Project report

AIT KHELIFA Tanina (n°21110432) and KAOUCH Abdelssamad (n°3801553)

1 Presentation of the files

You'll find 2 source files for the code. Both of them contain the exact same functions, the only difference is the types they were implemented for. As you can guess by their names, the *ver_double.c* file contains code dealing with double-precision floating-point numbers, and the *ver_mpfr.c* file contains code dealing with MPFR ones.

The most important functions are going to be described through algorithms in this report. However, much simpler functions (printing out matrices, initializing matrices...) are not : you will still find short descriptions of what they do in the comments of the code.

2 Proofs

Show that if Q is an orthogonal matrix, then QAQ^* has the same eigenvalues as the $n \times n$-matrix A.

Let $\lambda \in \mathbb{C}$ be the eigenvalue of A, with non zero eigenvector $v \in \mathbb{C}^n$.
By definition, we have $Av = \lambda v$. If v is non zero, then we can say that $Qv \neq 0$ for any orthogonal matrix Q because $0 \neq v = Q^*Qv$. Let's now observe $QA(Q^*Qv)$, with Q orthogonal :
$QA(Q^*Qv) = QAv = \lambda Qv$. From this, we can see that QAQ^* has Qv as an eigenvector, with eigenvalue λ.
Therefore, we showed that if Q is orthogonal, then QAQ^* has the same eigenvalues as the $n \times n$-matrix A.

For the following proofs, we define $A_{i,j}$ to be the coefficient in position (i, j) in the matrix A.

Show that there exists a rotation matrix $G_{n-1,n}$ such that both $G_{n-1,n}A$ and $A' = G_{n-1,n}AG^*_{n-1,n}$ have a 0 coefficient in position $(n, 1)$

Let's build the following $n \times n$-matrix $G_{n-1,n}$: $\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & c & s \\ 0 & \cdots & \cdots & -s & c \end{pmatrix}$

with $c = \frac{A_{n-1,1}}{\sqrt{A^2_{n,1}+A^2_{n-1,1}}}$ and $s = \frac{A_{n,1}}{\sqrt{A^2_{n,1}+A^2_{n-1,1}}}$.

The coefficient in position $(n, 1)$ of the matrix $G_{n-1,n}A$ will be exactly equal to :
$c \times A_{n,1} - s \times A_{n-1,1} = 0$.

The coefficient in position $(n, 1)$ of the matrix $G_{n-1,n}AG^*_{n-1,n}$ will be exactly equal to :
$(G_{n-1,n}A)_{n,1} \times 1 + (G_{n-1,n}A)_{n,2} \times 0 + \ldots + (G_{n-1,n}A)_{n,n} \times 0 = 0 \times 1 = 0$.

Therefore, such a $G_{n-1,n}$ matrix exists.

Show that there exists a rotation matrix $G_{n-2,n-1}$ such that both $G_{n-2,n-1}A'$ and $A'' = G_{n-2,n-1}A'G^*_{n-2,n-1}$ have a 0 coefficient in position $(n-1,1)$

Let's build the following $n \times n$-matrix $G_{n-2,n-1}$: $\begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & 1 & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & c & s & \vdots \\ \vdots & \cdots & \cdots & \cdots & -s & c & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 1 \end{pmatrix}$

with $c = \frac{A'_{n-2,1}}{\sqrt{A'^2_{n-1,1}+A'^2_{n-2,1}}}$ and $s = \frac{A'_{n-1,1}}{\sqrt{A'^2_{n-1,1}+A'^2_{n-2,1}}}$.

The coefficient in position $(n-1,1)$ of the matrix $G_{n-2,n-1}A'$ will be exactly equal to :
$c \times A'_{n-1,1} - s \times A'_{n-2,1} = 0$.

The coefficient in position $(n-1,1)$ of the matrix $G_{n-2,n-1}A'G^*_{n-2,n-1}$ will be exactly equal to :
$(G_{n-2,n-1}A')_{n-1,1} \times 1 + (G_{n-2,n-1}A')_{n-1,2} \times 0 + \ldots + (G_{n-2,n-1}A')_{n-1,n} \times 0 = 0 \times 1 = 0$.

Therefore, such a $G_{n-2,n-1}$ matrix exists.

Show that there exists a rotation matrix $G_{n-1,n}$ such that both $G_{n-1,n}A''$ and $A''' = G_{n-1,n}A''G^*_{n-1,n}$ have a 0 coefficient in position $(n,2)$

Let's build the following $n \times n$-matrix $G_{n-1,n}$: $\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & c & s \\ 0 & \cdots & \cdots & -s & c \end{pmatrix}$

with $c = \frac{A''_{n-1,2}}{\sqrt{A''^2_{n,2}+A''^2_{n-1,2}}}$ and $s = \frac{A''_{n,2}}{\sqrt{A''^2_{n,2}+A''^2_{n-1,2}}}$.

The coefficient in position $(n,2)$ of the matrix $G_{n-1,n}A''$ will be exactly equal to :
$c \times A''_{n,2} - s \times A''_{n-1,2} = 0$.

The coefficient in position $(n,2)$ of the matrix $G_{n-1,n}A''G^*_{n-1,n}$ will be exactly equal to :
$(G_{n-1,n}A'')_{n,1} \times 0 + (G_{n-1,n}A'')_{n,2} \times 1 + \ldots + (G_{n-1,n}A'')_{n,n} \times 0 = 0 \times 1 = 0$.

Therefore, such a $G_{n-1,n}$ matrix exists.

3 Algorithms

3.1 Making a matrix upper Hessenberg

Algorithm 1 GenerateRotationMatrix

```
Input : $A$, a $n \times n$ matrix, integers $i$ and $j$
Output : The rotation matrix $Q$ so that both $QA$ and $Q^*AQ$ have 0 in position (i, j)
  $Q \leftarrow Id_n$
  $c \leftarrow \frac{A_{i-1,j}}{\sqrt{A_{i,j}^2+A_{i-1,j}^2}}$
  $s \leftarrow \frac{A_{i,j}}{\sqrt{A_{i,j}^2+A_{i-1,j}^2}}$
  $Q_{i,i} \leftarrow c$
  $Q_{i-1,i-1} \leftarrow c$
  $Q_{i-1,i} \leftarrow s$
  $Q_{i,i-1} \leftarrow -s$
Return $Q$
```

In the C files, this algorithm is implemented through the function *rotation_matrix*.

Algorithm 2 MakeUpperHessenberg

```
Input : $A$, a $n \times n$ matrix
Output : The $n \times n$ upper Hessenberg matrix $R$ so that $R$ and $A$ have the same eigenvalues.
  $R \leftarrow A$
  for $j$ from 1 to $n-1$ do
      for $i$ from $n$ to $j+1$ do
          $Q \leftarrow GenerateRotationMatrix(R, i, j)$
          $R \leftarrow QRQ^*$
      end for
  end for
Return $R$
```

In the C files, this algorithm is implemented through the function *algo_hessenberg*.

3.2 Computing the eigenvalues

Algorithm 3 ComputeEigenvalues

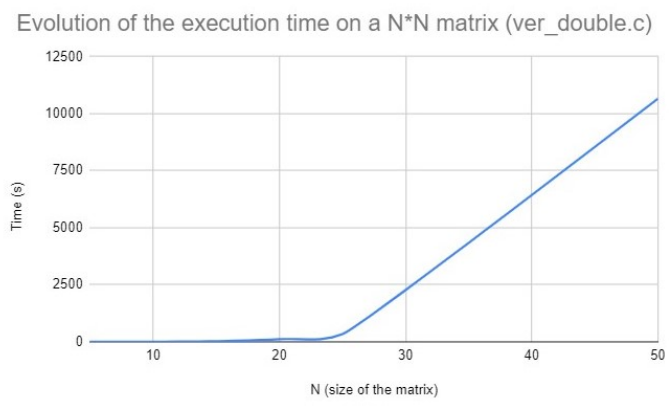
```
Input : $A$, a $n \times n$ matrix, integer $max\_iterations$
Output : The $n \times n$ upper triangular matrix $M$. The values on its diagonal are the eigenvalues of
         A.
  $i \leftarrow 0$
  $M \leftarrow MakeUpperHessenberg(A)$
  while $i \leq max\_iterations$ and $M$ is not upper triangular do
      $Q, R \leftarrow Givens'Algorithm(M)$        ▷ (Using the exact algorithm we saw in the course)
      $M \leftarrow Q^*MQ$
  end while
Return $M$
```

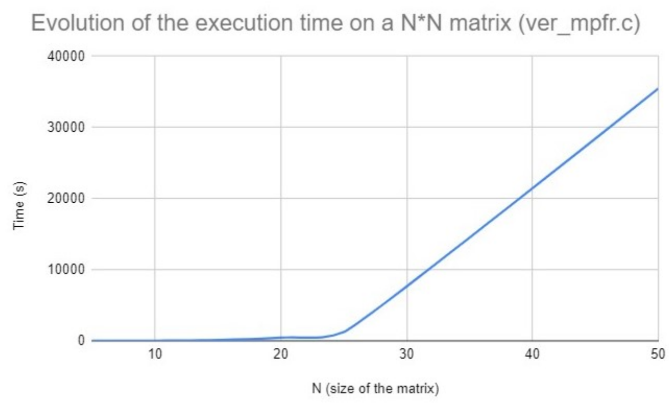
In the C files, this algorithm is implemented through the function *eigenvalues*.

4 Benchmarking

In order to have an idea of the time complexity for the computation of the eigenvalues, we executed the program on matrices of sizes 5×5, 10×10, 15×15, 20×20, 25×25 and 50×50. We did not try executing it on larger matrices, as our code is far from optimized.

Below, you can find the graph resulting from this benchmark. Here are the parameters used for the tests :
- Values in the matrices were integers (with a cast to double or stored in mpfr_t)
- Values in the initialized matrix ranged from 1 to 10, both included
- The maximum number of iteration was set to 10000
- The algorithm was set to stop when the values on the 1st sub-diagonal were lesser than $\epsilon = 0.0001$
- For the MPFR version, float precision was set to 100





We can clearly see that dealing with MPFR floating-point numbers takes more time (on average, a little more than 3 time longer) than working with double-precision ones.
Still the execution time seems to get longer with the same factor for both types, but on a different scale.

5 Remarks and conclusion

While using MPFR floating-point numbers, we noticed a visible improvement in the quality of the eigenvalues computed by the program. Moreover, when making a matrix upper-Hessenberg there was less coefficients that had an "approximately" zero value (by that, we mean numbers such as $\pm 0.000 \ldots X \ldots$).

As for possible improvements in out code, memory-wise, we tried to allocate as little space as we could and reuse that space whenever it was possible.

Performance-wise, we think there's room for improvement. A specialized function designed for matrix product involving an upper-Hessenberg matrix would improve the time complexity of the program. There may be calls to the function $copyMatrix()$ that could be avoided, had we used the space allocated more wisely. Getting rid of these calls would reduce the execution time of the program, but it could create more data dependencies within the code.

Overall, we would say we are satisfied with the code produced as it works and returns a satisfying result. However, its quality is not the best, as the execution time gets too long really easily.