PROBLEM 2 - Householder

Gisela Martí Guerrero, ICC 2023

Python program that computes the Householder diagonalization method applied to the Hilbert matrix $N = 10$.

In [1]:
```
import numpy as np
from pandas import *
```

We start by defining a function that implements the Householder tridiagonalization algorithm, which is a numerical technique used for transforming a symmetric matrix into a tridiagonal matrix. Tridiagonal matrices have zeros everywhere except on the main diagonal, the first subdiagonal, and the first superdiagonal. The function takes a symmetric matrix A as input and returns a tridiagonal matrix.

The function first initializes arrays v, u, and z with zeros to store values. It then iterates over each column index k from 0 to $n - 3$ (where n is the size of the matrix). Inside the loop, it calculates the Householder vector v and uses it to transform the matrix A to a tridiagonal form. The Householder vector is calculated based on the current column k of the matrix. The loop then updates the matrix A using the Householder transformation to eliminate the values below and above the tridiagonal. Finally, the function returns the tridiagonalized matrix A.

In [2]:
```
def householder(A):
    n = A.shape[0]
    v = np.zeros(n)
    u = np.zeros(n)
    z = np.zeros(n)

    for k in range(0, n-2):
        if np.isclose(A[k+1,k], 0.0):
            a = -np.sqrt(np.sum(A[(k+1):,k]**2))
        else:
            a = -np.sign(A[k+1,k]) * np.sqrt(np.sum(A[(k+1):,k]**2))
        r2 = a**2 - a*A[k+1,k]
        v[k] = 0.0
        v[k+1] = A[k+1,k] - a
        v[(k+2):] = A[(k+2):,k]
        u[k:] = 1.0 / r2 * np.dot(A[k:,(k+1):], v[(k+1):])
        z[k:] = u[k:] - np.dot(u,v) / (2.0*r2) * v[k:]

        for l in range(k+1, n-1):
            A[(l+1):,l] = (A[(l+1):,l] - v[l] * z[(l+1):] - v[(l+1):] * z[l])
            A[l,(l+1):] = A[(l+1):,l]
            A[l,l] = A[l,l] - 2*v[l]*z[l]

        A[-1,-1] = A[-1,-1] - 2*v[-1]*z[-1]
        A[k,(k+2):] = 0.0
        A[(k+2):,k] = 0.0
        A[k+1,k] = A[k+1,k] - v[k+1]*z[k]
        A[k,k+1] = A[k+1,k]

    return A
```

Now we define a new function "QR", which performs a QR decomposition of the given matrix A. QR

decomposition expresses A as the product of an orthogonal matrix Q and an upper triangular matrix R.

The function first initializes the first column of u with the first column of A and normalizes it to obtain the first column of Q. It then iterates over each column index i from 1 to $n - 1$ (where n is the number of columns in A). Inside the loop, it constructs the columns of u by subtracting the projections of the current column of A onto the previously computed columns of Q. This process ensures that the columns of Q are orthogonal. It normalizes each column of u to obtain the corresponding column of Q. It then initializes the matrix R with zeros and computes its elements by taking the dot product of the columns of A and the columns of Q. Finally, the function returns the matrices Q and R.

In [3]:
```
def QR(A):
    n, m = A.shape
    Q = np.empty((n, n))
    u = np.empty((n, n))
    u[:, 0] = A[:, 0]
    Q[:, 0] = u[:, 0] / np.linalg.norm(u[:, 0])

    for i in range(1, n):
        u[:, i] = A[:, i]
        for j in range(i):
            u[:, i] -= (A[:, i] @ Q[:, j]) * Q[:, j]
        Q[:, i] = u[:, i] / np.linalg.norm(u[:, i])
    R = np.zeros((n, m))

    for i in range(n):
        for j in range(i, m):
            R[i, j] = A[:, j] @ Q[:, i]

    return Q, R
```

A new function "eigenvalues" is designed to find the eigenvalues of the matrix A using the QR decomposition method.

The function enters a while loop that continues until either the change in the matrix ("diff") is below a specified tolerance ("tol") or the maximum number of iterations ("maxiter") is reached. Within the loop, it updates the value of "A_old" to be the same as "A_new". It then performs QR decomposition on the matrix "A_old" using the "QR" function. It updates the matrix "A_new" to be the product of the upper triangular matrix R and the orthogonal matrix Q. It calculates the maximum absolute difference between corresponding elements of "A_new" and "A_old" to determine if the iteration should continue. The loop continues until the convergence criteria are met or the maximum number of iterations is reached. After the loop, it extracts the diagonal elements of the matrix new matrix A, which represent the eigenvalues.

In [4]:
```
def eigenvalues(A):
    A_old = np.copy(A)
    A_new = np.copy(A)
    diff = np.inf
    i = 0
    while (diff > tolerance) and (i < max_iter):
        A_old[:, :] = A_new
        Q, R = QR(A_old)
        A_new[:, :] = R @ Q
        diff = np.abs(A_new - A_old).max()
        i += 1
    eigvals = np.diag(A_new)
    return eigvals
```

The Hilbert matrix is now defined in the following function, taking N as an input parameter

In [5]:
```
def hilbert(n):
    H_matrix = np.array([[1.0 / (i+j-1) for j in range(1, n+1)] for i in range(1, n+1)])
    return H_matrix
```

In [6]:
```
# Define parameters
tolerance = 1.0e-9
max_iter = 1000

# Hilbert matrix
N = 10
HM = hilbert(N)

print("Input matrix:")
cols = ['c{}'.format(i) for i in range(1, 11)]
rows = ['r{}'.format(i) for i in range(1, 11)]
print(DataFrame(HM, columns=cols, index=rows))
```

```
Input matrix:
           c1        c2        c3        c4        c5        c6        c7  \
r1   1.000000  0.500000  0.333333  0.250000  0.200000  0.166667  0.142857
r2   0.500000  0.333333  0.250000  0.200000  0.166667  0.142857  0.125000
r3   0.333333  0.250000  0.200000  0.166667  0.142857  0.125000  0.111111
r4   0.250000  0.200000  0.166667  0.142857  0.125000  0.111111  0.100000
r5   0.200000  0.166667  0.142857  0.125000  0.111111  0.100000  0.090909
r6   0.166667  0.142857  0.125000  0.111111  0.100000  0.090909  0.083333
r7   0.142857  0.125000  0.111111  0.100000  0.090909  0.083333  0.076923
r8   0.125000  0.111111  0.100000  0.090909  0.083333  0.076923  0.071429
r9   0.111111  0.100000  0.090909  0.083333  0.076923  0.071429  0.066667
r10  0.100000  0.090909  0.083333  0.076923  0.071429  0.066667  0.062500

           c8        c9       c10
r1   0.125000  0.111111  0.100000
r2   0.111111  0.100000  0.090909
r3   0.100000  0.090909  0.083333
r4   0.090909  0.083333  0.076923
r5   0.083333  0.076923  0.071429
r6   0.076923  0.071429  0.066667
r7   0.071429  0.066667  0.062500
r8   0.066667  0.062500  0.058824
r9   0.062500  0.058824  0.055556
r10  0.058824  0.055556  0.052632
```

In [7]:
```
# Compute the Householder tridiagonal matrix of the Hilbert matrix
HM_TD = householder(HM.copy())

# Compute the eigenvalues
eig = eigenvalues(HM_TD.copy())

print("Householder Tridiagonal matrix:")
print(DataFrame(HM_TD, columns=cols, index=rows))
print("")
print("Eigenvalues: ")
print(eig)
```

```
Householder Tridiagonal matrix:
           c1        c2        c3        c4        c5            c6  \
r1   1.000000 -0.741463  0.000000  0.000000  0.000000  0.000000e+00
r2  -0.741463  0.996713  0.197579  0.000000  0.000000  0.000000e+00
r3   0.000000  0.197579  0.128941 -0.013796  0.000000  0.000000e+00
r4   0.000000  0.000000 -0.013796  0.007256  0.000733  0.000000e+00
r5   0.000000  0.000000  0.000000  0.000733  0.000333  3.000190e-05
r6   0.000000  0.000000  0.000000  0.000000  0.000030  1.160359e-05
r7   0.000000  0.000000  0.000000  0.000000  0.000000  8.975576e-07
r8   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000e+00
r9   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000e+00
r10  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000e+00

               c7            c8            c9           c10
r1   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
r2   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
r3   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
r4   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
r5   0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
r6   8.975576e-07  0.000000e+00  0.000000e+00  0.000000e+00
r7   2.916478e-07 -1.871030e-08  0.000000e+00  0.000000e+00
r8  -1.871030e-08  4.979516e-09 -2.500365e-10  0.000000e+00
r9   0.000000e+00 -2.500365e-10  5.166744e-11 -1.761854e-12
r10  0.000000e+00  0.000000e+00 -1.761854e-12  2.458938e-13

Eigenvalues:
[1.75191967e+00 3.42929548e-01 3.57418163e-02 2.53089077e-03
 1.28749614e-04 4.72968929e-06 1.22896774e-07 2.14743882e-09
 2.26674552e-11 1.09326665e-13]
```