Homework Sheet 1

VU Numerical Algorithms, SoSe 2018

<u>*due date: 23.4.2018, 18:00*</u>

- Please submit an anonymized version of your report without your name or your Matrikelnummer appearing in the document.

Programming Exercise

The task is to implement an LU factorization-based linear solver in OCTAVE and to evaluate its accuracy for various test matrices. The solver consists of computing the LU decomposition of a square $n \times n$ double precision matrix A such that $A = LU$ with lower triangular L and upper triangular U and subsequent forward and back substitution. In particular:

Part I - LU Decomposition (3 points)

First implement the standard "scalar" (unblocked) algorithm (i.e. three nested loops) **with partial pivoting**.

- U is contained in the upper triangle (plus diagonal) of A, and the diagonal entries of L are all 1. The subdiagonal entries of L are given by the scalars m_{ik} (i.e. $L(i,k) = m_{ik}$). For storage efficiency, we can store L in the lower triangle of A, and thus $A(i,k)$ has to be overwritten with m_{ik}.

Detailed remarks:

1. *Accuracy*: Verify the correctness of your LU factorization by evaluating the relative residual

   $$R = \frac{\|P^T LU - A\|_1}{\|A\|_1}$$

   where $\|\cdot\|_1$ is the maximum absolute column sum of a matrix:

   $$\|M\|_1 = \underset{j=1,\ldots,n}{max} \sum_{i=1}^{n} |M_{ij}|$$

Algorithm 1 Pseudo-Code LU Decomposition with partial pivoting

```
for k = 1 to n − 1 do
  Find index p such that
  |a_pk| ≥ |a_ik| for k ≤ i ≤ n
  if p ≠ k then
    interchange rows k and p
  end if
  if a_kk = 0 then
    continue with next k
  end if
  for i = k + 1 to n do
    m_ik = a_ik / a_kk
  end for
  for j = k + 1 to n do
    for i = k + 1 to n do
      a_ij = a_ij − m_ik a_kj
    end for
  end for
end for
```

Plot these residuals R for all problem sizes you experimented with. *When plotting residuals, always use a* ***logarithmic scale*** *along the* ***y-axis!***

2. Use randomly generated matrices A as input, but please specify clearly in your report how you generated your test matrices!

3. Produce separate routines (i. e., preparation of the input matrices, LU decomposition). Write modular code! Use consistent interfaces for your routines:
   a) Input: A, n
   b) Output: $n \times n$ matrices L and U stored in the array A ($A = LU$) and the permutation matrix P.

4. What is NOT allowed:
   a) Do not use any existing code which you did not write yourself!
   b) Do not try to exploit any special structure in the input data. The code has to be generic and should run for all possible input matrices A.

Part II - Solving a Triangular Linear Systems (1 point)

1. **Forward substitution**: Write a routine which solves a given $n \times n$ lower triangular linear system $Lx = b$ for x.

2. **Back substitution**: Write another routine which solves a given $n \times n$ upper triangular linear system $Ux = b$ for x.

3. Evaluate the accuracy of your codes for increasing n in terms of the relative residual and the relative forward error. *(For the definition of relative residual and relative forward error please see Part III!)*

   For these experimental evaluations, use randomly generated (non-singular) L and U and determine b such that the exact solution x is a vector of all ones: $x = (1, 1, \ldots, 1, 1)^T$.

Part III - Numerical Accuracy of LU-Based Linear Solver (4 points)

The main purpose of this part is to experimentally evaluate the numerical accuracy of the linear systems solver you implemented in Parts I and II for different test matrices and to compare it with the built-in solver from OCTAVE. You can solve a linear system $Ax = b$ for x using the \ operator (e.g. $x = A \setminus b$).

1. Take your LU factorization from Part I and combine it with your triangular linear systems solvers from Part II in order to get a complete LU-based linear solver.

2. Input data for your experiments:

   a) Generate random test matrices S with entries uniformly distributed in the interval [-1,1].

   b) Generate test matrices H which are defined by

   $$H_{ij} := \frac{1}{i + j - 1} \quad for\ i = 1, \ldots, n \text{ and } j = 1, \ldots, n.$$

   c) In all your test cases, determine the corresponding right hand side b of length n such that the exact solution x of the linear system is a vector of all ones: $x = (1, 1, \ldots, 1, 1)^T$.

3. Solve the linear systems $Sx = b$ and $Hx = b$ with your LU-based linear solver and the built-in OCTAVE solver and evaluate the numerical accuracy of the computed solution.

   a) *Problem sizes*: Start with $n = 2, 3, 4, 5, \ldots, 10$ then incease in increments of 5. For $n > 50$ you can further increase the increment. The largest value of n should be as large as possible (so that your code terminates within a reasonable time).

   b) *Accuracy*: For the computed solution $\hat{x}$, evaluate the relative residual r:

   $$r := \frac{||M\hat{x} - b||_1}{||b||_1}$$

   (M is S or H) as well as the relative forward error f:

   $$f := \frac{||\hat{x} - x||_1}{||x||_1}.$$

4. For both your and the OCTAVE solver generate the following plots for the different test matrices:

   a) Relative residual and relative forward error in $\hat{x}$ vs. n: One figure for both accuracy metrics for matrix type S, another figure for both accuracy metrics for matrix type H.

5. Interpret and explain your experimental results in your report. Do you think that there is a fundamental difference in the numerical accuracy which your LU-based linear solver achieves for the two types of test matrices? If yes, explain the reasons for this difference. How does your solver compare to the OCTAVE version?