Mandatory Assignment 1
MAT-INF4130

Ivar Haugaløkken Stangeby

September 11, 2017

Problem 1

In this problem we consider how to invert a lower triangular matrix. Recall that a lower triangular matrix has a lower triangular inverse. Assume that $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ is a lower triangular matrix with $\boldsymbol{B}$ its lower triangular inverse. In terms of block matrices we have that

$$\boldsymbol{A}\boldsymbol{B} = \boldsymbol{A}[\boldsymbol{b}_1, \dots, \boldsymbol{b}_n] = [\boldsymbol{e}_1, \dots, \boldsymbol{e}_n] = \boldsymbol{I}. \tag{1}$$

Considering equation $\boldsymbol{A}\boldsymbol{b}_k = \boldsymbol{e}_k$ and looking at the k-th row in $\boldsymbol{A}$ we see that

$$\sum_{i=1}^{n} a_{ki} b_{ii} = 1. \tag{2}$$

Since both $\boldsymbol{A}$ and $\boldsymbol{B}$ are lower triangular, this sum reduces to $a_{kk}b_{kk} = 1$, and consequently $b_{kk} = 1/a_{kk}$ for all $k = 1, \dots, n$. In order to compute the rest of $\boldsymbol{b}_k$, consider the following system of matrices

$$\begin{bmatrix} \boldsymbol{A}_{[k]} & \boldsymbol{0} \\ \boldsymbol{C} & \tilde{\boldsymbol{A}} \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_k^1 \\ \boldsymbol{b}_k^2 \end{bmatrix} = \boldsymbol{e}_k. \tag{3}$$

Here $\boldsymbol{A}_{[k]}$ is the *principal leading submatrix* of $\boldsymbol{A}$, while

$$\boldsymbol{b}_k^1 = [0, \dots, 0, b_{kk}], \qquad \boldsymbol{b}_k^2 = [b_{k+1,k}, \dots, b_{nk}]. \tag{4}$$

Expanding this system yields the equation

$$\tilde{\boldsymbol{A}}\boldsymbol{b}_k^2 = -\boldsymbol{C}\boldsymbol{b}_k^1, \tag{5}$$

where the right hand side reduces to $-\boldsymbol{A}((k+1){:}n, k)b_{kk}$ by the properties of $\boldsymbol{e}_k$. We can therefore compute $\boldsymbol{B}$ by computing the diagonal elements, then solve the linear system

$$\boldsymbol{A}((k+1){:}n, (k+1){:}n)\boldsymbol{b}_k((k+1){:}n) = -\boldsymbol{A}((k+1){:}n, k)b_{kk} \tag{6}$$

for $k = 1, \dots, n-1$.

Numerical Considerations

For numerical efficiency it is feasible to do these computations inplace. From Equation (3) we see that in the k-th step of the computation we do not use the top right part of the matrix. Furthermore, the diagonal elements are only used once and not in any subsequent computations, hence we may safely replace these values by others. We can therefore store the vectors $\boldsymbol{b}_k^2$ in place of $\boldsymbol{A}(k, k:n)$. After the computations are done, $\boldsymbol{B}$ may be extracted from $\boldsymbol{A}$ by transposing and zeroing out the upper diagonals.

Problem 2

We are interested in the number of operations for computing the inverse of a lower triangular matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ following the algorithm described above. For $k = 1, \dots, n-1$, we compute the diagonal element b_{kk} and solve the corresponding linear system as given in Equation (6). Each such system can be solved using a solver relying on the fact that the matrix is lower triangular. Hence for $i = 1, \dots, n-k$ there are i multiplications and $i-1$ additions. Letting N denote the total number of operations, we have

$$N = \sum_{k=1}^{n-1}\sum_{i=1}^{n-k} 2i - 1 = \sum_{k=1}^{n-1}(n-k)^2 \tag{7}$$

and the leading term is given by

$$\approx \int_1^{n-1} (n-k)^2 \, \mathrm{d}k \approx \frac{1}{3} n^3. \tag{8}$$

The algorithm therefore has a time complexity of $\mathcal{O}(n^3)$.

Problem 3

```
from numpy import *

n = 8
A = matrix(random.random((n, n)))
A = triu(A)
U = A.copy()
for k in range(n-1, -1, -1):
  U[k, k] = 1 / U[k, k]
  for r in range(k-1, -1, -1):
     U[r, k] = -U[r, (r+1) : (k+1)] * U[(r+1) : (k+1), k] / U[r, r]
print(U*A)
```

Listing 1: Computing the upper triangular inverse of an upper triangular matrix. Implemented in Python.

We are tasked with discussing the code snipped given in Listing 1. This is a Python-implementation of the algorithm discussed above for computing the inverse of a triangular matrix. In this case, an upper triangular one. The variables `r` and `k` represent the row and column index respectively. Note that the computation is performed in-place, and the inverse of $\boldsymbol{A}$ is stored in memory allocated to storing $\boldsymbol{A}$. The final product `U * A` is the identity matrix.