Lab-session week 15 (1MA930/1MA931, VT2024)

Suggested exercises:

1. **Iterative methods and sparse matrices.**

   a) Read/skim through page 113 about the advantages/drawbacks of direct methods and iterative methods. At the end of the page, see the definition of a *full matrix* and a *sparse matrix.*

   b) Read/skim through Example 2.25 on page 114–115. Try to understand the code in Program 2.1 with the help of the MATLAB documentation. In particular, find out what the new commands `ones`, `zeros` and `spdiags` do by writing `help ones`, `help zeros` and `help spdiags`. Then copy the code of Program 2.1 to a separate file and run it (syntax: `[a,b]=sparsesetup(n);`) with $n = 6$.

   Write `a` in the command window afterwards. How is `a` stored? Compare to what happens when you write `afull=full(a)`. Compare the outputs of `spy(afull)` and `spy(a)`.

   c) Try to understand the code in Program 2.2 (page 115). Can you see that it does the same thing as the algorithm on page 108?

   d) Copy and run the code in Program 2.2. What is the solution after one iteration ($k = 1$)? After two? Six? Compare with Example 2.24, top of page 111.

   e) Investigate if there is a difference in the execution time when you use `a` as input compared to if you use `afull` as input. To see a difference, you need to construct `a` and `b` using bigger n, e.g. $n = 1000$ or more. To do this; learn how to use the commands `tic` and `toc` (write *help tic* in the command window).

2. **Conditioning.**

   So-called Hilbert matrices are known to be badly conditioned.

   a) Create the matrix `H=hilb(7)` in MATLAB, which gives you the Hilbert matrix of order 7. Create the right-hand-side $b = [10, 10, 10, 10, 10, 10, 10]^T$ using the command `b=10*ones(7,1)`.

   b) Solve the system $Hx = b$ using `x=H\b`. Compare with the exact solution, produced by `xexact=invhilb(7)*b`.

   c) Find the backward error, the forward error and the error magnification factor. Use the command `norm(b,'inf')` to compute the max-norms. Compare the error magnification factor with the condition number of H, computed using `cond(H,'inf')`.

3. **Complexity.**

   LU-factorization uses the fact that triangular matrices are easy to solve. Try this in MATLAB!

Continue on next page!

a) Construct an $n \times n$ matrix A with random entries, using the function `rand`. For example, let $n = 5$. Construct a right-hand-side $n \times 1$ vector b, also with random entries. Solve the system $Ax = b$ using the backslash-command in MATLAB. (Page 89 or lecture notes). Check that you have used the command correctly by computing the residual $r = b - Ax$ (it should be close to machine precision).

b) Compute the lower and upper triangular matrices L and U in the LU-factorization, using the command `[L,U]=lu(A)`. Solve $Ax = b$ in two steps using the matrices L and U (page 82 or lecture notes). Check that you get (approximately) the same answer as in (i).

c) Learn how to use the commands `tic` and `toc` (write *help tic* in the command window). Check how long time (t_0) it takes to solve $Ax = b$ in (i). Thereafter, check how long it takes to do the lu-factorization (t_1) and to solve the system (t_2) in (ii). As a comparison, you can also check the time (t_3) it takes to solve the system using `inv(A)`. Use `tic` and `toc` described above.

d) Vary n, for example $n = 100, 200, 300, ..., 1000$ (if your computer is slow you can try for example $n = 10, 20, 30, ..., 100$ instead). Can the theoretical (asymptotic) complexities $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ be confirmed? (For example, you can plot the times t_0, t_1, t_2, t_3 as functions of n.)

   Hint: The computer is usually busy with many programs at the same time, all needing attention from the processor. This influences the computational times for your experiment in an unpredictable way. To reduce this effect, it is usually a good idea to do a few simulations for each n and present an average time.

4. *If you have time left:* solve problems 2.7.3a and 2.7.4a in the book, followed by computer problem 2.7.1a.