COC473 - Trabalho 1 - 2021.1

Aluno: Henrique Chaves Magalhães de Menezes

DRE: 119025571

Algoritmos:

Gerais:

In [1]:

```python
import numpy as np
from sympy import Symbol

def verificar_simetria(A):
    if (A == A.T).all():
        return True
    else:
        return False

def verificar_positiva_definida(A):
    if np.all(np.linalg.eigvals(A) > 0):
        return True
    else:
        return False

def tem_diagonal_dominante(A):
    D = np.diag(np.abs(A))
    S = np.sum(np.abs(A), axis=1) - D
    if np.all(D > S):
        return True
    else:
        return False

def verificar_matriz_quadrada(A):
    if (len(A.shape) == 2) and (A.shape[0] == A.shape[1]):
        return True
    else:
        return False

def verifica_tamanho_vetor(A, b):
    if b.shape[0] == A.shape[0]:
        return True
    else:
        return False
```

Decomposição LU

In [2]:

```
def decomposicao_lu(A):
    n = A.shape[0]

    L = np.eye(n, dtype=np.double)

    for i in range(n):
        factor = A[i+1:, i]/A[i, i]
        L[i+1:, i] = factor
        A[i+1:] = A[i+1:] - factor[:, np.newaxis] * A[i]

    return L, A

def resolve_lu(L, U, b):
    n = b.shape[0]
    y = np.zeros((n,1))
    x = np.zeros((n,1))

    y[0][0] = b[0][0]/L[0, 0]

    for i in range(2, n+1):
        y[i-1][0] = (b[i-1][0] - sum(L[i-1, j-1]*y[j-1][0] for j in range(1, i)))/L


    x[n-1][0] = y[n-1][0]/U[n-1, n-1]

    for i in reversed(range(1, n)):
        x[i-1][0] = (y[i-1][0] - sum(U[i-1, j-1]*x[j-1][0] for j in range(i+1, n+1)

    return x
```

Exemplo:

In [3]:

```
A = np.array([[1, 2, 2],
              [4, 4, 2],
              [4, 6, 4]])

print("Matriz A:")
print(A)
```

```
Matriz A:
[[1 2 2]
 [4 4 2]
 [4 6 4]]
```

In [4]:

```
b = np.array([[3], [6], [10]])

print("Vetor B:")
print(b)
```

```
Vetor B:
[[ 3]
 [ 6]
 [10]]
```

In [5]:

```
L, U = decomposicao_lu(A)
```

In [6]:

```
print("Matriz L:")
print(L)
```

```
Matriz L:
[[1.  0.  0. ]
 [4.  1.  0. ]
 [4.  0.5 1. ]]
```

In [7]:

```
print("Matriz U:")
print(U)
```

```
Matriz U:
[[ 1  2  2]
 [ 0 -4 -6]
 [ 0  0 -1]]
```

In [8]:

```
x = resolve_lu(L, U, b)
```

In [9]:

```
print("Vetor X:")
print(x)
```

```
Vetor X:
[[-1.]
 [ 3.]
 [-1.]]
```

Decomposição Cholesky

In [10]:

```
def decomposicao_cholesky(A):
    n = A.shape[0]

    L = np.zeros((n, n))

    for i in range(n):
        for k in range(i+1):
            tmp_sum = sum(L[i][j] * L[k][j] for j in range(k))

            if (i == k):
                L[i][i] = np.sqrt(A[i][i] - tmp_sum)
            else:
                L[i][k] = (1.0 / L[k][k] * (A[i][k] - tmp_sum))
    return L

def resolve_cholesky(L, b):
    n = b.shape[0]
    y = np.zeros((n,1))
    x = np.zeros((n,1))

    y[0][0] = b[0][0]/L[0, 0]

    for i in range(2, n+1):
        y[i-1][0] = (b[i-1][0] - sum(L[i-1, j-1]*y[j-1][0] for j in range(1, i)))/L


    x[n-1][0] = y[n-1][0]/L.T[n-1, n-1]

    for i in reversed(range(1, n)):
        x[i-1][0] = (y[i-1][0] - sum(L.T[i-1, j-1]*x[j-1][0] for j in range(i+1, n+

    return x
```

Exemplo:

In [11]:

```
A = np.array([[1, 0.2, 0.4],
              [0.2, 1, 0.5],
              [0.4, 0.5, 1]])

print("Matriz A:")
print(A)
```

```
Matriz A:
[[1.  0.2 0.4]
 [0.2 1.  0.5]
 [0.4 0.5 1. ]]
```

In [12]:

```
b = np.array([[0.6], [-0.3], [-0.6]])

print("Vetor B:")
print(b)
```

```
Vetor B:
[[ 0.6]
 [-0.3]
 [-0.6]]
```

In [13]:

```
L = decomposicao_cholesky(A)

print("Matriz L:")
print(L)
```

```
Matriz L:
[[1.         0.         0.        ]
 [0.2        0.9797959  0.        ]
 [0.4        0.4286607  0.81009259]]
```

In [14]:

```
x = resolve_cholesky(L, b)

print("Vetor X:")
print(x)
```

```
Vetor X:
[[ 1.]
 [ 0.]
 [-1.]]
```

Procedimento iterativo Jacobi

In [15]:

```
def resolve_jacobi(A, b, tol=10**-10):
    if not tem_diagonal_dominante(A):
        raise Exception("Matriz A não tem diagonal dominante.")
    n = A.shape[0]

    R = np.inf
    x_new = np.ones(b.shape)
    count = 0
    historico_R = []

    while (R > tol):
        if count > 1000:
            raise Exception("Número de iterações ultrapassou 1000.")
        x_old = x_new
        x_new = np.zeros(b.shape)

        for i in range(1, n+1):
            x_new[i-1][0] = (b[i-1][0] - sum([A[i-1][j-1]*x_old[j-1][0] for j in ran
            R = np.linalg.norm(x_new - x_old)/np.linalg.norm(x_new)

        historico_R.append(R)
        count += 1

    return x_new, historico_R, count
```

Exemplo:

In [16]:

```
A = np.array([
    [3, -1, -1],
    [-1, 3, -1],
    [-1, -1, 3]
])

print("Matriz A:")
print(A)
```

```
Matriz A:
[[ 3 -1 -1]
 [-1  3 -1]
 [-1 -1  3]]
```

In [17]:

```
b = np.array([[1], [2], [1]])

print("Vetor B:")
print(b)
```

```
Vetor B:
[[1]
 [2]
 [1]]
```

In [18]:

```
x, erros, iteracoes = resolve_jacobi(A, b, tol=10**-10)

print("Vetor X:")
print(x)
```

```
Vetor X:
[[1.25]
 [1.5 ]
 [1.25]]
```

In [19]:

```
print(f"Número de iterações: {iteracoes}")
```

```
Número de iterações: 52
```

In [20]:

```
print(f"Histórico de erros: {np.round(erros, 11)}")
```

```
Histórico de erros: [1.71498585e-01 7.62492852e-02 4.22200331e-02 2.62
662274e-02
 1.70161288e-02 1.11856887e-02 7.39668116e-03 4.90615139e-03
 3.25999271e-03 2.16861023e-03 1.44365678e-03 9.61515430e-04
 6.40601130e-04 4.26885800e-04 2.84509880e-04 1.89637420e-04
 1.26409030e-04 8.42656100e-05 5.61739300e-05 3.74478900e-05
 2.49646400e-05 1.66428200e-05 1.10950900e-05 7.39667000e-06
 4.93109000e-06 3.28738000e-06 2.19158000e-06 1.46105000e-06
 9.74030000e-07 6.49360000e-07 4.32900000e-07 2.88600000e-07
 1.92400000e-07 1.28270000e-07 8.55100000e-08 5.70100000e-08
 3.80100000e-08 2.53400000e-08 1.68900000e-08 1.12600000e-08
 7.51000000e-09 5.00000000e-09 3.34000000e-09 2.22000000e-09
 1.48000000e-09 9.90000000e-10 6.60000000e-10 4.40000000e-10
 2.90000000e-10 2.00000000e-10 1.30000000e-10 9.00000000e-11]
```

Procedimento iterativo Gauss-Seidel

In [21]:

```
def resolve_gauss_seidel(A, b, tol=10**-10):
    if not tem_diagonal_dominante(A):
        raise Exception("Matriz A não tem diagonal dominante.")
    n = A.shape[0]

    R = np.inf
    x_new = np.ones((n, 1))
    count = 0
    historico_R = []

    while (R > tol):
        if count > 1000:
            raise Exception("Número de iterações ultrapassou 1000.")
        x_old = x_new.copy()
        x_new = np.zeros((n, 1))

        for i in range(1, n+1):
            x_new[i-1][0] = (b[i-1][0]- sum([A[i-1][j-1]*x_new[j-1][0] for j in rang
            R = np.linalg.norm(x_new - x_old)/np.linalg.norm(x_new)
        historico_R.append(R)
        count += 1

    return x_new, historico_R, count
```

Exemplo:

In [22]:

```
A = np.array([
    [3, -1, -1],
    [-1, 3, -1],
    [-1, -1, 3]
])

print("Matriz A:")
print(A)
```

```
Matriz A:
[[ 3 -1 -1]
 [-1  3 -1]
 [-1 -1  3]]
```

In [23]:

```
b = np.array([[1], [2], [1]])

print("Vetor B:")
print(b)
```

```
Vetor B:
[[1]
 [2]
 [1]]
```

In [24]:

```
x, erros, iteracoes = resolve_gauss_seidel(A, b, tol=10**-10)

print("Vetor X:")
print(x)
```

```
Vetor X:
[[1.25]
 [1.5 ]
 [1.25]]
```

In [25]:

```
print(f"Número de iterações: {iteracoes}")
```

```
Número de iterações: 28
```

In [26]:

```
print(f"Histórico de erros: {np.round(erros, 11)}")
```

```
Histórico de erros: [1.75411604e-01 8.65015332e-02 3.45549178e-02 1.56
294729e-02
 7.00073680e-03 3.15806131e-03 1.42616171e-03 6.44606650e-04
 2.91448250e-04 1.31794450e-04 5.96023000e-05 2.69552100e-05
 1.21907000e-05 5.51337000e-06 2.49349000e-06 1.12771000e-06
 5.10020000e-07 2.30660000e-07 1.04320000e-07 4.71800000e-08
 2.13400000e-08 9.65000000e-09 4.36000000e-09 1.97000000e-09
 8.90000000e-10 4.00000000e-10 1.80000000e-10 8.00000000e-11]
```

Método da Potência

In [27]:

```
def power_method_eigen(A, tol=10**-10):
    n = A.shape[0]
    R = np.inf

    x_new = np.ones((n, 1))
    lambda_new = x_new[0][0]

    count = 0
    historico_R = []

    while (R > tol):
        lambda_old = lambda_new
        y = np.matmul(A, x_new)
        lambda_new = y[0][0]
        x_new = y/lambda_new

        R = np.abs(lambda_new-lambda_old)/np.abs(lambda_new)
        count += 1
        historico_R.append(R)
    return lambda_new, x_new, historico_R, count
```

Exemplo:

In [28]:

```
A = np.array([
    [1, 0.2, 0],
    [0.2, 1, 0.5],
    [0, 0.5, 1]
])

print("Matriz A:")
print(A)
```

```
Matriz A:
[[1.  0.2 0. ]
 [0.2 1.  0.5]
 [0.  0.5 1. ]]
```

In [29]:

```
autovalor, autovetor, erros, iteracoes = power_method_eigen(A, tol=10**-10)
```

In [30]:

```
print(f"Maior autovalor: {autovalor}")
print("Autovetor associado:")
print(autovetor)
```

```
Maior autovalor: 1.5385164805263052
Autovetor associado:
[[1.       ]
 [2.6925824]
 [2.5      ]]
```

In [31]:

```
print(f"Número de iterações: {iteracoes}")
```

```
Número de iterações: 52
```

In [32]:

```
print(f"Histórico de erros: {np.round(erros, 11)}")
```

```
Histórico de erros: [1.66666667e-01 6.49350649e-02 4.89252486e-02 3.73
032173e-02
 2.79570165e-02 2.03453453e-02 1.43759053e-02 9.91241077e-03
 6.70967795e-03 4.48217021e-03 2.96693534e-03 1.95182844e-03
 1.27873651e-03 8.35473960e-04 5.44883150e-04 3.54945550e-04
 2.31039050e-04 1.50310960e-04 9.77582900e-05 6.35658500e-05
 4.13270000e-05 2.68661000e-05 1.74642500e-05 1.13521600e-05
 7.37898000e-06 4.79631000e-06 3.11755000e-06 2.02636000e-06
 1.31710000e-06 8.56090000e-07 5.56440000e-07 3.61670000e-07
 2.35080000e-07 1.52800000e-07 9.93100000e-08 6.45500000e-08
 4.19600000e-08 2.72700000e-08 1.77300000e-08 1.15200000e-08
 7.49000000e-09 4.87000000e-09 3.16000000e-09 2.06000000e-09
 1.34000000e-09 8.70000000e-10 5.60000000e-10 3.70000000e-10
 2.40000000e-10 1.60000000e-10 1.00000000e-10 7.00000000e-11]
```

Método de Jacobi

In [33]:

```
def indices_maximo_valor(A):
    n = A.shape[0]

    max_valor = np.NINF
    p, q = (np.nan, np.nan)


    for i in range(1, n+1):
        for j in range(i+1, n+1):
            if np.abs(A[i-1][j-1]) > max_valor:
                max_valor = np.abs(A[i-1][j-1])
                p, q = i, j

    return p, q

def calcula_phi(A, p, q):
    if A[p-1][p-1] == A[q-1][q-1]:
        phi = np.pi/4
    else:
        phi = np.arctan(2*A[p-1][q-1]/(A[p-1][p-1]-A[q-1][q-1]))/2
    return phi

def cria_matriz_p(phi, n, p, q):
    P = np.eye(n, n)

    P[p-1][p-1] = np.cos(phi)
    P[p-1][q-1] = -np.sin(phi)
    P[q-1][p-1] = np.sin(phi)
    P[q-1][q-1] = np.cos(phi)

    return P

def checar_tol(A, tol):
    return np.all([np.abs(A[i-1][j-1]) < tol for i in range(1, A.shape[0]+1) for j :

def jacob_eigen(A, tol=10**-10):

    if not verificar_simetria(A):
        raise Exception("Matriz A não é simétrica.")

    n = A.shape[0]

    X = np.eye(*A.shape)
    count = 0
    while True:
        p, q = indices_maximo_valor(A)
        phi = calcula_phi(A, p, q)
        P = cria_matriz_p(phi, n, p, q)

        A = np.matmul(P.T, A)
        A = np.matmul(A, P)

        X = np.matmul(X, P)
        count += 1

        if checar_tol(A, tol):
            return np.diagonal(A), X, count
```

Exemplo:

In [34]:

```
A = np.array([
    [1, 0.2, 0],
    [0.2, 1, 0.5],
    [0, 0.5, 1]
])

print("Matriz A:")
print(A)
```

```
Matriz A:
[[1.  0.2 0. ]
 [0.2 1.  0.5]
 [0.  0.5 1. ]]
```

In [35]:

```
A_, X, iteracoes = jacob_eigen(A)
```

In [36]:

```
for i in range(len(A_)):
    print(f"Autovalor {i+1}: {A_[i]}")
    print(f"Autovetor {i+1}:")
    print(X[:, i].reshape(-1, 1))
    print("\n")
```

```
Autovalor 1: 0.9999999999999999
Autovetor 1:
[[ 9.28476691e-01]
 [-2.02884663e-14]
 [-3.71390676e-01]]


Autovalor 2: 1.53851648071345
Autovetor 2:
[[0.26261287]
 [0.70710678]
 [0.65653216]]


Autovalor 3: 0.46148351928654946
Autovetor 3:
[[ 0.26261287]
 [-0.70710678]
 [ 0.65653216]]
```

In [37]:

```
print(f"Número de iterações: {iteracoes}")
```

```
Número de iterações: 8
```

Interpolação

In [38]:

```
def calcula_phi(x_arr, i, n, x):

    numerator = 1
    denominator = 1

    for k in range(1, n+1):
        if k != i:
            numerator *= (x-x_arr[k-1][0])
            denominator *= (x_arr[i-1][0] - x_arr[k-1][0])

    return numerator/denominator

def interpolacao_lagrange(x_arr, y_arr):
    x = Symbol('x')

    n = x_arr.shape[0]
    phi_arr = np.zeros(n, dtype="object")
    fn = 0

    for i in range(1, n+1):
        phi_arr[i-1] = calcula_phi(x_arr, i, n, x)
        fn += phi_arr[i-1]*y_arr[i-1][0]

    return (fn, lambda x_value: float(fn.subs(x, x_value)))
```

Exemplo:

In [39]:

```
x_arr = np.array([[-2], [0], [1]])
print("Vetor X:")
print(x_arr)
```

```
Vetor X:
[[-2]
 [ 0]
 [ 1]]
```

In [40]:

```
y_arr = np.array([[-27], [-1], [0]])
print("Vetor Y:")
print(y_arr)
```

```
Vetor Y:
[[-27]
 [ -1]
 [  0]]
```

In [41]:

```
expr, lagrange = interpolacao_lagrange(x_arr, y_arr)
```

In [42]:

```
print("Função encontrada:")
display(expr)
```

Função encontrada:

$$-\frac{9x(x-1)}{2} + \frac{(x-1)(x+2)}{2}$$

In [43]:

```
new_x = -2
new_y = lagrange(new_x)

print(f"f({new_x}) = {new_y}")
```

```
f(-2) = -27.0
```

Regressão

In [44]:

```
def regressao_multilinear(x_arr, y_arr, grau=1):
    x = Symbol('x')

    n = x_arr.shape[0]

    P = np.empty((n, grau+1))

    for i in range(grau+1):
        P[:, i] =  np.power(x_arr.ravel(), i)

    A = np.matmul(P.T, P)
    C = np.matmul(P.T, y_arr)
    b = np.round(np.matmul(np.linalg.inv(A), C), 3)

    fn = 0
    for i in range(b.shape[0]):
        fn += x ** i * b[i][0]

    return (fn, lambda x_value: float(fn.subs(x, x_value)))
```

Exemplo:

In [45]:

```
x_arr = np.array([[1], [2], [3]])
print("Vetor X:")
print(x_arr)
```

```
Vetor X:
[[1]
 [2]
 [3]]
```

In [46]:

```
y_arr = np.array([[2], [3.5], [6.5]])
print("Vetor Y:")
print(y_arr)
```

```
Vetor Y:
[[2. ]
 [3.5]
 [6.5]]
```

In [47]:

```
grau = 1
print(f"Regressão de Grau {grau}")
```

```
Regressão de Grau 1
```

In [48]:

```
expr, regressao = regressao_multilinear(x_arr, y_arr, grau=grau)
```

In [49]:

```
print("Função encontrada:")
display(expr)
```

```
Função encontrada:
```

$2.25x - 0.5$

In [50]:

```
new_x = -2
new_y = regressao(new_x)

print(f"f({new_x}) = {new_y}")
```

```
f(-2) = -5.0
```