University of Toronto at Scarborough
CSCC37—Numerical Algorithms for Computational Mathematics, Fall 2022

Assignment #2: Linear Systems / Gauss Elimination

Due: October 31, 2022 at 11:45 p.m.
This assignment is worth 10% of your final grade.

Warning: Your electronic submission on *MarkUs* affirms that this assignment is your own work and no one else's, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters, the Code of Student Conduct, and the guidelines for avoiding plagiarism in CSCC37.

This assignment is due by 11:45 p.m. October 31. If you haven't finished by then, you may hand in your assignment late with a penalty as specified in the course information sheet.

[15] 1. In lecture we showed how Gauss transforms could be used to compute the $A = LU$ or $PA = LU$ factorizations of $A \in \mathbb{R}^{n\times n}$. When doing examples we assumed certain properties of these transforms in order to simplify operations such as matrix inversion. In this question, you will attempt to prove these properties.

(a) Let $\mathcal{L}_k$ be the Gauss transform of order n used to annihilate elements in the k-th column during the k-th stage of Gaussian elimination. Show that $\mathcal{L}_k$ can be written compactly in matrix-vector notation as

$$\mathcal{L}_k = I - m_k e_k^T, \tag{1}$$

where e_k is the k-th column of the identity matrix and

$$e_i^T m_k = 0, \quad i = 1, 2, \ldots, k. \tag{2}$$

(b) Using the notation in (a), prove that $\mathcal{L}_k^{-1}$ can be computed simply by changing the sign of each multiplier (i.e. by changing the sign of each non-zero element in m_k).

(c) Using (a) and (b), prove that $(\mathcal{L}_k\mathcal{L}_j)^{-1}$, $j < k$, can be computed simply by adding the two transforms, changing the sign of each multiplier, and subtracting the identity.

(d) Let $\mathcal{P}_i \equiv \mathcal{P}_{ij} \in \mathbb{R}^{n\times n}$ be the permutation matrix which, when pre-multiplied to another matrix, swaps rows i and j of that matrix with $i < j$. Using the notation for $\mathcal{L}_k$ in (a), prove that $\tilde{\mathcal{L}}_k = \mathcal{P}_i\mathcal{L}_k\mathcal{P}_i$ is just $\mathcal{L}_k$ with multipliers i and j swapped, but only if $i > k$.

[10] 2. We know from linear algebra that the determinant of the product of two square matrices is equal to the product of the determinants; i.e., for $A, B \in \mathbb{R}^{n\times n}$, $\det(AB) = \det(A)\det(B)$. Using this fact, derive an *optimally efficient* formula for computing the determinant of $A \in \mathbb{R}^{n\times n}$ based on the $PA = LU$ factorization of A.

[15] 3. Consider the linear system $Ax = b$ where

$$A = \begin{bmatrix} 3 & 3 & 9 & 6 \\ 4 & 4 & 4 & 4 \\ 1 & 1 & 5 & 5 \\ 2 & 2 & 4 & 6 \end{bmatrix}, \quad b = \begin{bmatrix} 21 \\ 24 \\ 10 \\ 16 \end{bmatrix}. \tag{3}$$

(a) Compute the $PA = LU$ factorization of A. Use exact arithmetic. Show all intermediate calculations, including Gauss transforms and permutation matrices.

(b) Use the factorization obtained in (a) to compute the determinant of A. What does this determinant tell you about the solution to (3)?

(c) Use the factorization obtained in (a) to solve (3), if a solution exists.

(d) Why is Gaussian Elimination usually implemented as in this question (i.e., $PA = LU$ is computed separately, and then the factorization is used to solve $Ax = b$)?

[25] 4. Consider the $n \times n$ linear system $Ax = b$, where the elements of A and b are given by:

$$a_{i,j} = j^i \quad 1 \le i, j \le n$$

$$b_i = \sum_{j=1}^{n} (-1)^{j+1} a_{i,j} \quad 1 \le i \le n$$

The exact solution to this system is $x = (1, -1, \ldots, (-1)^{n+1})^T$. This is an instance where cond(A) increases very rapidly with increasing n; i.e. A is more nearly singular as n increases.

Write a MATLAB function that generates and solves this system using Gaussian elimination with partial pivoting and iterative improvement. Your function header is

```
function [rcnd,x0,re0,rr0,xf,ref] = badsys(n)
```

where

- `rcnd` is the reciprocal condition estimate,
- `x0` is the initial approximate solution,
- `re0` is the relative error in the initial solution,
- `rr0` is the initial relative residual,
- `xf` is the final approximate solution after iterative improvement,
- `ref` is the relative error in the final solution.

Use the 2-norm when computing errors and residuals. For example, compute the relative error in the initial solution as $\|x - x_0\|_2 / \|x\|_2$, and the initial relative residual as $\|b - Ax_0\|_2 / \|b\|_2$. (Since the true solution x is known for this problem, we can compute the relative error exactly. This, of course, is not possible in general.) Use MATLAB's `rcond()` to estimate the reciprocal condition of A.

Use the iterative improvement algorithm described in lecture. Note that this algorithm involves solving several systems with the same coefficient matrix but different right-hand sides. In order to get full marks for this question, you must implement this *efficiently*; i.e. factor the matrix *once only*, and then

use the $PA = LU$ factorization to do forward and backward solves for each right-hand side. You may use MATLAB's `LU()` to factor the matrix; you do not need to implement Gaussian Elimination!

An interesting question arises when doing this problem: Just how many iterations of iterative improvement is enough? There are several issues to consider here. For instance, if the initial approximate solution is quite good, the initial residual $b - Ax_0$ may suffer from subtractive cancellation. In this case, iterative improvement is essentially useless. Residuals are often computed in extended precision to avoid this cancellation. You won't be using extended precision in this problem, but you should check for subtractive cancellation. Your program should not continue with the improvement if this happens. Another problem occurs when the coefficient matrix A is so badly conditioned that *no* significant digits are correct in the approximate solution. In this case, no significant digits will be correct in the update either, so again iterative improvement is useless. This usually can be detected by monitoring the size of $\|x_i - x_{i-1}\|_2$. If the improvement is working, this should be getting smaller with each iteration. At the very least, you should set a limit for the maximum number of iterations. Assuming the number of correct digits doubles each time, it's probably not worth doing more than 4 or 5 iterations considering the mantissa length of double precision floats on your machine.

The execution of your program should generate a table that summarizes the output of your function. For each `n`, list `rcnd, re0, rr0, ref` in one row of the table. Try solving systems of order $n = 4, 5, 6, \ldots$ until the condition of A becomes so bad that iterative improvement no longer works. Verify that the relative residual always remains small, even though the approximate solution before doing iterative improvement becomes more and more inaccurate. Use some of the theory discussed in lecture to support your observations.

[total: 65 marks]