Homework Sheet 1

VU Numerical Algorithms, WiSe 2021

due date: 5.11.2021, 18:00

Basic Guidelines - please read carefully!

- Your homework report is *very important* for the grading of your homework. Your report has to provide a clear, well structured, compact and understandable summary of what you did for your homework and what the results are (including *well explained* and *understandable* figures of your experimental results).
- For the programming parts, a correct implementation is *not* sufficient. If you do not provide a clear and well readable summary of your implementation and of your experiments, you will not get a good grade on the programming part.
- Due to the number of students in class, we cannot debug your code in case it produces wrong results. As a consequence, incorrect code in a programming part will always lead to zero points for this part, since it is not possible to find your programming errors and distinguish between different implementation errors.
- In total, your report for this homework sheet may at most have *five pages* (just for the programming part, not including the Paper-and-Pencil exercises, which should be in the same document though). Anything beyond the fifth page will not be considered in the grading.

Paper-and-Pencil Exercises

1. (*1 point*) The IEEE standard 754 (known as the "floating-point standard") specifies the 128-bit word as having 15 bits for the exponent.

   What is the length of the fraction (mantissa)? What is the machine epsilon? How many significant decimal digits does this word have?

   Why is quadruple precision more than twice as accurate as double precision, which is in turn more than twice as accurate as single precision?

2. (*2 points*) The IEEE standard 754 specifies that all arithmetic operations are to be performed as if they were first calculated to infinite precision and then rounded according to one of four modes. The default rounding mode is to round to the nearest representable number, with rounding to even (zero least significant bit) in the case of a tie.

   Which of the following statements is true in IEEE arithmetic, assuming that a and b are normalized floating point numbers and that no exception occurs in the stated operations? In each case, give a short explanation.

   a) $\mathrm{fl}(a \text{ op } b) = \mathrm{fl}(b \text{ op } a), \qquad \text{op } = +, *.$
   b) $\mathrm{fl}(a + a) = \mathrm{fl}(2 * a)$
   c) $\mathrm{fl}((a + b) + c) = \mathrm{fl}(a + (b + c))$

3. (*3 points*) Using the standard model of floating point arithmetic introduced in the lecture, derive forward and backward error bounds for a simple summation,

$$S_n = \sum_{i=1}^{n} x_i \,,$$

   in which the x_i are machine numbers and the sum is evaluated from left to right[1].

Programming Exercise

Prerequisites

1. *Basics*:
   - Please use Octave[2] *version 4.4* or higher and indicate the Octave version in your report. Your submission will be evaluated.
   - Do not import additional packages and do not use global variables.
   - Pay attention to the interface definitions, i.e., use the specified terms. In/output parameters must be in the specified order.
   - Your routines should always check the number and types of input arguments.
   - Do not plot results in predefined routines! Plot results in scripts or self defined routines only.
   - Do not exploit any special structure in the input data. Your routines must be generic and have to work for all $n > 1$.
   - Do not use any existing code which you did not write yourself!
   - You can define your own routines in order to write modular code but please stay consistent with the predefined interface.

[1] E.g. $S_4 = ((x_1 + x_2) + x_3) + x_4$.
[2] Octave download page: https://www.gnu.org/software/octave/download.html

- Further, you can add more parameters, so you do not need to recompute parts of the program and accelerate your computation. However, you should add default parameters, so the program still works with the predefined parameters given in the assignment description.

2. *Interface*:
   - Mandatory for **all Parts**:
     a) Write a script *assignment1.m* to test your routines and plot your results.

     b) Create a file *accuracy.m* of the following form:

     ```
     [z] = accuracy(X, Y)
     ```

     - Input: X and Y are either both $n \times n$ matrices or both vectors of size n.
     - Output: scalar z, with

     $$z := \frac{||X - Y||_1}{||Y||_1}.$$

     Remark: Use this routine to verify the correctness of your *LU* factorization in **Part I** and to compute the *relative forward error* in **Part II** resp. **Part III**.

     c) Create a file *residual.m* of the following form:

     ```
     [r] = residual(A, x_hat, b)
     ```

     - Input: A is an $n \times n$ matrix, $\hat{x}$, b are vectors of size n.
     - Output: scalar r, with

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

     Remark: Use this routine to verify the correctness of your *LU* factorization in **Part I** and to compute *relative residual* in **Part II** resp. **Part III**.

   - Mandatory for **Part I**:
     a) Create a file *plu.m* for the *LU* factorization with partial pivoting:

     ```
     [A, P] = plu(A, n)
     ```

     - Input: $n \times n$ matrix A, n

- Output: $n \times n$ matrices L and U stored in the array A ($A = P^T LU$) and the permutation matrix P.

Remark:

- Mandatory for **Part II**:

  a) Create two files *solveL.m* / *solveU.m* for solving a lower / upper triangular system:

  ```
  [x] = solveL(B, b, n)
  [x] = solveU(B, b, n)
  ```

  - Input: $n \times n$ matrix B, the right hand side vector b of size n, n
  - Output: the solution vector x.

  Remark: The input matrix B has special structure $B = L + U - I$, where L (with fixed ones in the diagonal) and U are lower resp. upper triangular matrices and I is the Identity.

- Mandatory for **Part III**:

  a) Create a file *linSolve.m* for solving a linear system:

  ```
  [x] = linSolve(A, b, n)
  ```

  - Input: $n \times n$ nonsingular matrix A, the right hand side vector b of size n, n.
  - Output: the solution vector x.

  Remark: This routine must incorporate **plu.m**, **solveL.m** and **solveU.m** from previous Parts!

3. *Submission*:

- Upload a single zip archive with all your source code files and your report (as a single PDF file named *report.pdf* with all plots and discussions of results as well as your solution to the paper-and-pencil exercises) on the course page in Moodle.
- Name your archive `a<matriculation number>_<last name>.zip` (e.g. *a01234567_Mustermann.zip*)
- Directories in the archive are not allowed.
- A complete submission has to include the following files:

  a) Routines: *accuracy.m, linSolve.m, plu.m, solveL.m, solveU.m*, self defined routines (optional)

b) Script: *assignment1.m*

c) Documentation: *report.pdf*

The **programming exercise** consists of implementing 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 $PA = LU$ with a permutation matrix P, lower triangular L and upper triangular U and subsequent forward and back substitution. In particular:

Part I - LU Decomposition (4 points)

First implement the standard "scalar" (unblocked) algorithm (i.e. three nested loops) in two versions: (a) **with partial pivoting**, and (for comparison purposes) (b) **without partial pivoting**.

Algorithm 1 Pseudo-Code LU Decomposition with partial pivoting

```
for $k = 1$ to $n - 1$ do
    Find index $p$ such that
    $|a_{pk}| \geq |a_{ik}|\ for\ k \leq i \leq n$
    if $p \neq k$ 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
```

- 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} (see Algorithm 1).

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:

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

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!

Part II - Solving Triangular Linear Systems (2 points)

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. Start with $n = 2, 3, 4, 5, \ldots, 10$, then increase 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), *but at least* $n \geq 1000$. *(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 *with and without partial pivoting* for different test matrices and to compare it with the built-in solver from OCTAVE (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

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

   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: $x = (1, 1, \dots, 1, 1)^T$.

3. Solve the linear systems $Sx = b$ and $Hx = b$ with your LU-based linear solver *with and without partial pivoting* and also with the built-in OCTAVE solver and compare the numerical accuracy of the computed solutions.
   a) *Problem sizes*: Start with $n = 2, 3, 4, 5, \dots, 10$, then increase 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), *but at least* $n \geq 1000$.
   b) *Accuracy*: For the computed solution $\hat{x}$, evaluate the relative residual $\hat{r}$:

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

   (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 (including with and without partial pivoting), another figure for both accuracy metrics for matrix type H (including with and without partial pivoting).

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?

Octave cheat sheet

Below is a list of recommendations that might help with your implementation:

- You should at least be aware of the following routines provided by Octave:

  $validateattributes$, $nargin$, $norm$, $tril$, $triu$, $semilogy$, …

  All Octave routines are described here:
  `https://octave.sourceforge.io/list_functions.php`

- We recommend vectorization to simplify and optimize your code (i.e., eliminate for loops whenever possible):

  `https://octave.org/doc/v4.0.1/Basic-Vectorization.html`

- For efficiency, the LU factorization algorithm usually operates on a permutation vector (instead of a whole matrix) during the factorization process and creates the permutation matrix at the very end. You might consider using a permutation vector as well (although, this is not mandatory). Creating permutation matrices is explained here:

  `https://octave.org/doc/v4.4.1/Creating-Permutation-Matrices.html`