Aschaffenburg University of Applied Sciences
Faculty for Engineering
Prof. Dr. Michael Möckel

Email: Michael.Moeckel@th-ab.de
Telephone: +49 (0)6021 / 4206 - 507

Numerical Methods and Simulation
Module MT 03
Winter term 2022/23

https://moodle.th-ab.de/course/view.php?id=624

Problem Set 6

LU decomposition of matrices

Homework

Exercise 1: Using LU decomposition for solving equations

Using the LU decomposition of a matrix A allows for solving the system of linear equations $Ax = b$. As it does not modify the vector b it is very useful for solving systems which have the same coefficient matrices but different vectors b. The method is based on the equivalence

$$PA = LU$$

Hence, we can re-write the linear system

$$Ax = b \quad \Rightarrow \quad PAx = Pb \quad \Rightarrow \quad LUx = Pb \quad \Rightarrow \quad L(Ux) = Pb$$

Then one can solve the problem simply by forward and backward substitution: First, solve $Ly = Pb$, then $Ux = y$ as in exercise 1.

Exercise 2: LU decomposition of matrices

The LU decomposition is a method to split a given rectangular matrix A into the two lower and upper matrices L and U and a permutation matrix P. We only consider the case of square matrices here. It is very similar to the Gaussian algorithm and will be helpful to solve systems of linear equations efficiently.

(1) How it works. A lower diagonal matrix represents matrix manipulations. To see this, define an elementary lower diagonal matrix by the two commands `L = diag(ones(4,1))`. Then add another nonzero element by `L(3,2) = 8`. Define a simple 4×4 matrix `A = hadamard(4,4)` and calculate the (matrix) product $L * A$. What does an elementary lower diagonal matrix do? Look at the lines!

(2) So the lower triangular matrix provides a kind of bookkeeping of the matrix manipulations. Understand and run the following code to calculate the LU decomposition of the matrix M.

```
n = 4;
A = hadamard(4)
L = zeros(n,n);

for i = 1:n-1
    L(i+1:n,i) = A(i+1:n,i)/A(i,i)
    for j = i+1:n
        A(j, i+1:n) = A(j, i+1:n) - L(j,i) * A(i, i+1:n)
    end
end
```

(3) Note that the programme does not update all matrix elements. Why?

(4) Compare with the direct Matlab command for the lu decomposition `lu(M)`. Note that Matlab is memory efficient and saves the L matrix into the unused matrix elements of the upper triangular matrix! A separated output can be obtained with `[L U P] = lu(M)`. P is a special permutation matrix which allows to apply the same permutations to the constant vector b (see exercise 4).

(5) Replace the inner loop of the programme in (2) by a Matlab-style vectorization. Hint: Note how j appears in all variables of the inner loop and compare with the declaration of the loop.

Exercise 3: Solving an electric circuit with different voltages

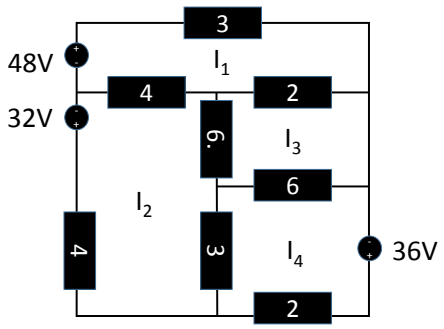
Systems of coupled linear equations arise in many technical fields. One example is calculating currents in an electrical circuit. Consider the following circuit of resistors (in Ohm) and voltage sources. The Kirchhoff laws connect currents and voltages in a electrical circuit. For each plaquette of the circuit they can be written as

$$\sum_j R_j (I_{j,+} - I_{j,-}) = \sum_j V_j$$

(1) Set up the Kirchhoff rules for the currents I_1 ... I_4 assuming that all currents run clockwise in each plaquette of the circuit.

(2) Solve the resulting system of coupled equations. Which method would you use?

(3) Now switch off the upper left voltage source and recalculate the results



Exercise 4: Some symbolic operations in MATLAB

Sometimes little calculations occur for which analytical results give an advantage over numerical numbers. Matlab is not primarily designed for symbolic calculations but some of them do work. We only consider the most practical ones.

(1) Pure symbolic. Symbolic calculations require the definition of symbolic variables using the keyword `syms` followed by the names of the symbolic variables, e.g. `syms x y`. Then, a symbolic function can be created by `f = symfun(sin(x),x)`. Note that this is a symbolic function! Typing `f(2)` will not produce $\sin(x = 2) = 0.9093$ but the symbolic substitution `sin(2)` (test it). If you want to evaluate a particular value, you have to do so explicitly `eval(f(2))`.

(2) Symbolic derivative. You can calculate the symbolic derivative from a (not too complicated) symbolic function using `diff`. For instance, `g = diff(f)` should produce the symbolic cosine function. It is symbolic, so you cannot evalute it directly: `g(3)` fails! Use instead `eval(subs(g,x,0))` and check the result. You can plot any symbolic function using `ezplot`.

(3) Symbolic Taylor expansion. There was a typo in the definition of the Taylor expansion command on Problem Set 3. For the symbolic function $f(x)$ the Taylor expansion around zero can be obtained by `taylor(f)`. More parameters can be used, e.g. an expansion around a point $x0$ (number) and to order n, e.g. in `taylor(f, x, 5, 'order', 3 )`. Note how the so-called *order* of the Matlab command is defined and compare with the usual order of a polynomial.

(4) Mixed numeric and symbolic calculations. Assume we had defined an anonymous function before by `f = @(x) (5 - x.*exp(0.5.*x))./1.2`. Calulate the symbolic derivative by (i) declaring x an symbolic variable after the function definition using `syms x` and (ii) creating a new symbolic function which contains the derivative by `df = symfun(diff(f,x))`. Simplify the expression using `simplify(df)`. Afterwards, you can translate it into the anonymous function dfa by `dfa=@(x) eval(df)` and plot it in the usual way.

(5) Symbolic matrices. You can enter a symbolic matrix using symbolic representations of variables and numbers. Create the symbolic matrix `S1 = [sin(sym(1)) sin(sym(2)) ; sin(sym(3)) sin(sym(4)) ]` and the vector `c = [1,2]`. A symbolic solution is created by the usual command `x = S1 \ c`. Print the result in a nicer way using `pretty(x)` and simplify the result symbolically using `simplify(x)`.

(6) Symbolic and numeric matrices. Enter the simple symbolical matrix `Cs = [ sym(1) sym(2) ; sym(2) sym(4)]` and create a numerical version by the cast `Cn = double(Cs)`. Define two vectors `d1 =[4;8]` and `d2 = [1;1]` and solve the systems $Cx = d_i$ for $i = 1, 2$ both symbolically and numerically. Check the determinant of the matrix C. Which output gives more information?

References:

[1] Amos Gilat, MATLAB, An Introduction with Applications, Wiley 2015