

```

In [1]: import numpy as np
import pandas as pd

def QfatoracaoGS(n, A):
    Q = np.copy(A) #inicializa Q como uma copia de A

    normas2 = [] #cria um vetor que armazena as normas das bases ortogonais

    for j in range(n):
        for k in range(j):
            for i in range(n):
                Q[i][j] -= np.dot(A[:,j],Q[:,k])*Q[i][k]/(normas2[k]) #remove
as projecoes do vetor com relacao as outras bases ortogonais

            normas2.append(np.dot(Q[:,j],Q[:,j]))

    #ate aqui, Q tem bases ortoGONAIIS, precisamos torna-las ortoNORMAIS: divid
imos pelo modulo do vetor da base

    for j in range(n):
        for i in range(n):
            Q[i,j] = Q[i,j]/np.sqrt(normas2[j])

    return Q

def transposta(A):
    linhas = len(A)
    colunas = len(A[0])

    if(linhas != colunas):
        print("error: A matriz não é quadrada")

    return np.array([[ A[i][j] for i in range(linhas)] for j in range(colunas)
    ]])

def RfatoracaoGS(Q, A):
    return np.dot(transposta(Q), A)

def getHilbert(n):
    A = np.zeros([n,n])

    for i in range(n):
        for j in range(n):
            A[i][j] = 1/(i+j+1)

    return A

def retroSub(A,b):
    n = len(A)
    m = len(A[0])

    x = np.zeros(n)
    # Checando consistência A

```

```
# Resolvendo retro substituição para A triangular superior
# Iniciando o último elemento
x[n-1] = b[n-1]/A[n-1][n-1]

soma = 0

for i in range(n-2,-1,-1):
    soma = b[i]

    for j in range(i+1, n):
        soma = soma - A[i][j]*x[j]

    x[i] = soma/A[i][i]

return x

def resolve_sistema_1d(R,b):
    b = np.array(b)

    return retroSub(R, b)

def resolve_sistema(Q, R, b):
    b = np.array(b)

    b = np.dot(transposta(Q),b)

    n2 = b.shape[0]

    if(len(b.shape) == 1):
        m = 1
    else:
        m = b.shape[1]

    resultado = np.eye(n2)

    for i in range(m):
        resultado[:,i] = resolve_sistema_1d(R,b[:,i])

    return resultado

def norma_mat1(A):

    A = np.array(A)

    if((A.shape == (int(A.shape[0]),)) | (A.shape[0] == 1)):

        return sum([np.abs(v) for v in A])

    c = []

    for i in range(A.shape[0]):
        c.append(sum([np.abs(v) for v in A[:,i]]))
```

```
return np.max(c)
```

## Inicializando uma matriz de hilbert

```
In [2]: n=3  
H = getHilbert(n)  
H
```

```
Out[2]: array([[1.         , 0.5         , 0.33333333],  
               [0.5         , 0.33333333, 0.25        ],  
               [0.33333333, 0.25         , 0.2         ]])
```

## Fazendo a fatoração QR pela classe construida

```
In [3]: Q = QfatoracaoGS(n,H)  
R = RfatoracaoGS(Q,H)
```

## Testando propriedades de Q

```
In [4]: np.dot(Q,transposta(Q)).round(4)
```

```
Out[4]: array([[ 1., -0.,  0.],  
               [-0.,  1.,  0.],  
               [ 0.,  0.,  1.]])
```

## Testando a resolução de sistema linear e encontrando a inversa

```
In [5]: inv_GS = resolve_sistema(Q, R, np.eye(n))  
inv_GS
```

```
Out[5]: array([[ 9., -36.,  30.],  
               [-36., 192., -180.],  
               [ 30., -180., 180.]])
```

```
In [6]: np.dot(H,inv_GS).round(4)
```

```
Out[6]: array([[ 1., -0.,  0.],  
               [-0.,  1., -0.],  
               [ 0., -0.,  1.]])
```

```
In [7]: print(Q)
        print('\n')
        print(R)
```

```
[[ 0.85714286 -0.50160492  0.11704115]
 [ 0.42857143  0.56848557 -0.70224688]
 [ 0.28571429  0.65208639  0.70224688]]
```

```
[[ 1.16666667e+00  6.42857143e-01  4.50000000e-01]
 [ 6.66133815e-16  1.01714330e-01  1.05337032e-01]
 [-2.46191956e-14 -1.98452366e-14  3.90137157e-03]]
```

```
In [8]: inf = np.linalg.solve(R, np.dot(transposta(Q),np.eye(n)))
        inf
```

```
Out[8]: array([[ 9., -36.,  30.],
               [-36., 192., -180.],
               [ 30., -180., 180.]])
```

```
In [9]: np.dot(H,inf).round(4)
```

```
Out[9]: array([[ 1.,  0.,  0.],
               [ 0.,  1., -0.],
               [ 0.,  0.,  1.]])
```

```
In [10]: invof = np.linalg.solve(H,np.eye(n))
        invof
```

```
Out[10]: array([[ 9., -36.,  30.],
                [-36., 192., -180.],
                [ 30., -180., 180.]])
```

```
In [11]: np.dot(H,invof).round(4)
```

```
Out[11]: array([[ 1.,  0.,  0.],
                [ 0.,  1.,  0.],
                [ 0., -0.,  1.]])
```

## Calculando K1 e conferindo as condições

```
In [12]: K1 = norma_mat1(inv_GS)*norma_mat1(H)
        K1
```

```
Out[12]: 748.0000000030523
```

```
In [13]: for k in range(n):
          for j in range(n):
              if(k!=j):
                  ni = norma_mat1(H[:,k])
                  nj = norma_mat1(H[:,j])

                  cond = ni/nj

                  if(K1 > cond):
                      print(f"\nK1 é maior que a coluna {k} pela coluna {j}, K1 vale
{K1.round(4)} e cond vale {cond.round(4)}\n")
                  else:
                      print("não funciona")
```

K1 é maior que a coluna 0 pela coluna 1, K1 vale 748.0 e cond vale 1.6923

K1 é maior que a coluna 0 pela coluna 2, K1 vale 748.0 e cond vale 2.3404

K1 é maior que a coluna 1 pela coluna 0, K1 vale 748.0 e cond vale 0.5909

K1 é maior que a coluna 1 pela coluna 2, K1 vale 748.0 e cond vale 1.383

K1 é maior que a coluna 2 pela coluna 0, K1 vale 748.0 e cond vale 0.4273

K1 é maior que a coluna 2 pela coluna 1, K1 vale 748.0 e cond vale 0.7231