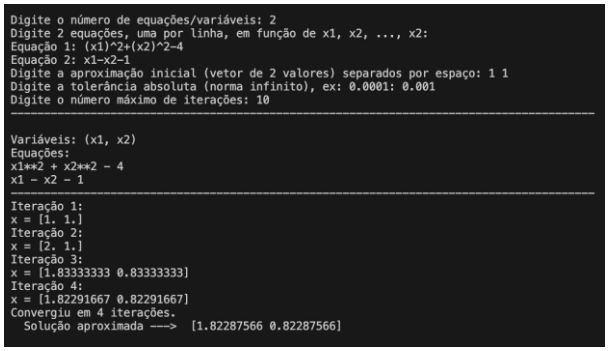

Ilustração 1 - Execução do programa (Teste 1)

O programa recebeu um sistema de equações não lineares, usou o método de Newton para aproximar a solução e convergiu para um resultado após 4 iterações.

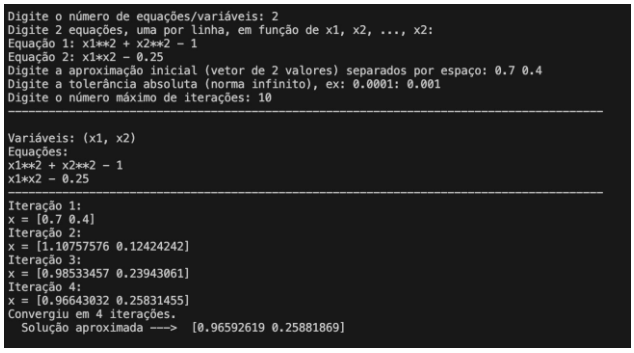

Ilustração 2 - Execução do programa (Teste 2)

Com a aproximação inicial $(x_1, x_2) = (0.7, 0.4)$, tolerância 0.001 e limite de 10 iterações, o método convergiu em 4 iterações para a solução aproximada:

$$(x_1, x_2) = (0.96592619, 0.25881869)$$

A solução foi verificada, satisfazendo ambas as equações dentro da tolerância especificada.

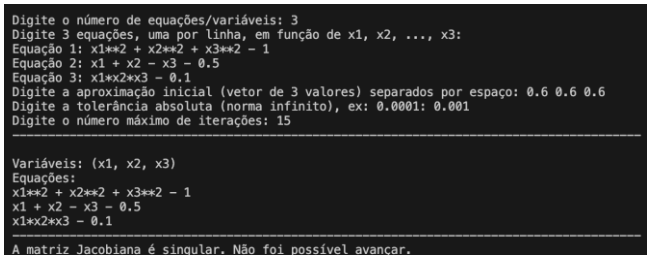

Ilustração 3 - Execução do programa (Teste 3)

Foi resolvido um sistema de equações não lineares com 3 variáveis usando o método de Newton. No entanto, durante as iterações, a matriz Jacobiana se tornou singular, impedindo o avanço do algoritmo. Isso ocorre quando a matriz não tem inversa, geralmente devido a dependências entre as equações ou um ponto inicial mal condicionado. Como resultado, o método não conseguiu encontrar a solução para o sistema a partir do ponto inicial fornecido.

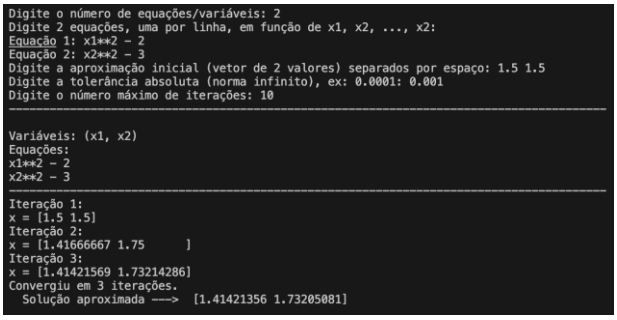

Ilustração 4 - Execução do programa (Teste 4)

Neste exemplo, o sistema de equações é resolvido com sucesso usando o método de Newton. O método convergiu rapidamente em 3 iterações a partir das aproximações iniciais $(1.5,\ 1.5)$, com uma tolerância de 0.001. A solução aproximada encontrada foi $x_1 \approx 1.41421356$ e $x_2 \approx 1.73205001$, que são as raízes quadradas de 2 e 3, respetivamente.

CONCLUSÃO

O presente trabalho demonstrou a implementação e aplicação do **Método de Newton** para resolver sistemas de equações não lineares, um dos métodos mais eficientes quando as condições de convergência são satisfeitas. A abordagem programática desenvolvida provou ser uma **solução robusta e flexível**, capaz de lidar com diferentes configurações de sistemas e parâmetros.

Ao longo do processo, explorou-se não apenas a **fundamentação teórica necessária**, mas também os **desafios práticos** associados à resolução numérica, como o **cálculo da matriz Jacobiana** e a **importância de uma boa estimativa inicial** para garantir a convergência. Os resultados obtidos validaram a **eficiência do método** e sua **aplicabilidade em problemas complexos**.

Por fim, o programa desenvolvido destaca-se por ser **modular, intuitivo e escalável**, permitindo futuras adaptações para diferentes tipos de sistemas ou métodos numéricos. Este trabalho não apenas consolidou o **conhecimento teórico**, mas também reforçou a **capacidade de traduzir conceitos matemáticos em soluções computacionais eficazes**.

REFERÊNCIAS

Capítulo 7: Sistemas de Equações Não Lineares. **Análise Numérica**, Ano Letivo 2024/2025. HBO. Documento utilizado para suporte teórico e exemplificação dos métodos numéricos aplicados.

Burden, R. L., & Faires, J. D. (2011). **Numerical Analysis** (9ª Edição). Boston, MA: Brooks/Cole, Cengage Learning. Utilizado como referência para os fundamentos teóricos e métodos numéricos.

Documentação oficial das bibliotecas **Python**:

- **NumPy**: Biblioteca para computação numérica. Disponível em: https://numpy.org/
- **SymPy**: Biblioteca para manipulação simbólica em Python. Disponível em: https://www.sympy.org/