Lab: Solving Linear Systems

Marcus Mellor

Oklahoma State University

Nov 07, 2025

Outline

Cholesky Decomposition

Lab Assignment

LU Factorization in Python

To perform LU factorization with partial pivoting in Python, we can use the `scipy.linalg.lu` function. This returns $[L]$, $[U]$, and $[P]$ given an $[A]$ input.

Python

```python
import numpy as np
import scipy
# Example 10.4 from the textbook
A = np.array([[3, -0.1, -0.2], [0.1, 7, -0.3], [0.3, -0.2, 10]])
print("A =\n", A)
P, L, U = scipy.linalg.lu(A)
print("P =\n", P)
print("L =\n", L)
print("U =\n", U)
```

Outline

Cholesky Decomposition

Lab Assignment

Cholesky Decomposition

For symmetric matrices (where $[A] = [A]^T$), we have very efficient algorithms for LU decomposition. One of the most widely used approaches is **Cholesky Decomposition**.

This algorithm is based on the fact that LU decompositions of symmetric matrices take the form

$$[A] = [U]^T[U]$$

where $[U]$ is an upper triangular matrix.

Cholesky Decomposition

To see how this works, let's look at a 3-variable system with a symmetric coefficient matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} = \begin{bmatrix} u_{11} & 0 & 0 \\ u_{12} & u_{22} & 0 \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$= \begin{bmatrix} u_{11}^2 & u_{11}u_{12} & u_{11}u_{13} \\ u_{11}u_{12} & u_{12}^2 + u_{22}^2 & u_{12}u_{13} + u_{22}u_{23} \\ u_{11}u_{13} & u_{12}u_{13} + u_{22}u_{23} & u_{13}^2 + u_{23}^2 + u_{33}^2 \end{bmatrix}$$

Since we know each element of $[A]$, we can solve for each element of $[U]$. As it turns out, we can write a set of recurrence equations for the ith row of $[U]$.

Cholesky Decomposition

Cholesky Decomposition

Given a nonsingular symmetric matrix $[A]$, we can decompose the matrix into the product of an upper triangular matrix $[U]$ and its transpose

$$[A] = [U]^T[U].$$

The elements of the ith row of $[U]$ can be determined from the recurrence relations

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2} \quad \text{and}$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} u_{ki}u_{kj}}{u_{ii}} \text{ for } j = i+1, \ldots, n$$

Cholesky Decomposition

Cholesky Decomposition is much more efficient than Guassian Elimination for LU factorization, requiring essentially the same amount of computational effort as back substitution.

However, since this method only applies to symmetric matrices, we still have to resort to other methods much of the time.

Cholesky Decomposition in Python

We can use `scipy.linalg.cholesky` to perform Cholesky decomposition.

Python

```python
import numpy as np
import scipy
# Example 10.6 from the textbook
A = np.array([[6, 15, 55], [15, 55, 225], [55, 225, 979]])
print("A =\n", A)
U = scipy.linalg.cholesky(A)
print("U =\n", U)
A_recovered = np.dot(U.transpose(), U)
print("A (recovered) =\n", A_recovered)
assert(np.allclose(A, A_recovered)) # Raise an AssertionError if not all elements are within 1e-8 of each other
```

numpy.linalg.solve

What does numpy use under the hood to solve linear systems?

Depending on the characteristics of the matrix system, numpy may select from several methods.

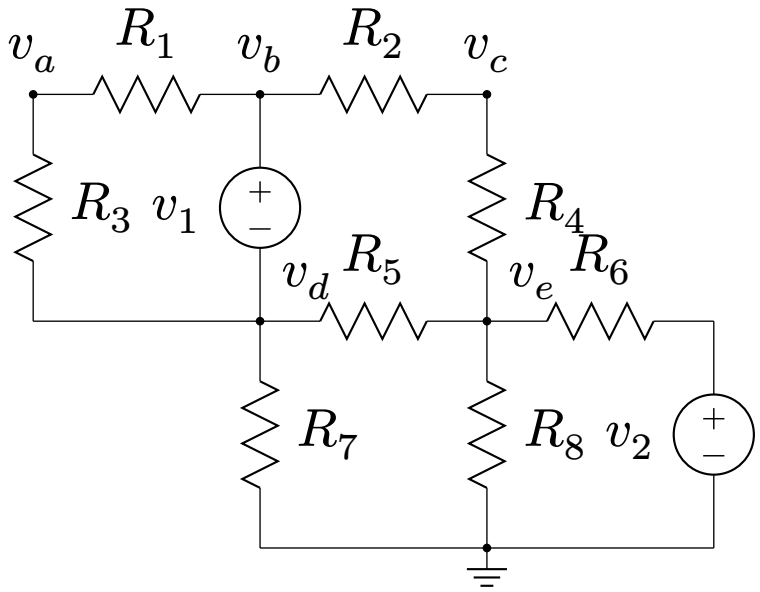
- For most systems numpy will use LU factorization by Guassian Elimination (or a variation thereof).
- For symmetric and tridiagonal systems, numpy can select more efficient methods such as Cholesky decomposition.
- Almost all of this is done by making calls to LAPACK, the Fortran linear algebra package.

Outline

Cholesky Decomposition

Lab Assignment

Node Voltage



Component	Value
v_1	1.8V
v_2	3.3V
R_1	5.1 kΩ
R_2, R_3, R_6	3.3 kΩ
R_4, R_5	390 Ω
R_7, R_8	10 kΩ

Given the circuit depicted above, Write a Python program to find the voltage at each node using Kirchhoff's Current Law and Ohm's Law.

Node Voltage

1. Write a function to perform Gaussian Elimination with partial pivoting. Your program should use this function to compute the solution. (You can test your function with examples from previous slides or the textbook.)
2. (Bonus) Suppose you need the voltage at node c to be exactly 3V. Adjust the value of v_1 so that this is achieved. What is the new supply voltage v_1?
3. Include Python code and a text file containing your computed results in your Canvas submission.

Heat Diffusion

Suppose you're designing a subzero cooling system for a new quantum computer. You're concerned about heat transfer along one of the support columns for the cooling system, wrecking your compute performance.

Steady-state heat transfer can be modeled as an initial value problem in a differential equation:

$$u''(x) = f(x), \quad x \in (a, b)$$
$$u(a) = \alpha, \quad u(b) = \beta$$

In this case a represents the "hot" end of the column where it meets the casing and b represents the "cold" side of the column where the computer is installed.

Heat Diffusion

Initial value problems like this can be approximated with a linear system of equations. Here's the procedure:

1. Divide the domain (a, b) up into equally-spaced points:

$$x = a + ih, \quad h = \frac{b-a}{m+1}$$

2. Write an approximation for the second derivative at each point:

$$u''(x_i) = f(x_i) \quad \rightarrow \quad \frac{1}{h^2}(U_{i-1} - 2U_i + U_{i+1}) \approx f(x_i)$$

3. Formulate this as a linear system:

$$[A]\{U\} = \{f\}$$

Heat Diffusion

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}, \quad \{U\} = \begin{bmatrix} U_1 \\ U_2 \\ \dots \\ U_{m-2} \\ U_{m-1} \end{bmatrix}, \quad \{f\} = \begin{bmatrix} f(x_1) - \frac{\alpha}{h^2} \\ f(x_2) \\ \dots \\ f(x_{m-2}) \\ f(x_{m-1}) - \frac{\beta}{h^2} \end{bmatrix}$$

Heat Diffusion

The support column is 1 meter long, so $a = 0$ and $b = 1$. The computer system will be prechilled to 0.25 degrees before the test starts, so $\beta = 0.25$. The ambient temperature around the casing has been measured at $\alpha = 37$ degrees.

Last but not least, some of your colleages have determined that the initial temperature distribution is given by $f(x) = 5e^{-5x+2}$.

Heat Diffusion

Write a Python program to solve this heat transfer problem to approximate the steady-state temperature distribution $\{U\}$.

1. Divide up the domain $(0, 1)$ using `numpy.linspace(a, b, h)`. Use the equation for h from a couple slides back. Choose m so that you have enough points to make nice plots.
2. Set up the matrix system using the $f(x)$, α and β given on the previous slide.
3. Compute the solution using Cholesky decomposition (You can use `scipy.linalg.cholesky` for this). Plot your solution.
4. (Bonus) Also compute the solution using LU factorization. Measure the time it takes your system to compute each result. Which is faster? Why?
5. Include Python code and plot images in your Canvas submission.