Numerical Algorithms - Homework Assignment 2

VU Numerical Algorithms, summer semester 2018. Due to 21.05.2018.

Paper-and-Pencil Exercises

1. *(2 points)* We have discussed elementary elimination matrices M_k in class. Prove the following two properties of elementary elimination matrices, which are very important for making LU factorization work efficiently in practice:

- M_k is nonsingular. Represent M_k^{-1} explicitly and show that $M_k M_k^{-1} = M_k^{-1} M_k = I$.
- The product of two elementary elimination matrices M_k and M_j with $k \leq j$ is essentially their "union"; and therefore they can be multiplied without any computational cost.

Answer:

In the following, we assume that M_k are lower triangular elimination matrices as used for e. g. the LU decomposition.

Part 1: Show that $M_k M_k^{-1} = M_k^{-1} M_k = I$.

(1) Since $M_k = I - v_k e_k^T$ and because of our assumption, we can conclude that $v_k e_k^T$ is a strictly lower triangular matrix.

(2) More specifically, due to the structure of M_k, $v_k e_k^T$ is a matrix that's equal to a zero matrix with variable values in one column k under the diagonale.

(3) We want to show that $M_k M_k^{-1} = M_k^{-1} M_k = I$. Hence:

$$\begin{aligned} M_k M_k^{-1} &= (I - v_k e_k^T)(I + v_k e_k^T) \\ &= I^2 - I * v_k e_k^T + I * v_k e_k^T - (v_k e_k^T)^2 \\ &= I - (v_k e_k^T)^2 \end{aligned}$$

Note that $I = I^2$ and that $I * v_k e_k^T = v_k e_k^T$.

(4) We know that $v_k e_k^T$ is a strictly lower triangular matrix with non-zero values in only one column (see statement 2.). That implies that multiplying a matrix in the form of $v_k e_k^T$ with any other matrix of this pattern leads to zero matrix, because at least one out of the two numbers involved in scalar multiplications in the dot product is zero and hence every vector dot product equals zero.
Thus:

$$\begin{aligned} M_k M_k^{-1} &= I - (v_k e_k^T)^2 \\ &= I \end{aligned}$$

(5) Since the multiplication operation in $(I - v_k e_k^T)(I + v_k e_k^T)$ is a commutative operation,

$$(I - v_k e_k^T)(I + v_k e_k^T) = (I + v_k e_k^T)(I - v_k e_k^T)$$

holds.

(6) Let's transfer step 5. to the left hand side of the equation:

$$M_k M_k^{-1} = M_k^{-1} M_k$$

(7) Since we know that $M_k M_k^{-1} = I$ and that $M_k M_k^{-1} = M_k^{-1} M_k$, we know that

$$M_k M_k^{-1} = M_k^{-1} M_k = I$$

and thus our statement is proven.

Part 2: Show that the product of two elementary elimination matrices M_k and M_j with $k \leq j$ is essentially their "union"; and therefore they can be multiplied without any computational cost.

Assumptions:

- $k \leq j$.
- Multiplication sequence is $M_k M_j$.

(1) Each elimination matrix M_n is a diagonal matrix having non-zero values only on it's diagonale (which are exclusively ones) and below the diagonale in column n. Let $M_n^{i,j}$ be the value of M_n in row i and column j. Then:

$$i < j \Rightarrow M_n^{i,j} = 0$$
$$i = j \Rightarrow M_n^{i,j} = 1$$

(2) Let r be a row index and c be a column index. Let s_k be the set of indices $(r > k, c = k)$ of sub-diagonale entries in M_k and s_j be the set of indices $(r > j, c = j)$ of sub-diagonale entries in M_j.

(3) Let r_k, c_k be the current row and column indices in M_k. Let r_j, c_j the current row and column indices in M_j. This leads to the following case distinction:

(3.1) $(r_k \leq k \vee c_k \neq k) \Rightarrow (r_k, c_k) \notin s_k$:

(3.1a.) $r_k = k \wedge c_k = k$: Is an entry on the diagonale: $M_j^{r_j, c_j}$ is added to the dot product.

(3.1b.) All other cases: Above the diagonale or below the diag. in a column $c \neq k$ - hence, the scalar and the corresponding term in the dot product is 0.

(3.2.) $(r_k > k, c_k = k) \Rightarrow (r_k, c_k) \in s_k$:

(3.2a.) $(r_j, c_j) \in s_j$: Not possible, since it would contradict step 3.

(3.2b.) $r_j = j \wedge c_j = j$: $M_j^{r_j,c_j}$ is on the diagonale and hence 1, therefore $M_k^{r_k,c_k}$ is added to dot product.

(3.2c.) $\neg(r_j = j \wedge c_j = j) \wedge (r_j, c_j) \notin s_j$: $M_j^{r_j,c_j}$ is 0, therefore $M_k^{r_k,c_k}$ is 0.

(4) Based on 3. and the definitions of a matrix multiplication and furthermore of the dot product, there is no dot product calculated during the computation of the matrix multiplication involving a multiplication $M_k^{a,b} M_j^{c,d}$ for which $(a, b) \in s_k \wedge (c, d) \in s_j$ holds.

(5) Due to the structure of M_k and M_j and the relationships described in step 4., each dot product calculating a value in the resulting matrix $M_k M_j$ includes *exactly* one non-zero term, i. e. $M_k M_j$ contains only values that appear in M_k or M_j.

(6) According to steps 3. and 4., every non-zero term consists of at least one 1 (i. e. $1 * 1$ or $1 * x$).

(7) Furthermore, each subdiagonal entry $\in s_k \vee \in s_j$ is part a non-zero term used for the dot product exactly once.

(8) If an element is a part of a non-zero term and $\in s_k \vee \in s_j$, it is (according to the definition of matrix multiplication and statements 1. to 7.) contained by

(9) Based on statements (1) to (8) we conclude that

$$M_k M_j = M_k \cup M_j$$

2. *(2 points)* We know that a matrix A is singular if $det(A) = 0$. Can we also conclude that the determinant of a matrix is a good indicator of near singularity? In other words, does the magnitude of a nonzero determinant give any information about how close to singular the matrix is?

Give a proof for your answer.

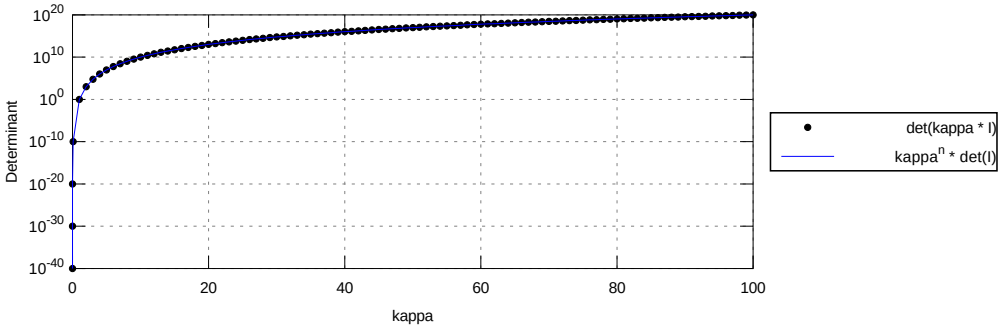
Hints: Experimental investigations may help for getting a first idea. It may also help to consider simple cases, such as multiples of the identity matrix.

Answer:

In order to prove that the determinant is *not* a reliable indicator of the proximity of a matrix to singularity (to this purpose, $cond(A)$ can be used), we argue that $det(\kappa A) = \kappa^n det(A)$ holds, where n is the dimension of the square matrix A and κ an arbitrary constant scalar.

Therefore, the determinant for any matrix can be arbitrarily increased or decreased independent from the content of this matrix. For illustration purposes see this chart of $det(\kappa I)$ with increasing κ and $n = 10$:

Fig. 1: det(I) with increasing kappa



The determinant is still the definitive measure to determine whether matrices are singular or not - but on machines with finite precision rounding errors can produce misleading results, as can be seen above.

We now show that $det(\kappa A) = \kappa^n det(A)$.

1. The determinant of a matrix is calculated based on the permutations of all $[1...n]$, meaning that we end up with a sequence of terms that are each n elements long. E.g.:
$$det \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} = a_1b_2c_3 - a_1b_3c_2 - a_2b_1c_3 + a_2b_3c_1 + a_3b_1c_2 - a_3b_2c_1$$
2. All these terms are of length n.
3. To multiply a matrix A with a constanst scalar κ, we multiply the scalar with all elements of A:
$$\kappa \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} \kappa a_1 & \kappa a_2 & \kappa a_3 \\ \kappa b_1 & \kappa b_2 & \kappa b_3 \\ \kappa c_1 & \kappa c_2 & \kappa c_3 \end{bmatrix}$$
4. Thus, every element of the matrix is increased by the factor κ. To continue the example from above:
$$\begin{aligned} det(\kappa A) &= \kappa a_1 \kappa b_2 \kappa c_3 - \kappa a_1 \kappa b_3 \kappa c_2 - \kappa a_2 \kappa b_1 \kappa c_3 + \kappa a_2 \kappa b_3 \kappa c_1 + a_3 \kappa b_1 \kappa c_2 - \kappa a_3 \kappa b_2 \kappa c_1 \\ &= \kappa^3 a_1 b_2 c_3 - \kappa^3 a_1 b_3 c_2 - \kappa^3 a_2 b_1 c_3 + \kappa^3 a_2 b_3 c_1 + \kappa^3 a_3 b_1 c_2 - \kappa^3 a_3 b_2 c_1 \\ &= \kappa^3 (a_1 b_2 c_3 - a_1 b_3 c_2 - a_2 b_1 c_3 + a_2 b_3 c_1 + a_3 b_1 c_2 - a_3 b_2 c_1) \\ &= \kappa^3 det(A) \end{aligned}$$
Speaking more generally: As illustrated in the last line in the example above, $det(\kappa A) = \kappa^n det(A)$ holds assuming $det(A)$ is always a sum of terms with n variables each, which is the case according to its definition.

This shows that $det(A)$ can be arbitrarily scaled independently from the conditioning of A. Thus, the magnitude of a nonzero determinant is not a good indicator for the closeness of a matrix to singularity.

Programming Exercises

PR I - Condition Estimation *(5 points)*

Implement the randomized estimator introduced in the lecture for the 1-norm condition number of square $n \times n$ matrices and compare its results for varying dimensions up to $n = 500$ (random matrices) with the Octave functions `cond` and `condest`.

For fixed $n \in 5, 10, 100$ investigate how the number of random vectors generated influences the quality of the approximation, by plotting the relative error of the estimated condition number.

Illustrate your results graphically. If your data is too noisy, average over a few iterations.
What are your conclusions? Is this a practical way for condition number estimation?

Answer:

One heuristic to approximate the condition number of a matrix without having to calculate its inverse is to use a randomized estimator. We implement a naive version of such an approach which works by repeatedly picking random vectors y with the objective of maximizing the ratio $\frac{||z||_1}{||y||_1}$ where $Ay = z$ with given A.
All random elements are generated using Octave's `rand(n)` function.

Part 1: Comparison of `cond(A)`, `condest(A)` and the manually implemented version.

We use $t = 1000$ different random vectors (generated using `rand(n, 1)`) in the manually written approximation.

See figure 2 for a plot of the behaviour of the condition number of a random matrix A (generated with `rand()`) with dimension $n \leq 500$. The chart also includes a comparison with the built-in `condest()` and the manual implementation of a randomized condition number estimation approach[1].
While `condest()` yields a very close approximation, the manual implementation is significantly off - usually by one order of magnitude. This might be sufficient for a crude lower bound of $cond(A)$ but leaves plenty of room for improvement. The manually implemented stochastic approximation shows the same behaviour as $cond(A)$ though.
Also, $cond(A)$ seems to rise exponentially with the n for small matrices until it saturates and reaches a plateau.

Part 2: Investigate how the number of picked vectors for y influences the accuracy of the resulting approximation for $cond(A)$.

See figure 3 for the corresponding plot[2]. We conclude that the benefit of increasing the number of picked random vectors y is correlated with the size of the matrix - the smaller n, the more a higher number of randomly selected vectors can contribute. With bigger matrices, visible already for $n = 100$, we can't significantly improve on the accuracy of the condition number approximation.

One explanation that applies to part 2 and, to a lesser extent, also to part 1: The approach implemented for this exercise generates random vectors of length n (the dimension of the matrix A). In order to achieve a close approximation, $\frac{||z||}{||y||}$ has to be as large as possible - which implies $||y||$ has to be as small as possible and at the same time z in $Az = y$ ought to be maximized. Since we rely on a purely randomized approach with the generated vectors for y being independent from each other, the probability of a random vector fulfilling both these conditions decreases with increasing n.

Conclusio: The approach implemented for this task is not practical for applications beyond toy datasets. Octave's `condest()` shows that randomized estimations of the condition number can work quite well, but this requires additional measures not included in the naive approach used here.

[1]: The result for the procedure calculating the manually implemented random approximation was averaged over five executions to reduce the noisiness in the data.
[2]: Note that we used a linear scale for this plot as it's more useful for the values displayed there.

Fig. 2: Comparison of condition estimates for random matrices

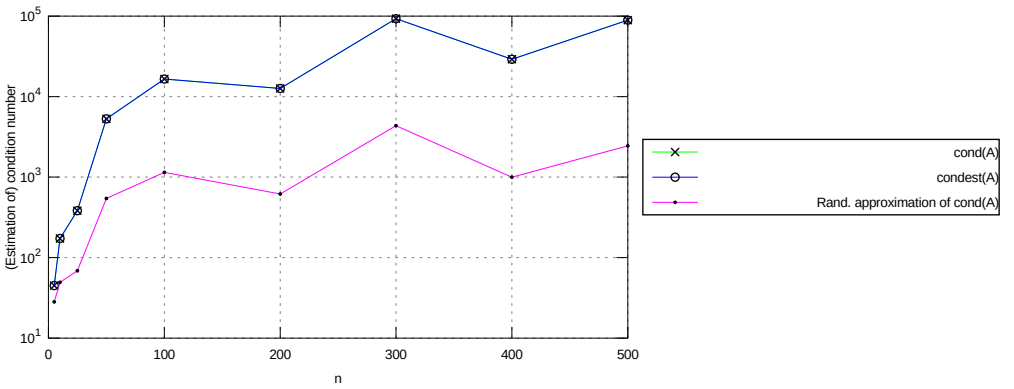
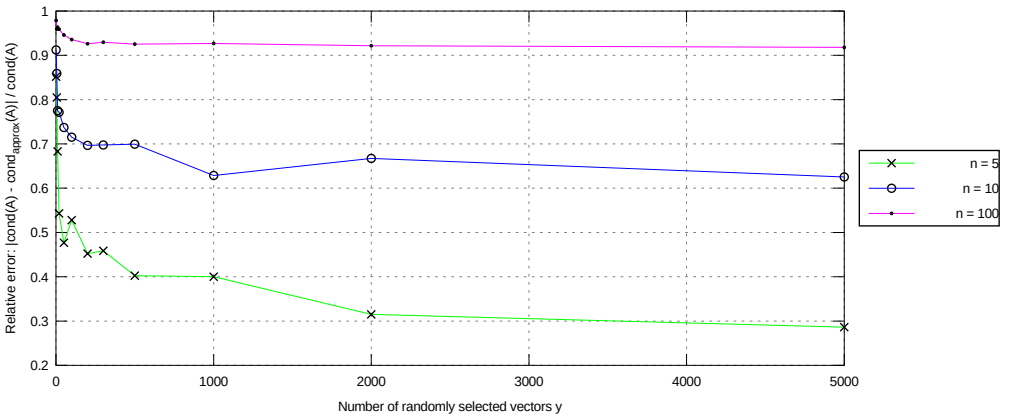


Fig. 3: Behaviour of random. approximation of cond with growing number of tries



PR II - Average Case Perturbation Errors (3 points)

We have seen in class that the relative error in the solution of a linear system due to perturbations E in the matrix A and Δb in the right hand side b can be bounded as

$$\frac{\|\Delta x\|}{\|x\|} \leq cond(A) \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|E\|}{\|A\|} \right)$$

This is a *worst-case bound*.
In this exercise your task is to empirically evaluate how tight this analytical bound is and how the relative error in practice relates to this bound. For this purpose, proceed as follows:

- Consider the 1-norm and $n = 100 : 50 : 1500$
- For each n, generate
  - a single random Δb with $||\Delta b||_1 = 10^{-8}$
  - a single random E with $||E||_1 = 10^{-8}$
  - many random A and b
- For each n, compute the averages of the left and right hand sides of the bound (1) over the randomly generated input data A and b.
- Plot the averages of the left and right hand sides of the bound (1) over n.

What are your conclusions?

Answer:

We want to compare the actual relative error $\frac{\|\Delta x\|}{\|x\|}$ in a linear system with the theoretically derived upper bound $cond(A) \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|E\|}{\|A\|} \right)$.

Note: The 1-norm is used consistently for all norms appearing in this problem.

For a system $Ax = b$ we define x as vector of ones[1] and let $b = Ax$ and $\hat{b} = b + \Delta b$. Thus: $\hat{x} = A \setminus \hat{b}$ and $\Delta x = abs(\hat{x} - x)$. We generate A, Δb and E randomly using `rand(n)` 10 times for each problem size $n \in \{100 : 50 : 1500\}$[2]. Both Δb and E are renormed so that $\|\Delta b\|_1 = \|E\|_1 = 10^{-8}$. Given that, we have all information necessary to compute both sides of the inequalition whose practical bounds we are to investigate.

Figure 4 shows that the actual relative error (on the left hand side) is, inside the sampled region, between roughly 3 and 5.5 orders of magnitude smaller than the upper bound represented by the right hand side of

$$\frac{\|\Delta x\|}{\|x\|} \leq cond(A) \left(\frac{\|\Delta b\|}{\|b\|} + \frac{\|E\|}{\|A\|} \right).$$

We conclude that, if our data is not an extreme case, the relative error we can expect is significantly lower than the one defined by the upper bound described on the right hand side of this inequation. Also, while the absolute number described by the upper bound does not match the actual relative error, the trend matches very well - the calculated upper bound over varying n largely follows the same pattern as the actual relative error.
This means that while we (depending on our requirements) can't reasonably rely on the upper bound to calculate an useful approximation of our relative error, it can still be very useful for (1) telling us the upper threshold for our relative error and (2) to estimate or interpolate how the relative errors of similarly generated matrices of different sizes relate to each other without knowing all of their actual relative errors.

[1]: As discussed in the lecture on 2018-05-17, we are not generating random b, but random A and assume x to be a vector of ones.
[2]: I. e. n is an element of all numbers between (including) 100 and 1500 divisible by 50.

Fig. 4: Bounds of the relative error
in a linear system in practice