ASEN 3502 Computational Methods
Lab 1

Exact Due Date on Gradescope

General Instructions

1. In your code for solving linear systems, any single access may only touch a chunk of at most n entries of an $n \times n$ matrix, e.g. a row, a column, or a $n/m \times m$ block. For example, you cannot call `A\b`.
2. You may use any programming language, and you may use different languages for different parts. However, the primary goal of this lab is understanding, and *you should use Matlab unless every team member is enthusiastic about using something else.*
3. Adhere to the AI policy at asen3502.com/ai-policy.
4. During each lab session, each student in the group should spend some time as the "driver": the person typing things into the computer or the person typing equations in latex.

In this lab, you will begin by analyzing the truss below as a component of the Saturn V launch umbilical tower (LUT).

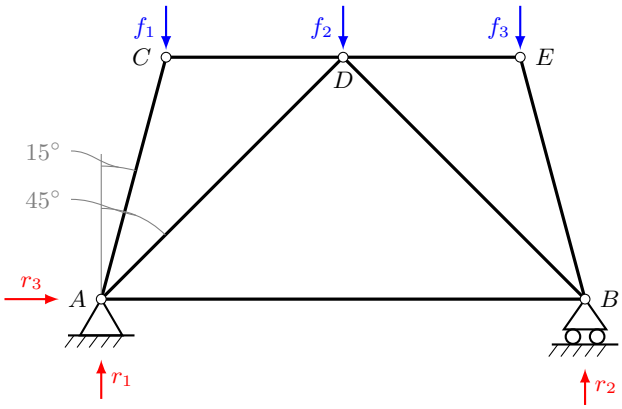

Figure 1: The truss section to be analyzed, and the corresponding part of the launch umbilical tower circled on the Apollo 17 stack (Photo: NASA).

Compiled 2026-08-28 08:11:23

Deliverables (at end of Week 3)

1. To the **Lab 1 Writeup** assignment on Gradescope: A **single pdf** writeup that contains all of the items marked writeup.
   - Make sure to clearly label all plots and figures and use correct units where appropriate.
   - The lab writeup does *not* need to include narrative text unless specifically requrested; it can just include the plot or results from each writeup task labeled with which task they are from.
2. To the **Lab 1 Code** assignment on Gradescope: Your **results.json** file and **source code** for all code items.
   - If you are working in Matlab, there will be one file for each task. In other programming languages, you may combine multiple parts into one file. However, please help the TAs by clearly labelling where everything is, e.g. use filenames like `week_1.jl`.
   - In this lab, you will *not* be graded based on documentation or comments in your code. (Though in practice, documentation is extremely important!)

1 Week 1: Gauss Elimination

1.1 Define the mathematical model

writeup

Use your knowledge from statics to write out a set of linear equations that will allow you to solve for the magnitude of tension or compression in each of the members in the truss section, along with the reaction forces, r_1, r_2, and r_3.

1.2 Implement Gaussian elimination

code

Write a function called `gauss` that takes arguments `A` and `b` solves $[A]\{x\} = \{b\}$ for $\{x\}$ using Gaussian elimination *without* pivoting and returns $\{x\}$. Use the `verify_gauss` script to verify that this has produced the correct answers.

1.3 Implement Gaussian elimination with partial pivoting

code

Create another function called `gauss_pivot` that takes arguments `A` and `b` solves $[A]\{x\} = \{b\}$ for $\{x\}$ using Gaussian elimination *with* partial pivoting and returns $\{x\}$. Use the `verify_gauss` script to verify that this has produced the correct answers.

1.4 Create a verification script

code

Create a script called `verify_gauss`. This should test the `gauss` and `gauss_pivot` functions on two cases:

1. Randomly generate a 4x4 matrix $[A]$ and vector $\{b\}$. Solve $[A]\{x\} = \{b\}$ for $\{x\}$ and then check that $\|[A]\{x\} - \{b\}\| \leq \epsilon$ for some small ϵ.
2. Create a matrix and vector pair that `gauss` will fail to solve, but `gauss_pivot` will succeed. For the `gauss` function, the verification script should test that the result contains `NaN` or `Inf`; for `gauss_pivot`, it should check that the result satisfies the original equations.

If one of the tests fails, use `error('Error message')` to indicate that it failed.

1.5 Solve for a load case

writeup code

Suppose the three loads are $f_1 = 5000kN$, $f_2 = 1200kN$ and $f_3 = 100kN$. Convert the equations from Task 1.1 to the form $[A]\{x\} = \{b\}$, and write a script that uses one of your Gaussian elimination functions to solve it. Make sure to include symbolic representation of the $[A]$, $\{x\}$, and $\{b\}$ that you used in your writeup. List the axial force in each of the truss members and indicate which are in tension and which are in compression.

1.6 Solve some big matrices

code

This is a puzzle and friendly competition to see who can solve the largest system of equations in 30 seconds. Your score is n, the size of the largest system you solve within the limit, and it goes on a leaderboard.

If you are using Matlab, you can just plug your `gauss` function in to get started:

```
evaluate(100, 'your.gradescope.email@colorado.edu', @gauss)
```

Start with a small n and work upward. When a run succeeds, `evaluate` writes `results.json`; that will be submitted at the end of the third week.

If you attain a score of 1500 or greater, you will receive full credit. If you can attain a score of 30,000 or greater and explain your code to the professor (or a TA if the professor is not there), you can leave early.

`evaluate` can also hand the system off through files, so you may solve it in any language you like. `evaluate-docs.md` documents both routes in full.

Things to try:

- `evaluate-docs.md` contains options to control sparsity patterns in the matrix - you can use these and exploit structure to get a much higher score.
- Use other algorithms in the book.
- Use multiple processing cores of your computer.

The general instructions for the lab apply to this section: You must write the code yourself, the AI policy applies, and you can only access or write to size n groups of matrix entries at once. The winner will explain their solution to the class, so make sure to understand it!

2 Week 2: LU Decomposition

To start, split your team into two subgroups. One subgroup should start on Task 2.1 and the other on Task 2.2.

2.1 Write down the LU decomposition algorithm

writeup

Subgroup 1: Create a representation of the LU decomposition algorithm using pseudocode or clearly written out steps. The algorithm should take a matrix $[A]$ as input and output two matrices, $[L]$, and $[U]$.

2.2 Create an LU verification script

code

Subgroup 2: you will create your verification script *before* you write your functions. This is sometimes known as "test-driven development". Create a verification script for `lu_decompose` and `lu_solve` functions (see Tasks 2.3 and 2.4).

1. For `lu_decompose`, the script should generate one or more `A`,`b` pairs and verify that `L*U` is close to `A`. Note: since `lu_decompose` does not pivot, the test matrix must have nonzero pivots. You can ensure this with `A = rand(n) + n*eye(n)`.

2. For `lu_solve`, check that `A*x` is close to `b` after solving with `lu_solve`.

2.3 Implement LU decomposition

code

Subgroup 2: Implement LU decomposition based off the pseudocode that Subgroup 1 created. If there are any ambiguities, ask Subgroup 1 to revise their pseudocode. `lu_decompose` should take in a matrix `A` as an argument and return a lower triangular matrix `L` and upper triangular matrix `U` that satisfy $[L][U] = [A]$. Test this with your LU verification script.

2.4 Implement LU solve

code

Implement a function `lu_solve` that takes lower triangular matrix `L`, upper triangular matrix `U`, and vector `b` as inputs, and solves for `x` so that $[L][U]\{x\} = \{b\}$. Test this with your LU verification script.

2.5 Find the worst loading case

code writeup

Suppose that an additional load of 8000 kN in addition to the loads from Task 1.5 is going to be added to the structure, but you do not know the split, that is, the new loads are

$$f_1' = f_1 + \Delta f_1 \qquad f_2' = f_2 + \Delta f_2 \qquad f_3' = f_3 + \Delta f_3$$

$$\Delta f_1 + \Delta f_2 + \Delta f_3 = 8000\text{kN}$$

Create a heatmap of the *maximum compressive axial force* in any member, where the x axis is Δf_1, the y axis is Δf_2. Starter code for this task is in `compression_heatmap`. In your code, you should only call `lu_decompose` once, then re-use `L` and `U` with `lu_solve` to calculate the axial forces as needed.

Note: Since you implemented LU decomposition *without* pivoting, you may need to rearrange the equations to avoid a divide-by-zero.

2.6 Solve some big matrices

code

Continue with Task 1.6.

3 Week 3: Matrix Inverse and Roundoff Error

3.1 Write down LU-based matrix inverse algorithm

writeup

Create a pseudocode (or other clear) representation of an algorithm that will compute the matrix inverse by first invoking the LU decomposition subroutine that you created in Task 2.1.

3.2 Implement the LU-based matrix inverse

code

Implement the algorithm from Task 3.1 as a function `lu_inv` that takes nonsingular square matrix `A` as an input and returns its inverse.

3.3 Verify the LU-based inverse

code

Create a verification script for the LU-based inverse. Determine what tests should go into the file to give you confidence that `lu_inv` is implemented correctly.

3.4 Calculate condition numbers

code

Create a function `lu_cond` that calculates the condition number, $\text{Cond}[A] = \|[A]\| \, \|[A]^{-1}\|$, of a matrix `A` using `lu_inv`. Use the spectral (2) norm. You may use Matlab's `norm` function within `lu_cond`.

3.5 An ill-conditioned stiffness matrix

code writeup

Consider the system of springs in Figure 2. The middle spring has a high stiffness, K, and the outer springs have low stiffness, k. This is similar to calculations that occur inside of finite element analysis (FEA) codes

that incorporate multiple materials with different elastic moduli. If a force F is applied in the rightward direction at mass 1, the displacements of the masses, $\{u\}$, behave according to

$$\underbrace{\begin{bmatrix} k+K & -K \\ -K & K+k \end{bmatrix}}_{\text{stiffness matrix}} \underbrace{\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix}}_{\text{displacement}} = \underbrace{\begin{Bmatrix} F \\ 0 \end{Bmatrix}}_{\text{force}}. \tag{1}$$

As K becomes much larger than k, the stiffness matrix approaches

$$\begin{bmatrix} K & -K \\ -K & K \end{bmatrix} \tag{2}$$

which has determinant $K^2 - K^2 = 0$, and is therefore singular. In practice, the matrix becomes ill-conditioned as K increases. With $k = 0.01$, calculate and plot the condition number for K ranging from 10^{-2} to 10^{14}, using logarithmic scales on both axes. Starter code for this task is given in `condition_plot.m`.

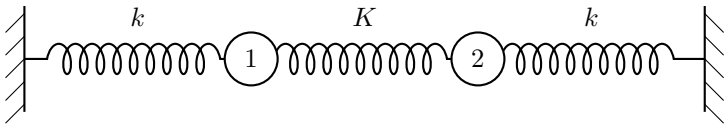

Figure 2: Two masses connected to each other and to rigid walls by three springs.

3.6 Observe round-off error

code writeup

Suppose that the middle spring represents a solid elastic link, and we want to calculate the tension T in it. As K gets very large, there are two imperfect ways to estimate this:

1. Assume that the link is perfectly rigid. In this case, $u_1 = u_2$, and the tension is $T_\infty = -F/2$. This is the asymptotically correct solution as $K \to \infty$.
2. Use Equation 1 to solve for u_1 and u_2, and calculate $T_{\text{mat}} = K\,(u_2 - u_1)$. This is a more accurate model, but involves inverting an ill-conditioned matrix as $K \to \infty$.

For K ranging from 1 to 10^{14}, with $F = 1$ and $k = 0.01$: Use your `lu_inv` function to calculate u_1 and u_2, and find $T_{\text{mat}} = K\,(u_2 - u_1)$ with both single and double precision (note that at some point you will start to get `NaN` results for the single precision system). Build the single-precision system with `single(A)` and `single(b)`, and do not convert back with `double`.

Plot the relative distance between the calculated tension and the asymptotic value, $|T_{\text{mat}} - T_\infty|/|T_\infty|$ against K on a log-log plot. On the same axes, plot the rough error bound that the condition number suggests, $\text{Cond}[A]\,\epsilon$, once with $\epsilon =$ `eps` $\approx 2.2 \times 10^{-16}$ for double precision and once with $\epsilon =$ `eps('single')` $\approx 1.2 \times 10^{-7}$ for single. Then **write a paragraph** explaining the shape of each curve. Why does it approach or move away from zero in different parts? Starter code for this task is given in `roundoff_plot.m`.

3.7 Solve some big matrices

Continue with Task 1.6.