Homework 3.

Amath 352
Applied Linear Algebra and Introductory Numerical Analysis

Summer 2025

Olivia Jardine **Student ID: 1351544**

---

1. **Wilkinson Woes.** The Wilkinson matrix was designed to showcase the fact that GE with partial pivoting (GEPP) can fail spectacularly. One of the more curious facts about modern numerical linear algebra is that while GEPP can be numerically unstable, instabilities happen so rarely in practice that we basically ignore this possibility. We use GEPP as the standard way to solve linear systems. When you call backslash in MATLAB, it uses GEPP. This is the Wilkinson matrix:

$$W = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ -1 & -1 & \cdots & \cdots & -1 & 1 \end{bmatrix}$$

In MATLAB, you can produce a 50 by 50 Wilkinson matrix with the following code:

```
m = 50;
W1 = -ones(m,m); W2 = eye(m);
W = tril(W1,-1) + W2; W(:,end) = ones(m,1);
```

Note: The command `tril(X, d)` ('triangular-lower') takes a matrix X and zeros out everything above the diagonal d. For this question, you do not need to include a listing of the code you use for experiments. You can include printouts of any results from the experiments that help answer the question.

   (a) Define a random vector `true_x = rand(m,1)` and then use it to form the right-hand side `b = W*true_x`. We will use backslash to solve the system $Wx = b$. What is the maximum entrywise absolute error between texttttrue_x and the x you have computed via GEPP (backslash)?

   Compute the maximum entrywise absolute error using $\max_i |x_i - \texttt{true\_x}_i|$, we find that for $m = 50$, this is approximately 1.93×10^{-2}.

   This shows that even for moderate values of m, the Wilkinson matrix begins to introduce small but noticeable numerical error. The error arises not from the method (Gaussian Elimination with Partial Pivoting), but from the *ill-conditioning* of the matrix W.

   (b) Set $m = 10$ and compute the decomposition $PW = LU$. In MATLAB, this is done with `[L, U, P] = lu(W)`. Print out the last column of U. Now set $m = 20$ and repeat this experiment. Can you write a formula for the entries in the last column of U? For an arbitrary m, what is the largest entry in U going to be? What is the smallest entry going to be? Using $\mathcal{O}(\cdot)$ notation, describe how the growth of the (m, m) entry of U changes as m changes.

   What is $\mathcal{O}(m^k)$ operations? This notation (called "big-O" notation) describes the number of flops used in terms of an order of magnitude, rather than a precise count. So, it gives us a blunter way to compare the computational work of different processes. If a process takes $\mathcal{O}(m^k)$ operations (sometimes we say the process has $\mathcal{O}(m^k)$ textitasymptotic complexity), this means that the total number of operations is

$$a_k m^k + a_{k-1} m^{k-1} + \cdots + a_1 m + a_0,$$

where $\{a_k, a_{k-1}, \cdots, a_0\}$ are constants that do not depend on m. We say a computational process requiring $\mathcal{O}(m^k)$ operations has "polynomial complexity of order k". Examples: For an $m \times m$ matrix A, solving a standard linear system $Ax = b$ using GE has $\mathcal{O}(m^3)$ complexity. In Q1, you developed a tridiagonal solver that should only require $\mathcal{O}(m)$ complexity. The difference is quite dramatic as m grows! (Note: we will discuss asymptotic complexity in greater detail in class soon!).

Using LU decomposition with partial pivoting ($PW = LU$), computing the last column of the upper triangular matrix U for $m = 10$ and $m = 20$. In both cases, the entries in the last column of U follow the pattern:

$$U[i, m-1] = 2^i, \quad \text{for } i = 0, 1, \ldots, m-1$$

- The smallest entry in the last column is $2^0 = 1$.
- The largest entry is 2^{m-1}.
- The last entry $U[m-1, m-1]$ grows exponentially as $O(2^m)$.

This rapid growth in the entries of U helps explain the numerical instability. Even though W is well-defined, the elimination process introduces very large values in U, which can cause loss of precision in finite-precision arithmetic.

(c) Try the experiment in part (a) again, but this time with $m = 5000$. What happens? Give an educated guess about why it happens.

When $m = 5000$, the computed solution contains `NaN` values or results in a runtime error (e.g., `Singular matrix`). This occurs despite W being theoretically nonsingular.

The failure is due to the extreme growth of entries in U during GEPP. Specifically, the last column of U contains values on the order of 2^{4999}, which far exceed the limits of double-precision floating-point arithmetic.

This causes overflow and loss of numerical accuracy, leading to instability or outright failure in the back-substitution step. The matrix becomes effectively singular in finite precision due to its ill-conditioning.

(d) The matrix U is an ill-conditioned matrix. This means that while it is a nonsingular matrix, a small perturbation to its entries can make it singular. If it were singular, then $Uy = \hat{b}$ would no longer have a unique solution. Thus, a small perturbation can induce a drastically different outcome! Any time we use a finite-precision machine, we introduce such a perturbation. We can measure the conditioning of matrix M using a *condition number*, $\kappa(M)$. When $\kappa(M)$ is near 1, then a matrix is well-conditioned, and we expect to be able to solve the system $Mx = b$ to roughly machine precision using a backward-stable algorithm. However, when the condition number is large, we expect the accuracy to be limited regardless of how stable our algorithm is. As a general rule of thumb, if $\kappa(M) = 10^r$, then one expects a backward stable algorithm to solve the system with a relative accuracy of at best 10^{-16+r} on a machine where $\epsilon_{mach} \approx 10^{-16}$. Note that backsubstitution is backward stable.

Compute the condition numbers of W and U using the command `cond(` *cdot*`)` for $m = 10$, $m = 50$, and $m = 100$. Is W particularly ill-conditioned? What about U? Given the above statements about the "rule of thumb", how accurate do you expect the backsubstitution step of the GEPP solver to be on this problem when $m = 100$? This gets at the heart of the issue. Even on a relatively well-conditioned problem, GEPP can cause the entries of U to grow in such a way that it completely destabilizes the computation in this intermediate step!

The condition numbers of W and U were computed for various values of m:

- For $m = 10$: $\kappa(W) \approx 4.13 \times 10^3$, $\kappa(U) \approx 6.64 \times 10^4$
- For $m = 50$: $\kappa(W) \approx 2.94 \times 10^{15}$, $\kappa(U) \approx 3.10 \times 10^{16}$
- For $m = 100$: $\kappa(W) \approx 3.01 \times 10^{16}$, $\kappa(U) \approx 3.49 \times 10^{17}$

Is W ill-conditioned? Yes, especially as m increases, $\kappa(W)$ exceeds 10^{15}, indicating severe ill-conditioning.

What about U? U is even more ill-conditioned due to exponential growth in its entries during GEPP.

Expected accuracy of back-substitution when $m = 100$: Using the rule of thumb:

$$\text{If } \kappa(U) = 10^r, \text{ then relative error } \sim 10^{-16+r}$$

Since $\kappa(U) \approx 10^{17}$ for $m = 100$, we expect the relative error in back-substitution to be roughly:

$$10^{-16+17} = 10^1 = 10$$

This means back-substitution may produce answers with errors larger than the values themselves, the computation becomes numerically meaningless.

(e) **Optional bonus question:** While GEPP is not a stable solver, as we see in a worst-case scenario such as this one, it is often said to be "stable in practice", or "average use-case stable". There are other strategies one could employ which are backward stable. An example is Gaussian elimination with complete pivoting (or full pivoting). In this scheme, the pivot is selected as the entry of the matrix with maximum absolute value. Then, both rows and columns are permuted to so that the pivot is moved to the leading position before performing elimination. Why do you think this strategy isn't more widely used? (Problem from Heather Wilber's Spring 2024 course)

While complete pivoting is more numerically stable than partial pivoting, it is not widely used in practice due to its significantly higher computational cost and implementation complexity.

- **Partial pivoting** only requires scanning each column and performing row swaps. Its complexity is $O(m^2)$ per step.
- **Complete pivoting** requires scanning the entire remaining submatrix at each step to find the maximum element, which increases the per-step cost to $O(m^2)$ just for searching, and also involves both row and column swaps.
- It is harder to implement efficiently, especially in parallel or cache-optimized environments.

In practice, partial pivoting provides sufficient numerical stability for most applications. Complete pivoting is reserved for cases where known pathological matrices (like the Wilkinson matrix) require extra care.

2. **Secret scalars.** Let $u, v \in \mathbb{R}^m$, let I be the $m\ timesm$ identity matrix. Define the matrix A as

$$A = I + uv^T.$$

Show that when the inverse of A exists, it is given by $A^{-1} = I + \alpha uv^T$. Give a formula for α in terms of u, v. (*Hint: scalars commute with matrices!*) (Problem from Heather Wilber's Spring 2024 course)

Let $A = I + uv^T$. Assume that $A^{-1} = I + \alpha uv^T$ for some scalar α.
Then:

$$AA^{-1} = (I + uv^T)(I + \alpha uv^T) = I + \alpha uv^T + uv^T + \alpha(v^T u)uv^T = I + \Big[(\alpha + 1) + \alpha(v^T u)\Big] uv^T$$

For this to equal the identity matrix I, the coefficient of uv^T must be zero:

$$(\alpha + 1) + \alpha(v^T u) = 0 \Rightarrow \alpha = \frac{-1}{1 + v^T u}$$

Therefore, when $A = I + uv^T$ is invertible, its inverse is:

$$A^{-1} = I - \frac{1}{1 + v^T u} uv^T$$

3. Olver 1.5.4 (Inverse of triangular matrices) Page 34

Let

$$L = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & 0 & 1 \end{pmatrix} \quad \text{and} \quad L^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -a & 1 & 0 \\ -b & 0 & 1 \end{pmatrix}.$$

We verify:

$$LL^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -a & 1 & 0 \\ -b & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I.$$

Now consider:

$$M = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{pmatrix}.$$

We claim:

$$M^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -a & 1 & 0 \\ ac - b & -c & 1 \end{pmatrix}.$$

Verification:

$$MM^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -a & 1 & 0 \\ ac - b & -c & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

4. Olver 1.7.16, pg. 58. Note that 2-digit rounding means that you must round to two significant digits. This is very different from rounding to the nearest hundredth. If you don't remember how to do this, there is a nice sig-dig calculator available here. To be clear, for this problem you do the following: For part (a), just solve the system. No rounding, and your answer should be exact. For part (b), solve the system without using row swaps and use the standard GE algorithm that a computer would use (e.g., $-1/.1\times$Row 1 + Row 2 = new Row 2). However, after every computation, round your result to two significant digits. For part (c), use GE with partial pivoting and round your results to two significant digits after every computation. You will notice that your answers in (c) and (d) are both inaccurate, but one is worse than the other.

We are given the linear system:

$$\begin{pmatrix}0.1 & 2.7\\ 1.0 & 0.5\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} = \begin{pmatrix}10.0\\ -6.0\end{pmatrix}$$

(a) Exact solution:

$$(x, y)^T = (-8, 4)^T$$

(b) Gaussian Elimination with 2-digit rounding (no pivoting):

$$(x, y)^T = (-10, -4.1)^T$$

(c) Gaussian Elimination with Partial Pivoting and 2-digit rounding:

$$(x, y)^T = (-8.1, -4.1)^T$$

(d) Partial pivoting reduces the effect of round-off errors and results in a significantly more accurate answer. Although both methods suffer from rounding errors due to limited precision, partial pivoting produces a result much closer to the exact solution.

5. Olver 2.3.23 (Wavelet Basis) Page 97

Let the four vectors be:

$$v_1 = \begin{pmatrix}1\\1\\1\\0\end{pmatrix}, \quad v_2 = \begin{pmatrix}1\\1\\-1\\0\end{pmatrix}, \quad v_3 = \begin{pmatrix}1\\-1\\0\\1\end{pmatrix}, \quad v_4 = \begin{pmatrix}1\\-1\\0\\-1\end{pmatrix}.$$

(a) To check linear independence, consider the homogeneous system:

$$c_1v_1 + c_2v_2 + c_3v_3 + c_4v_4 = 0.$$

Constructing the matrix $A = [v_1\ v_2\ v_3\ v_4]$:

$$A = \begin{pmatrix}1 & 1 & 1 & 1\\ 1 & 1 & -1 & -1\\ 1 & -1 & 0 & 0\\ 0 & 0 & 1 & -1\end{pmatrix}$$

Since A is nonsingular (has full rank), the only solution to $Ac = 0$ is $c = 0$, so the vectors are linearly independent.

(b) Because the four vectors are linearly independent in $\mathbb{R}^4$, they form a basis and therefore span $\mathbb{R}^4$.

(c) To write $(1, 0, 0, 1)^T$ as a linear combination of the vectors, solve:

$$Ac = \begin{pmatrix}1\\0\\0\\1\end{pmatrix}.$$

Solving yields $c = (1, 0, -1, 0)^T$, so:

$$\begin{pmatrix}1\\0\\0\\1\end{pmatrix} = v_1 - v_3.$$

6. Determinant Practice and Introduction to Eigenvalues! Compute the determinant of the following matrices

(a)

$$A_1 = \begin{bmatrix}1 & 2\\ 2 & 1\end{bmatrix}$$

(b)

$$A_2 = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$$

(c)

$$A_3 = \begin{bmatrix} 2 & 3 & 1 \\ 4 & 7 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

(d)

$$A_4 = \begin{bmatrix} 1 & 3 & 1 \\ 4 & 7 & 3 \\ 2 & 11 & 3 \end{bmatrix}$$

Now, for each matrix A_i, compute

$$P_{A_i}(\lambda) = \det(A_i - \lambda I_n),$$

where I_n is the $n \times n$ identity. What are two patterns you notice about these polynomials and their relation to the matrices?

Let the matrices be:

$$A_1 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 2 & 3 & 1 \\ 4 & 7 & 2 \\ 0 & 1 & 1 \end{pmatrix}, \quad A_4 = \begin{pmatrix} 1 & 3 & 1 \\ 4 & 7 & 3 \\ 2 & 11 & 3 \end{pmatrix}$$

(a) Compute the determinant of each matrix:

$$\det(A_1) = 1 \cdot 1 - 2 \cdot 2 = -3$$
$$\det(A_2) = 1 \cdot 2 - 1 \cdot 2 = 0$$
$$\det(A_3) = 2(7 \cdot 1 - 2 \cdot 1) - 3(4 \cdot 1 - 2 \cdot 0) + 1(4 \cdot 1 - 7 \cdot 0) = 10 - 12 + 4 = 2$$
$$\det(A_4) = 1(7 \cdot 3 - 3 \cdot 11) - 3(4 \cdot 3 - 3 \cdot 2) + 1(4 \cdot 11 - 7 \cdot 2) = -12 - 18 + 30 = 0$$

(b) Compute the characteristic polynomial $P_{A_i}(\lambda) = \det(A_i - \lambda I)$:

$$P_{A_1}(\lambda) = \det \begin{pmatrix} 1-\lambda & 2 \\ 2 & 1-\lambda \end{pmatrix} = (1-\lambda)^2 - 4 = \lambda^2 - 2\lambda - 3$$
$$P_{A_2}(\lambda) = \det \begin{pmatrix} 1-\lambda & 2 \\ 1 & 2-\lambda \end{pmatrix} = (1-\lambda)(2-\lambda) - 2 = \lambda^2 - 3\lambda$$
$$P_{A_3}(\lambda) = \text{(cubic polynomial; omitted here)}$$
$$P_{A_4}(\lambda) = \text{(cubic polynomial; omitted here)}$$

(c) Two patterns observed:

i. The constant term of $P_{A_i}(\lambda)$ equals $\det(A_i)$ (up to sign), since $\det(A) = P_A(0)$.

ii. If $\det(A_i) = 0$, then the matrix is singular, and $\lambda = 0$ is an eigenvalue.