PROBLEM 3 - Lanczos-Davidson

Gisela Martí Guerrero, ICC 2023

Python program that performs the Lanczos and Lanczos-Davidson diagonalization methods for large matrices applied to Hilbert matrix $N = 100$.

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

We start by implementing the Lanczos algorithm to generate the representation of a Hermitian NxN matrix H in a subspace. The Lanczos algorithm is an iterative method for finding a few eigenvalues and corresponding eigenvectors of a Hermitian matrix.

The function first initializes two matrices, L_v for Lanczos vectors and H_k for the Hamiltonian in the subspace, both with dimensions $(len(vg), len(vg))$, where vg is the initial guess vector. This vector is normalized and is set as the first Lanczos vector in the matrix L_v. It then performs the first iteration step of the Lanczos algorithm:

1. It computes the matrix-vector product $w = H \cdot L_v[0]$.
2. It computes the coefficient a as the inner product of the conjugate of w and $L_v[0]$.
3. It updates w to be orthogonal to the first Lanczos vector.
4. It sets the element $H_k[0, 0]$ in the tridiagonal matrix H_k to the value of a.

It then iterates over the remaining Lanczos vectors:

1. It computes the coefficient b based on the norm of w.
2. It normalizes the current Lanczos vector.
3. It updates w to be orthogonal to the current Lanczos vector and the previous one.
4. It computes the coefficients a and b for the tridiagonal matrix H_k.

The function returns (H_k, L_v), where H_k is the tridiagonal matrix representing the Hamiltonian in the subspace, and L_v is the matrix containing the Lanczos vectors.

In [2]:
```
def lanczos(H,vg):
    Lv=np.zeros((len(vg),len(vg)), dtype=complex)
    Hk=np.zeros((len(vg),len(vg)), dtype=complex)
    Lv[0]=vg/la.norm(vg)
    w=np.dot(H,Lv[0])
    a=np.dot(np.conj(w),Lv[0])
    w=w-a*Lv[0]
    Hk[0,0]=a
    for j in range(1,len(vg)):
        b=(np.dot(np.conj(w),np.transpose(w)))**0.5
        Lv[j]=w/b
        w=np.dot(H,Lv[j])
        a=np.dot(np.conj(w),Lv[j])
        w=w-a*Lv[j]-b*Lv[j-1]
        Hk[j,j]=a
        Hk[j-1,j]=b
        Hk[j,j-1]=np.conj(b)

    return (Hk,Lv)
```

Now we define the function "QR", which is the same one used in the previous exercise (Householder diagonalization), to perform a QR decomposition of a given matrix.

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
```

The same way, the function "eigenvalues" is also used using the same code from the previous exercise to find the eigenvalues of the given matrix using the QR decomposition methid.

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 = 100
HM = hilbert(N)

print("Input matrix:")
cols = ['c{}'.format(i) for i in range(1, 101)]
rows = ['r{}'.format(i) for i in range(1, 101)]
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
...        ...       ...       ...       ...       ...       ...       ...
r96   0.010417  0.010309  0.010204  0.010101  0.010000  0.009901  0.009804
r97   0.010309  0.010204  0.010101  0.010000  0.009901  0.009804  0.009709
r98   0.010204  0.010101  0.010000  0.009901  0.009804  0.009709  0.009615
r99   0.010101  0.010000  0.009901  0.009804  0.009709  0.009615  0.009524
r100  0.010000  0.009901  0.009804  0.009709  0.009615  0.009524  0.009434

            c8        c9       c10  ...       c91       c92       c93  \
r1    0.125000  0.111111  0.100000  ...  0.010989  0.010870  0.010753
r2    0.111111  0.100000  0.090909  ...  0.010870  0.010753  0.010638
r3    0.100000  0.090909  0.083333  ...  0.010753  0.010638  0.010526
r4    0.090909  0.083333  0.076923  ...  0.010638  0.010526  0.010417
r5    0.083333  0.076923  0.071429  ...  0.010526  0.010417  0.010309
...        ...       ...       ...  ...       ...       ...       ...
r96   0.009709  0.009615  0.009524  ...  0.005376  0.005348  0.005319
r97   0.009615  0.009524  0.009434  ...  0.005348  0.005319  0.005291
r98   0.009524  0.009434  0.009346  ...  0.005319  0.005291  0.005263
r99   0.009434  0.009346  0.009259  ...  0.005291  0.005263  0.005236
r100  0.009346  0.009259  0.009174  ...  0.005263  0.005236  0.005208

           c94       c95       c96       c97       c98       c99      c100
r1    0.010638  0.010526  0.010417  0.010309  0.010204  0.010101  0.010000
r2    0.010526  0.010417  0.010309  0.010204  0.010101  0.010000  0.009901
r3    0.010417  0.010309  0.010204  0.010101  0.010000  0.009901  0.009804
r4    0.010309  0.010204  0.010101  0.010000  0.009901  0.009804  0.009709
r5    0.010204  0.010101  0.010000  0.009901  0.009804  0.009709  0.009615
...        ...       ...       ...       ...       ...       ...       ...
r96   0.005291  0.005263  0.005236  0.005208  0.005181  0.005155  0.005128
r97   0.005263  0.005236  0.005208  0.005181  0.005155  0.005128  0.005102
r98   0.005236  0.005208  0.005181  0.005155  0.005128  0.005102  0.005076
r99   0.005208  0.005181  0.005155  0.005128  0.005102  0.005076  0.005051
r100  0.005181  0.005155  0.005128  0.005102  0.005076  0.005051  0.005025

[100 rows x 100 columns]
```

A random vector v of length N is generated to initialize the Lanczos algorithm with an arbitrary initial guess. Then the Lanczos algorithm is applied to the Hilbert matrix HM using the random vector v. The resulting tridiagonal matrix is stored in T, and the Lanczos vectors are stored in V. The "eigenvalues" function is called with the real part of the tridiagonal matrix T as its argument, to get the eigenvalues of the matrix.

In [7]:
```
v_rand = np.random.random(N)
T,V = lanczos(HM.copy(),v_rand)
eigenval = eigenvalues(T.real)
```

In [8]:
```
print("Tridiagonal matrix:")
print(DataFrame(T.real, columns=cols, index=rows))
print("")
print("Eigenvalues:")
print(eigenval)
```

```
Tridiagonal matrix:
            c1        c2        c3        c4        c5       c6   c7   c8  \
r1    1.034562  0.930871  0.000000  0.000000  0.000000  0.00000  0.0  0.0
r2    0.930871  1.245887  0.513891  0.000000  0.000000  0.00000  0.0  0.0
r3    0.000000  0.513891  0.725001  0.120593  0.000000  0.00000  0.0  0.0
r4    0.000000  0.000000  0.120593  0.211738  0.037631  0.00000  0.0  0.0
r5    0.000000  0.000000  0.000000  0.037631  0.054833  0.00068  0.0  0.0
...        ...       ...       ...       ...       ...      ...  ...  ...
r96   0.000000  0.000000  0.000000  0.000000  0.000000  0.00000  0.0  0.0
r97   0.000000  0.000000  0.000000  0.000000  0.000000  0.00000  0.0  0.0
r98   0.000000  0.000000  0.000000  0.000000  0.000000  0.00000  0.0  0.0
r99   0.000000  0.000000  0.000000  0.000000  0.000000  0.00000  0.0  0.0
r100  0.000000  0.000000  0.000000  0.000000  0.000000  0.00000  0.0  0.0

       c9  c10  ...  c91  c92  c93  c94       c95       c96       c97  \
r1    0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.000000
r2    0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.000000
r3    0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.000000
r4    0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.000000
r5    0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.000000
...   ...  ...  ...  ...  ...  ...  ...       ...       ...       ...
r96   0.0  0.0  ...  0.0  0.0  0.0  0.0  0.351655  0.264636  0.000019
r97   0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000019  0.819166
r98   0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.042950
r99   0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.000000
r100  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.000000  0.000000  0.000000

           c98       c99      c100
r1    0.000000  0.000000  0.000000
r2    0.000000  0.000000  0.000000
r3    0.000000  0.000000  0.000000
r4    0.000000  0.000000  0.000000
r5    0.000000  0.000000  0.000000
...        ...       ...       ...
r96   0.000000  0.000000  0.000000
r97   0.042950  0.000000  0.000000
r98   0.012315  0.000423  0.000000
r99   0.000423  0.047544  0.009657
r100  0.000000  0.009657  0.239817

[100 rows x 100 columns]

Eigenvalues:
[ 2.18269610e+00  2.18269610e+00  2.18269610e+00  2.18269610e+00
  2.18269610e+00  2.18269610e+00  2.18269610e+00  2.18269610e+00
  2.18269610e+00  2.18269610e+00  2.18269610e+00  2.18269610e+00
  2.18269610e+00  2.18269610e+00  2.18269610e+00  2.18269610e+00
  2.18269610e+00  8.21445561e-01  8.21445561e-01  8.21445561e-01
  8.21445561e-01  8.21445561e-01  8.21445561e-01  8.21445561e-01
  8.21445561e-01  8.21445561e-01  8.21445561e-01  8.21445561e-01
  8.21445561e-01  8.21445561e-01  8.21445561e-01  8.21445560e-01
  2.18595882e-01  2.18595882e-01  2.18595882e-01  2.18595882e-01
  2.18595882e-01  2.18595882e-01  2.18595882e-01  2.18595882e-01
  2.18595882e-01  2.18595882e-01  2.40348322e-01  2.18595882e-01
  2.18595882e-01  2.18595882e-01  4.92922510e-02  4.92922510e-02
  4.92922510e-02  4.92922510e-02  4.92922510e-02  4.92922510e-02
```

```
4.92922510e-02  4.92922510e-02  4.92922510e-02  4.92922510e-02
4.70751606e-02  1.00318122e-02  1.00318122e-02  1.00318122e-02
1.00318122e-02  1.00318122e-02  1.00318122e-02  1.00318122e-02
1.00318122e-02  1.00307683e-02  1.88506328e-03  1.88506328e-03
1.88506328e-03  1.88506328e-03  1.88506328e-03  1.88506328e-03
1.88506315e-03  3.30867810e-04  3.30867811e-04  3.30867809e-04
3.30867812e-04  3.30867811e-04  3.30842750e-04  5.46452976e-05
5.46453026e-05  5.46453018e-05  5.46453039e-05  5.44585027e-05
4.09338768e-05 -1.70009293e-05  8.53626581e-06  8.53628286e-06
8.53628272e-06  8.53906605e-06 -3.44355469e-06  4.74495695e-07
1.01027334e-06  1.26617626e-06  1.26617685e-06  1.26615904e-06
1.78868744e-07  1.79046810e-07  2.41612430e-08  4.65601614e-09]
```