Mandatory Assignment 2
MAT-INF4130

Ivar Haugaløkken Stangeby

September 28, 2017

1 Introduction

In this assignment we discuss implementations of various numerical algorithms for solving a linear system of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where $\boldsymbol{A} \in \mathbb{C}^{n \times n}$. The three algorithms of choice are:

1. Gaussian elimination;
2. Gaussian elimination with pivoting; and
3. Householders triangulation.

From the analysis of the algorithms, one can deduce that pivoting should incur no additional cost, computationally speaking. Furthermore, Householders triangulation should scale approximately at twice the rate as Gaussian elimination. We will try to verify these claims in the following.

2 Implementation

All implementations has been done in Python in a small library called NULL (NUmerical Linear aLgebra). Each method has been tested with the following

linear system $\boldsymbol{Ax} = \boldsymbol{b}$ where

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 2 & 1 \\ 1 & 2 & 3 \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{b} = \begin{bmatrix} 9 \\ 9 \\ 14 \end{bmatrix}. \tag{2}$$

In this case, the solution is known to be $\boldsymbol{x} = (1, 2, 3)^T$, and the implementation can be tested by running `pytest matrixsolvers.py -m test_system` on the command-line.

Since naive gaussian elimination leads to an LU factorization and our matrix $\boldsymbol{A}$ has a singular leading submatrix — no such factorization exists — I have decided to interpret the distinction between Gaussian elimination and Gaussian elimination with pivoting as the former only using row switches, while the latter stores these row operations in a permuation matrix.

3 Results

Numerical Stability

By considering the matrix equation

$$\begin{bmatrix} \varepsilon & 2 \\ 1 & 1 \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \tag{3}$$

for $\varepsilon = 10^{-12}, 10^{-14}, 10^{-16}$ we may uncover some information about the numerical stability of the above methods. By running the function `stability()`, we obtain the relative errors

$$E_{\text{rel}} \coloneqq \frac{\|\boldsymbol{Ax} - \boldsymbol{b}\|}{\|\boldsymbol{b}\|}. \tag{4}$$

The results are given in table 1. As ε tends to zero, the solution tends to

$$\boldsymbol{x} = (5/2, 3/2)^T \tag{5}$$

and from the table we see that none of the algorithms above struggle with numerical instability, however the Householder triangulation does have an insignificant relative error.

ε	GE	GEP	Householder
10^{-12}	0	0	$8.88178 \cdot 10^{-17}$
10^{-14}	0	0	$8.88178 \cdot 10^{-17}$
10^{-16}	0	0	$8.88178 \cdot 10^{-17}$

Table 1: Relative errors when solving the system in eq. (3) for the three algorithms discussed above.

Time Complexity

In order to examine the time complexity of the three algorithms we run the algorithms on $n \times n$ random matrices where $n = 50 \cdot 2^m$ for $m = 0, \ldots, 5$. Each solver is ran $N = 5$ times and the average of the elapsed time is computed. The results are plotted, and shown in fig. 1.

We would expect that Gaussian Elimination and Gaussian Elimination with pivoting perform about equally. However, according to the results, the pivoting perform much better. This might be due to implementation, as the pivoting method is vectorized to a higher extent and employs a PLU-factorization.

We also expect the Householder triangulation method to scale at twice the cost of the Gaussian elimination, and according to the results, we see that it does indeed scale approximately at twice the rate of the Gaussian Elimination with pivots.

A Code Snippets

In the following we list a select few of the algorithms used. The complete source code can be found at the following link:
`https://github.com/qTipTip/NULL/blob/master/NULL/matrices.py`

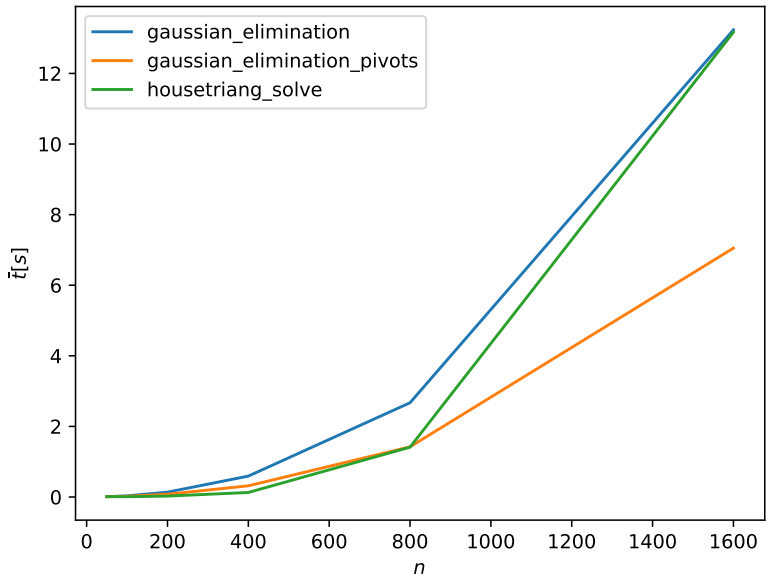

Figure 1: The average runtime for each algorithm as a function of n.

```
def gaussian_elimination(A, b):
    """
    Given a (nxn) matrix A and a right hand side b, computes x such that
    Ax = b.
    """

    m, n = A.shape
    U = A.copy()
    b = b.copy()

    # forward sweep, reduce A to a upper triangular matrix
    for k in range(min(m, n)):
        swap = np.argmax(np.abs(U[k:, k])) + k
        if U[swap, k] == 0:
            raise ValueError('Singular matrix')
        U[[k, swap], :] = U[[swap, k], :]
        b[[k, swap]] = b[[swap, k]]

        for i in range(k + 1, m):
            factor = U[i, k] / U[k, k]
            b[i] = b[i] - factor*b[k]
            U[i, k+1:] = U[i, k+1:] - U[k, k+1:] * factor
            U[i, k] = 0

    # solve by back subistitution
    x = rbackwardsolve(U, b, m)

    return x
```

Listing 1: Gaussian elimination with row interchanges.

```
def gaussian_elimination_pivots(A, b):
    """
    Given an nxn matrix A and a right hand side b, computes
    the matrices P, L, U such that A = PLU,
    then computes x such that LUx = (P.T)b.
    """

    P, L, U = PLU(A)
    n,_ = A.shape
    y = rforwardsolve(L, (P.T).dot(b), n)
    x = rbackwardsolve(U, y, n)

    return x
```

Listing 2: Gaussian Elimination with partial pivoting

```
def housetriang_solve(A, b):
    """
    Given an nxn matrix A and a right hand side b, computes the matrix R and
    the vector c such that Rx = c, where R is upper triangular. Hence can be
    solved by back-substitution.

    n, _ = A.shape
    b = np.reshape(b.copy(), (n, 1))
    R, c = housetriang(A, b)
    x = np.reshape(rbackwardsolve(R, c, n), (n,))


    return x
```

Listing 3: Householder triangulation based solver