Funções auxiliares e a classe de Householder

```
# Pacote com funções auxiliares, transposta, normal, produto interno e resolução de
sistema linear
#
#@Author: Lucas hideki Ueda, Inaye Caroline, Gustavo guedes

import pandas as pd
import numpy as np

def norma2(x):
    return np.sqrt(np.sum([x[i]**2 for i in range(len(x))]))

def produto_interno(x,y):

    if(len(x)!=len(y)):
        print("error: Os vetores tem tamanho diferentes.")
    else:
        return [x[i]*y[i] for i in range(len(x))]

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 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 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)

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



# Biblioteca de funções para  o projeto de MS512 - Análise numérica 2s2019

# Aqui terão todas as funções necessárias para o projeto
```

```
import utils as ut
import numpy as np
import pandas as pd

# Definindo uma classe HouseHolder, que realizará todas as etapas da decomposição QR por refletores, desde a fatoração até a resolução de um sistema
class HouseHolder():


    def __init__(self,M):
        self.M = M


    def padding(self,A):

        '''

        Função que cria uma hora de zeros a esquerda e em cima na matriz A, e 1 onde for na região concatenada e i=j

        '''


        size = A.shape[0]

        result = np.zeros([size+1,size+1])

        result[1:A.shape[0]+1, 1:A.shape[1]+1] = A

        new_size = result.shape[0]

        for i in range(new_size-size):
            for j in range(new_size-size):
                if(i==j):
                    result[i,j] = 1

        return result

    def getQ(self,x):

        '''
        Função que retorna a matriz ortogonal Q para um dado vetor x de R^n onde,

        Qx = [-t,0,...,00]t


        '''
```

```
    # Checagem se x não é um vetor nulo

    u = x.copy()

    #print(u)
    #print(x)

    beta = max(u)

    if(beta == 0):
        return np.eye(len(u))
    else:
        tau = ut.norma2(x)
        #print(tau)
        #print(u[0])
        if(u[0] < 0):
            tau = -tau

        u[0] = tau +u[0]

        #print(u[0])
        gamma = u[0]/tau

        u =[i/u[0] for i in u]

        #print(np.linalg.norm(x))
        #print(tau)
        Q = np.eye(len(u)) - (2/np.dot(u,u))*np.dot(ut.transposta([u]), [u])

        return Q


def getQR(self,A):

    R = np.array(A)

    n = len(A)

    Qf = np.eye(n)

    for i in range(n - 1):

        Qa = np.eye(n)
        Qa[i:, i:] = self.getQ(R[i:, i])
        Qf = np.dot(Qf,Qa)
```

```
            R = np.dot(Qa, R)

        return Qf, R

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

        return ut.retroSub(self.R, b)

    def resolve_sistema(self,b):
        b = np.array(b)

        Q, R = self.getQR(self.M)

        self.R = R

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

        n = b.shape[0]

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

        resultado = np.eye(n)

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

        return resultado
```