Linear equations 1

Linear algebra basics

Dr.ir. Ivo Roghair, Prof.dr.ir. Martin van Sint Annaland

Chemical Process Intensification group
Eindhoven University of Technology

Numerical Methods (6BER03), 2024-2025

Today's outline

- Introduction
- Matrix inversion
- Solving a linear system
- Towards larger systems
- Summary

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Today's outline

- Introduction
- Matrix inversion
- Solving a linear system
- Towards larger systems
- Summary

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Overview

Goals

- Different ways of looking at a system of linear equations
- Determination of the inverse, determinant and the rank of a matrix
- The existence of a solution to a set of linear equations

Different views of linear systems

- Separate equations:

$$\begin{aligned} x + y + z &= 4 \\ 2x + y + 3z &= 7 \\ 3x + y + 6z &= 5 \end{aligned}$$

Different views of linear systems

- Separate equations:

$$\begin{aligned} x + y + z &= 4 \\ 2x + y + 3z &= 7 \\ 3x + y + 6z &= 5 \end{aligned}$$

- Matrix mapping $Mx = b$:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

Different views of linear systems

- Separate equations:

$$\begin{aligned} x + y + z &= 4 \\ 2x + y + 3z &= 7 \\ 3x + y + 6z &= 5 \end{aligned}$$

- Matrix mapping $Mx = b$:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

- Linear combination:

$$x \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + y \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + z \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

Different views of linear systems

- Separate equations:

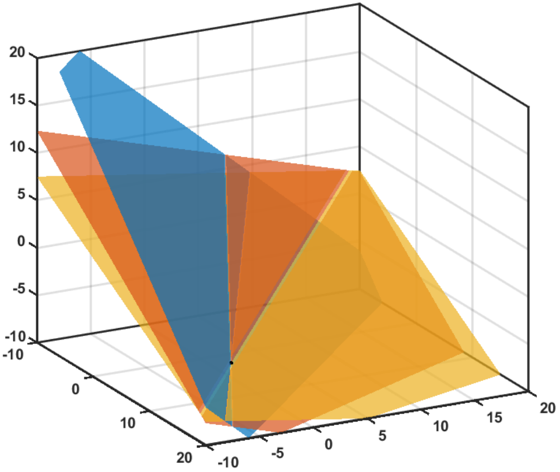
$$\begin{aligned} x+y+z &= 4 \\ 2x+y+3z &= 7 \\ 3x+y+6z &= 5 \end{aligned}$$

- Matrix mapping $Mx = b$:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

- Linear combination:

$$x \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + y \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + z \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$



Today's outline

- Introduction
- Matrix inversion
- Solving a linear system
- Towards larger systems
- Summary

Inverse of a matrix

- The inverse M^{-1} is defined such that:

$$MM^{-1} = I \quad \text{and} \quad M^{-1}M = I$$

- Use the inverse to solve a set of linear equations:

$$\begin{aligned} M\boldsymbol{x} &= \boldsymbol{b} \\ M^{-1}M\boldsymbol{x} &= M^{-1}\boldsymbol{b} \\ I\boldsymbol{x} &= M^{-1}\boldsymbol{b} \\ \boldsymbol{x} &= M^{-1}\boldsymbol{b} \end{aligned}$$

How to calculate the inverse?

- The inverse of an $N \times N$ matrix can be calculated using the co-factors of each element of the matrix:

$$M^{-1} = \frac{1}{\det\left|M\right|}\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix}^T$$

- $\det\left|M\right|$ is the *determinant* of matrix M.
- C_{ij} is the *co-factor* of the ij^{th} element in M.

Computing the co-factors

Consider the following example matrix: $M = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}$

Computing the co-factors

Consider the following example matrix: $M = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}$

A co-factor (e.g. C_{11}) is the determinant of the elements left over when you cover up the row and column of the element in question, multiplied by ± 1, depending on the position.

$$\begin{bmatrix} \boxed{1} & \times & \times \\ \times & 1 & 3 \\ \times & 1 & 6 \end{bmatrix}$$

Computing the co-factors

Consider the following example matrix: $M = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}$

A co-factor (e.g. C_{11}) is the determinant of the elements left over when you cover up the row and column of the element in question, multiplied by ± 1, depending on the position.

$$\begin{bmatrix} 1 & \times & \times \\ \times & 1 & 3 \\ \times & 1 & 6 \end{bmatrix} \qquad \begin{bmatrix} + & - & + \\ - & + & - \\ + & - & + \end{bmatrix}$$

Computing the co-factors

Consider the following example matrix: $M = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}$

A co-factor (e.g. C_{11}) is the determinant of the elements left over when you cover up the row and column of the element in question, multiplied by ± 1, depending on the position.

$$\begin{bmatrix} 1 & \times & \times \\ \times & 1 & 3 \\ \times & 1 & 6 \end{bmatrix} \qquad \begin{bmatrix} + & - & + \\ - & + & - \\ + & - & + \end{bmatrix} \qquad C_{11} = +1 \cdot \det \begin{vmatrix} 1 & 3 \\ 1 & 6 \end{vmatrix}$$

$$= 6 \times 1 - 3 \times 1 = 3$$

Computing the co-factors

Back to our example:

$$M^{-1} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}^{-1} = \frac{1}{\det |M|} \begin{bmatrix} 3 & -3 & -1 \\ -5 & 3 & 2 \\ 2 & -1 & -1 \end{bmatrix}^{T}$$

Computing the co-factors

Back to our example:

$$M^{-1} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}^{-1} = \frac{1}{\det\left|M\right|} \begin{bmatrix} 3 & -3 & -1 \\ -5 & 3 & 2 \\ 2 & -1 & -1 \end{bmatrix}^{T}$$

- The determinant is very important
- If $\det\left|M\right| = 0$, the inverse does not exist (singular matrix)

Calculating the determinant

Compute the determinant by multiplication of each element on a row (or column) by its cofactor and adding the results:

$$\det\left|\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}\right| = +\det\left|\begin{bmatrix} 1 & 3 \\ 1 & 6 \end{bmatrix}\right| - \det\left|\begin{bmatrix} 2 & 3 \\ 3 & 6 \end{bmatrix}\right| + \det\left|\begin{bmatrix} 2 & 1 \\ 3 & 1 \end{bmatrix}\right| = -1$$

Calculating the determinant

Compute the determinant by multiplication of each element on a row (or column) by its cofactor and adding the results:

$$\det\left|\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}\right| = +\det\left|\begin{bmatrix} 1 & 3 \\ 1 & 6 \end{bmatrix}\right| - \det\left|\begin{bmatrix} 2 & 3 \\ 3 & 6 \end{bmatrix}\right| + \det\left|\begin{bmatrix} 2 & 1 \\ 3 & 1 \end{bmatrix}\right| = -1$$

$$\det\left|\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}\right| = +\det\left|\begin{bmatrix} 2 & 1 \\ 3 & 1 \end{bmatrix}\right| - 3\det\left|\begin{bmatrix} 1 & 1 \\ 3 & 1 \end{bmatrix}\right| + 6\det\left|\begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}\right| = -1$$

Today's outline

Solving a linear system

- Our example:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

Solving a linear system

- Our example:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

- The solution is:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = M^{-1}b = \frac{1}{-1}\begin{bmatrix} 3 & -5 & 2 \\ -3 & 3 & -1 \\ -1 & 2 & -1 \end{bmatrix}\begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix} = \frac{1}{-1}\begin{bmatrix} -13 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 13 \\ -4 \\ -5 \end{bmatrix}$$

Solving a linear system

- Our example:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 3 \\ 3 & 1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix}$$

- The solution is:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = M^{-1} b = \frac{1}{-1} \begin{bmatrix} 3 & -5 & 2 \\ -3 & 3 & -1 \\ -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 4 \\ 7 \\ 5 \end{bmatrix} = \frac{1}{-1} \begin{bmatrix} -13 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 13 \\ -4 \\ -5 \end{bmatrix}$$

- The inverse exists, because $\det \left| M \right| = -1$.

Solving a linear system in Python using the inverse

- Create the matrix:

```
>>> A = np.array([[1, 1, 1], [2, 1, 3], [3, 1, 6]])
```

Solving a linear system in Python using the inverse

- Create the matrix:

```
>>> A = np.array([[1, 1, 1], [2, 1, 3], [3, 1, 6]])
```

- Create solution vector:

```
>>> b = np.array([4, 7, 5])
```

Solving a linear system in Python using the inverse

- Create the matrix:

```
>>> A = np.array([[1, 1, 1], [2, 1, 3], [3, 1, 6]])
```

- Create solution vector:

```
>>> b = np.array([4, 7, 5])
```

- Get the matrix inverse:

```
>>> Ainv = np.linalg.inv(A)
```

Solving a linear system in Python using the inverse

- Create the matrix:

```
>>> A = np.array([[1, 1, 1], [2, 1, 3], [3, 1, 6]])
```

- Create solution vector:

```
>>> b = np.array([4, 7, 5])
```

- Get the matrix inverse:

```
>>> Ainv = np.linalg.inv(A)
```

- Compute the solution:

```
>>> x = np.dot(Ainv, b)
```

Solving a linear system in Python using the inverse

- Create the matrix:

```
>>> A = np.array([[1, 1, 1], [2, 1, 3], [3, 1, 6]])
```

- Create solution vector:

```
>>> b = np.array([4, 7, 5])
```

- Get the matrix inverse:

```
>>> Ainv = np.linalg.inv(A)
```

- Compute the solution:

```
>>> x = np.dot(Ainv, b)
```

- Python's internal direct solver:

```
>>> x = np.linalg.solve(A, b)
```

- These are black boxes! We are going over some methods later!

Exercise: performance of inverse computation

Create a script that generates matrices with random elements of various sizes $N \times N$ (e.g. values of $N \in \{10, 20, 50, 100, 200, \ldots, 5000, 10000\}$). Compute the inverse of each matrix, and use `import time` and `time.time()` to see the computing time for each inversion. Plot the time as a function of the matrix size N.

Exercise: performance of inverse computation

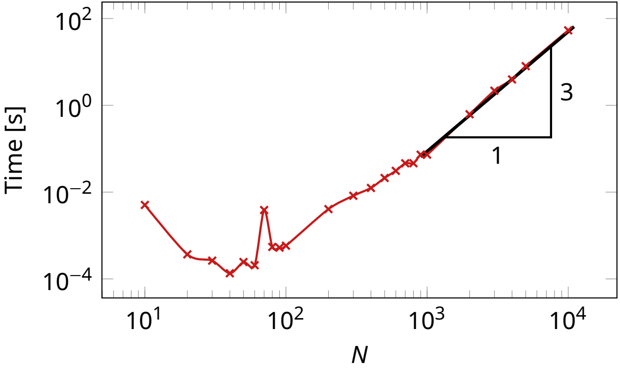
Create a script that generates matrices with random elements of various sizes $N \times N$ (e.g. values of $N \in \{10, 20, 50, 100, 200, \ldots, 5000, 10000\}$). Compute the inverse of each matrix, and use `import time` and `time.time()` to see the computing time for each inversion. Plot the time as a function of the matrix size N.

```python
import numpy as np
import matplotlib.pyplot as plt
import time

# Generate random matrices of various sizes 's'.
# Invert the matrices and store the time required
# for the inversion. Plot the times vs 's'
s = np.array([10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000])
t_inv = []
for n in s:
    print(f'Working on size {n}')
    A = np.random.rand(n, n)
    start_time = time.time()
    Ainv = np.linalg.inv(A)
    t_inv.append(time.time() - start_time)

plt.loglog(s, t_inv)
plt.xlabel('N')
plt.ylabel('Time [s]')
plt.show()
```

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Exercise: sample results

Each computer produces slightly different results because of background tasks, different matrices, etc. This is especially noticable for small systems.



The time increases by 3 orders of magnitude, for every magnitude in N. The *computational complexity* of matrix inversion scales with $\mathcal{O}(N^3)$!

Today's outline

- Introduction
- Matrix inversion
- Solving a linear system
- Towards larger systems
- Summary

Towards larger systems

Computation of determinants and inverses of large matrices in this way is too difficult (slow), so we need other methods to solve large linear systems!

Towards larger systems

- Determinant of upper triangular matrix:

$$\det\left|M_{\text{tri}}\right| = \prod_{i=1}^{n} a_{ii} \qquad M = \begin{bmatrix} 5 & 3 & 2 \\ 0 & 9 & 1 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \det\left|M\right| = 5 \times 9 \times 1 = 45$$

- Matrix multiplication:

$$\det\left|AM\right| = \det\left|A\right| \times \det\left|M\right|$$

- When A is an identity matrix ($\det\left|A\right| = 1$):

$$\det\left|AM\right| = \det\left|A\right| \times \det\left|M\right| = 1 \times \det\left|M\right|$$

- With rules like this, we can use row-operations so that we can compute the determinant more cheaply.

Solutions of linear systems

Rank of a matrix: the number of linearly independent columns (columns that can not be expressed as a linear combination of the other columns) of a matrix.

$$M = \begin{bmatrix} 5 & 3 & 2 \\ 0 & 9 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- 3 independent columns
- In Python:

```
>>> numpy.linalg.matrix_rank(M)
```

$$M = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- col 2 = 2× col 1
- col 4 = col 3 − col 1
- 2 independent columns: rank = 2

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Solutions of linear systems

The solution of a system of linear equations may or may not exist, and it may or may not be unique. Existence of solutions can be determined by comparing the rank of the Matrix M with the rank of the augmented matrix M_a:

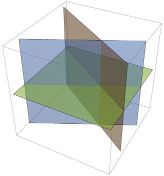
```
>>> numpy.linalg.matrix_rank(A)
>>> numpy.linalg.matrix_rank(np.column_stack((A,b))) # Concatenated matrices
```

Our system: $Mx = b$

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \Rightarrow M_a = \begin{bmatrix} M_{11} & M_{12} & M_{13} & b_1 \\ M_{21} & M_{22} & M_{23} & b_2 \\ M_{31} & M_{32} & M_{33} & b_3 \end{bmatrix}$$

Existence of solutions for linear systems

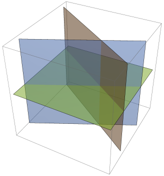
For a matrix M of size $n \times n$, and augmented matrix M_a:



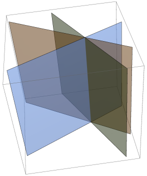
- $\mathrm{Rank}(M) = n$:
  Unique solution

Existence of solutions for linear systems

For a matrix M of size $n \times n$, and augmented matrix M_a:



- $\text{Rank}(M) = n$:
  Unique solution

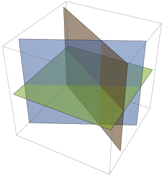


- $\text{Rank}(M) = \text{Rank}(M_a) < n$:
  Infinite number of solutions

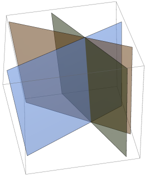
Existence of solutions for linear systems

For a matrix M of size $n \times n$, and augmented matrix M_a:

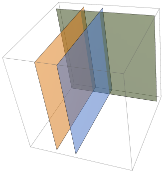
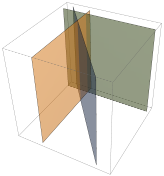
- $\text{Rank}(M) = n$:
  Unique solution



- $\text{Rank}(M) = \text{Rank}(M_a) < n$:
  Infinite number of solutions



- $\text{Rank}(M) < n$, $\text{Rank}(M) < \text{Rank}(M_a)$:
  No solutions

Two examples

$$M = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 3 & 1 \\ 0 & 0 & 2 \end{bmatrix} \quad b = \begin{bmatrix} 17 \\ 11 \\ 4 \end{bmatrix} \Rightarrow M_a = \begin{bmatrix} 1 & 1 & 2 & 17 \\ 0 & 3 & 1 & 11 \\ 0 & 0 & 2 & 4 \end{bmatrix}$$

$\mathrm{rank}(M) = 3 = n \Rightarrow$ Unique solution

Two examples

$$M = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 3 & 1 \\ 0 & 0 & 2 \end{bmatrix} \quad b = \begin{bmatrix} 17 \\ 11 \\ 4 \end{bmatrix} \Rightarrow M_a = \begin{bmatrix} 1 & 1 & 2 & 17 \\ 0 & 3 & 1 & 11 \\ 0 & 0 & 2 & 4 \end{bmatrix}$$

$\text{rank}(M) = 3 = n \Rightarrow$ Unique solution

$$M = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 3 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad b = \begin{bmatrix} 17 \\ 11 \\ 0 \end{bmatrix} \Rightarrow M_a = \begin{bmatrix} 1 & 1 & 2 & 17 \\ 0 & 3 & 1 & 11 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$\text{rank}(M) = \text{rank}(M_a) = 2 < n \Rightarrow$ Infinite number of solutions

Today's outline

- Introduction
- Matrix inversion
- Solving a linear system
- Towards larger systems
- Summary

Summary

- Linear equations can be written as matrices
- Using the inverse, the solution can be determined
  - Inverse via cofactors
  - Inverse and solution in Python
- Introduced the concept of computational complexity: matrix inversion scales with N^3
- A solution depends on the rank of a matrix

Linear equations 2

Direct methods

Dr.ir. Ivo Roghair, Prof.dr.ir. Martin van Sint Annaland

Chemical Process Intensification group
Eindhoven University of Technology

Numerical Methods (6BER03), 2024-2025

Today's outline

- Introduction
- Gauss elimination
- Partial Pivoting
- LU decomposition
- Summary

Today's outline

- Introduction
- Gauss elimination
- Partial Pivoting
- LU decomposition
- Summary

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Overview

Goals

Today we are going to write a program, which can solve a set of linear equations

- The first method is called Gaussian elimination
- We will encounter some problems with Gaussian elimination
- Then LU decomposition will be introduced

Today's outline

- Introduction
- Gauss elimination
- Partial Pivoting
- LU decomposition
- Summary

Define the linear system

Consider the system:

$$Ax = b$$

In general:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

Desired solution:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \end{bmatrix}$$

Using row operations

- Use row operations to simplify the system. Eliminate element A_{10} by subtracting $A_{10}/A_{00} = d_{10}$ times row 1 from row 2.
- In this case, Row 1 is the pivot row, and A_{00} is the pivot element.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ A_{10} & A_{11} & A_{12} & b_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right]$$

Using row operations

Eliminate element A_{10} using $d_{10} = A_{10}/A_{00}$.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ A_{10} & A_{11} & A_{12} & b_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right]$$

Using row operations

Eliminate element A_{10} using $d_{10} = A_{10}/A_{00}$.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ A_{10} & A_{11} & A_{12} & b_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right]$$

- $d_{10} \rightarrow A_{10}/A_{00}$
- $A_{10} \rightarrow A_{10} - A_{00}d_{10}$
- $A_{11} \rightarrow A_{11} - A_{01}d_{10}$
- $A_{12} \rightarrow A_{12} - A_{02}d_{10}$
- $b_1 \rightarrow b_1 - b_0d_{10}$

Using row operations

Eliminate element A_{10} using $d_{10} = A_{10}/A_{00}$.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ A_{10} & A_{11} & A_{12} & b_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right]$$

- $d_{10} \to A_{10}/A_{00}$
- $A_{10} \to A_{10} - A_{00}d_{10}$
- $A_{11} \to A_{11} - A_{01}d_{10}$
- $A_{12} \to A_{12} - A_{02}d_{10}$
- $b_1 \to b_1 - b_0 d_{10}$

```
d10 = A[1,0] / A[0,0]

A[1,0] = A[1,0] - A[0,0] * d10
A[1,1] = A[1,1] - A[0,1] * d10
A[1,2] = A[1,2] - A[0,2] * d10

b[1] = b[1] - b[0] * d10
```

Using row operations

Eliminate element A_{20} using $d_{20} = A_{20}/A_{00}$.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & A'_{21} & A'_{22} & b'_2 \end{array}\right]$$

Using row operations

Eliminate element A_{20} using $d_{20} = A_{20}/A_{00}$.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ A_{20} & A_{21} & A_{22} & b_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & A'_{21} & A'_{22} & b'_2 \end{array}\right]$$

- $d_{20} \to A_{20}/A_{00}$
- $A_{20} \to A_{20} - A_{00}d_{20}$
- $A_{21} \to A_{21} - A_{01}d_{20}$
- $A_{22} \to A_{22} - A_{02}d_{20}$
- $b_2 \to b_2 - b_0 d_{20}$

```
d20 = A[2, 0] / A[0, 0]

A[2, 0] = A[2, 0] - A[0, 0] * d20
A[2, 1] = A[2, 1] - A[0, 1] * d20
A[2, 2] = A[2, 2] - A[0, 2] * d20
b[2] = b[2] - b[0] * d20
```

Using row operations

Eliminate element A'_{21} using $d_{21} = A'_{21}/A'_{11}$. Note that now the second row has become the pivot row.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & A'_{21} & A'_{22} & b'_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & 0 & A''_{22} & b''_2 \end{array}\right]$$

Using row operations

Eliminate element A'_{21} using $d_{21} = A'_{21}/A'_{11}$. Note that now the second row has become the pivot row.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & A'_{21} & A'_{22} & b'_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & 0 & A''_{22} & b''_2 \end{array}\right]$$

- $d_{21} \to A_{21}/A'_{11}$
- $A_{21} \to A_{21} - A'_{11} d_{21}$
- $A_{22} \to A_{22} - A'_{12} d_{21}$
- $b_2 \to b_2 - b'_2 d_{21}$

```
d21 = A[2, 1] / A[1, 1]
A[2, 1] = A[2, 1] - A[1, 1] * d21
A[2, 2] = A[2, 2] - A[1, 2] * d21
b[2] = b[2] - b[1] * d21
```

Using row operations

Eliminate element A'_{21} using $d_{21} = A'_{21}/A'_{11}$. Note that now the second row has become the pivot row.

$$\left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & A'_{21} & A'_{22} & b'_2 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} A_{00} & A_{01} & A_{02} & b_0 \\ 0 & A'_{11} & A'_{12} & b'_1 \\ 0 & 0 & A''_{22} & b''_2 \end{array}\right]$$

- $d_{21} \rightarrow A_{21}/A'_{11}$
- $A_{21} \rightarrow A_{21} - A'_{11} d_{21}$
- $A_{22} \rightarrow A_{22} - A'_{12} d_{21}$
- $b_2 \rightarrow b_2 - b'_2 d_{21}$

```
d21 = A[2, 1] / A[1, 1]
A[2, 1] = A[2, 1] - A[1, 1] * d21
A[2, 2] = A[2, 2] - A[1, 2] * d21
b[2] = b[2] - b[1] * d21
```

The matrix is now a triangular matrix — the solution can be obtained by back-substitution.

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Backsubstitution

The system now reads:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ 0 & A'_{11} & A'_{12} \\ 0 & 0 & A''_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b'_1 \\ b''_2 \end{bmatrix}$$

Backsubstitution

The system now reads:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ 0 & A'_{11} & A'_{12} \\ 0 & 0 & A''_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b'_1 \\ b''_2 \end{bmatrix}$$

Start at the last row N, and work upward until row 1.

$$x_2 = b''_2 / A''_{22}$$
$$x_1 = (b'_1 - A'_{12} x_2)/A'_{11}$$
$$x_0 = (b_0 - A_{01} x_1 - A_{02} x_2)/A_{00}$$

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Backsubstitution

The system now reads:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ 0 & A'_{11} & A'_{12} \\ 0 & 0 & A''_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b'_1 \\ b''_2 \end{bmatrix}$$

Start at the last row N, and work upward until row 1.

$$x_2 = b''_2 / A''_{22}$$
$$x_1 = (b'_1 - A'_{12} x_2) / A'_{11}$$
$$x_0 = (b_0 - A_{01} x_1 - A_{02} x_2) / A_{00}$$

```python
x = np.empty_like(b)
x[2] = b[2] / A[2,2]
x[1] = (b[1] - A[1,2] * x[2]) / A[1,1]
x[0] = (b[0] - A[0,1] * x[1] - A[0,2] * x[2]) / A[0,0]
```

In general:

$$x_N = \frac{b_N}{A_{NN}} \qquad x_i = \frac{b_i - \sum_{j=i+1}^{N} A_{ij} x_j}{A_{ii}}$$

Writing the program

- Create a function that provides the framework: take matrix A and vector b as an input, and return the solution x as output:

```
def gaussian_eliminate(A, b):
    pass # Your implementation here
```

- We will use *for-loops* instead of typing out each command line.
- Useful Python (with NumPy) shortcuts:
  - `A[0, :]` $= [A_{00}, A_{01}, A_{02}]$
  - `A[:, 1]` $= [A_{01}, A_{11}, A_{21}]$
  - `A[0, 1:]` $= [A_{01}, A_{02}]$
- A row operation could look like:

```
A[i, :] = A[i, :] - d * A[0, :]
```

The program: elimination step

An initial draft could look like:

```python
def gaussian_eliminate_draft(A,b):
    """Perform elimination to obtain an upper triangular matrix"""
    A = np.array(A,dtype=np.float64)
    b = np.array(b,dtype=np.float64)

    assert A.shape[0] == A.shape[1], "Coefficient matrix should be square"

    N = len(b)
    for col in range(N-1): # Select pivot
        for row in range(col+1,N): # Loop over rows below pivot
            d = A[row,col] / A[col,col] # Choose elimination factor
            for element in range(row,N): # Elements from diagonal to right
                A[row,element] = A[row,element] - d * A[col,element]
            b[row] = b[row] - d * b[col]

    return A,b
```

The program: elimination step

Employing some of the row operations to create `gaussian_eliminate_v1`:

```python
for element in range(row,N):
    A[row,element] = A[row,element] - d * A[col,element]
```

```python
A[row,:] = A[row,:] - d * A[col,:]
```

EINDHOVEN UNIVERSITY OF TECHNOLOGY

The program: elimination step

Employing some of the row operations to create `gaussian_eliminate_v1`:

```python
for element in range(row,N):
    A[row,element] = A[row,element] - d * A[col,element]
```

```python
A[row,:] = A[row,:] - d * A[col,:]
```

```python
def gaussian_eliminate_v1(A,b):
    A = np.array(A,dtype=np.float64)
    b = np.array(b,dtype=np.float64)

    assert A.shape[0] == A.shape[1], "Coefficient matrix should be square"

    N = len(b)
    for col in range(N-1):
        for row in range(col+1,N):
            d = A[row,col] / A[col,col]
            A[row,:] = A[row,:] - d * A[col,:]
            b[row] = b[row] - d * b[col]

    return A,b
```

Testing

Let's try to eliminate our linear system! If you create/downloaded our file `gaussjordan.py`, you can access the functions by importing them. The file should be stored where your own code/notebook is:

```python
from gaussjordan import gaussian_eliminate_draft,gaussian_eliminate_v1
import numpy as np

A = np.array([[1, 1, 1], [2, 1, 3], [3, 1, 6]])
b = np.array([4, 7, 5])

Aprime,bprime = gaussian_eliminate_draft(A,b)
print(Aprime)
print(bprime)
```

The program: Backsubstitution

Now we have elimination working, let's create a back substitution algorithm too. Recall:

$$x_N = \frac{b_N}{A_{NN}} \qquad x_i = \frac{b_i - \sum_{j=i+1}^{N} A_{ij}x_j}{A_{ii}}$$

```python
def backsubstitution_draft(A, b):
    """Back substitutes an upper triangular matrix to
        find x in Ax=b"""
    x = np.copy(b)
    N = len(b)

    for row in range(N-1, -1, -1):
        for i in range(row+1, N):
            x[row] = x[row] - A[row, i] * x[i]
        x[row] = x[row] / A[row, row]

    return x
```

EINDHOVEN UNIVERSITY OF TECHNOLOGY

The program: Backsubstitution

Now we have elimination working, let's create a back substitution algorithm too. Recall:

$$x_N = \frac{b_N}{A_{NN}} \qquad x_i = \frac{b_i - \sum_{j=i+1}^{N} A_{ij} x_j}{A_{ii}}$$

```python
def backsubstitution_draft(A, b):
    """Back substitutes an upper triangular matrix to
        find x in Ax=b"""
    x = np.copy(b)
    N = len(b)

    for row in range(N-1, -1, -1):
        for i in range(row+1, N):
            x[row] = x[row] - A[row, i] * x[i]
        x[row] = x[row] / A[row, row]

    return x
```

```python
def backsubstitution_v1(A,b):
    """Back substitutes an upper triangular matrix to find x in Ax=b"""
    x = np.empty_like(b)
    N = len(b)

    for row in range(N)[::-1]:
        x[row] = (b[row] - np.sum(A[row,row+1:] * x[row+1:])) / A[row,row]

    return x
```

A full Gauss Elimination solver

- The functions we just built are distributed via Canvas too
- Use `help GaussianEliminate` to find out how it works
- Solve the following system of equations:

$$\begin{bmatrix} 9 & 9 & 5 & 2 \\ 6 & 7 & 1 & 3 \\ 6 & 4 & 3 & 5 \\ 2 & 6 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \\ 10 \\ 1 \end{bmatrix}$$

- Compare your solution with `np.linalg.solve(A,b)`

Today's outline

Partial pivoting

- Now try to run the algorithm with the following system:

$$\begin{bmatrix} 0 & 2 & 1 \\ 3 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 10 \end{bmatrix}$$

Partial pivoting

- Now try to run the algorithm with the following system:

$$\begin{bmatrix} 0 & 2 & 1 \\ 3 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 10 \end{bmatrix}$$

- It does not work! Division by zero, due to $A_{11} = 0$.
- Solution: Swap rows to move largest element to the diagonal.

Partial pivoting: implementing row swaps

- Find maximum element row below pivot in current column

```
index = np.argmax(np.abs(A[col:, col])) + col
```

Partial pivoting: implementing row swaps

- Find maximum element row below pivot in current column

```
index = np.argmax(np.abs(A[col:, col])) + col
```

- Store current row

```
temp = A[column,:]
```

Partial pivoting: implementing row swaps

- Find maximum element row below pivot in current column

```
index = np.argmax(np.abs(A[col:, col])) + col
```

- Store current row

```
temp = A[column,:]
```

- Swap pivot row and desired row in `A`

```
A[column,:] = A[index,:]
A[index,:] = temp
```

Partial pivoting: implementing row swaps

- Find maximum element row below pivot in current column

```
index = np.argmax(np.abs(A[col:, col])) + col
```

- Store current row

```
temp = A[column,:]
```

- Swap pivot row and desired row in `A`

```
A[column,:] = A[index,:]
A[index,:] = temp
```

- Do the same for `b` — store and swap

```
temp = b[column]
b[column] = b[index]
b[index] = temp
```

Adding the partial pivoting rules

```python
def gaussian_eliminate_partial_pivot(A,b):
    A = np.array(A,dtype=np.float64)
    b = np.array(b,dtype=np.float64)

    assert A.shape[0] == A.shape[1], "Coefficient matrix should be square"

    N = len(b)
    for col in range(N-1):
        index = np.argmax(np.abs(A[col:, col])) + col
        temp = A[col,:]
        A[col,:] = A[index,:]
        A[index,:] = temp

        temp = b[col]
        b[col] = b[index]
        b[index] = temp
        for row in range(col+1,N):
            d = A[row,col] / A[col,col]
            A[row,:] = A[row,:] - d * A[col,:]
            b[row] = b[row] - d * b[col]

    return A,b
```

Improve the program by using re-usable functions

```python
def swap_rows(mat,i1,i2):
    """Swap two rows in a matrix/vector"""
    temp = mat[i1,...].copy()
    mat[i1,...] = mat[i2,...]
    mat[i2,...] = temp
```

```python
def gaussian_eliminate_v2(A,b):
    A = np.array(A,dtype=np.float64)
    b = np.array(b,dtype=np.float64)

    assert A.shape[0] == A.shape[1], "Coefficient matrix should be square"

    N = len(b)
    for col in range(N-1):
        index = np.argmax(np.abs(A[col:, col])) + col
        swap_rows(A,col,index)
        swap_rows(b,col,index)
        for row in range(col+1,N):
            d = A[row,col] / A[col,col]
            A[row,:] = A[row,:] - d * A[col,:]
            b[row] = b[row] - d * b[col]

    return A,b
```

Alternatives to this program

- Python can compute the solution to Ax=b with `scipy.linalg.solve` or `numpy.linalg.solve` solvers (more efficient)
- Too many loops. Loops make Python slow.
- There are fundamental problems with Gaussian elimination

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Alternatives to this program

- Python can compute the solution to Ax=b with `scipy.linalg.solve` or `numpy.linalg.solve` solvers (more efficient)
- Too many loops. Loops make Python slow.
- There are fundamental problems with Gaussian elimination
  - You can add a counter to the algorithm to see how many subtraction and multiplication operations it performs for a given size of matrix A.
  - The number of operations to perform Gaussian elimination is $\mathcal{O}(2N^3)$ (where N is the number of equations)
  - Exercise: verify this for our script

Alternatives to this program

- Python can compute the solution to Ax=b with `scipy.linalg.solve` or `numpy.linalg.solve` solvers (more efficient)
- Too many loops. Loops make Python slow.
- There are fundamental problems with Gaussian elimination
  - You can add a counter to the algorithm to see how many subtraction and multiplication operations it performs for a given size of matrix A.
  - The number of operations to perform Gaussian elimination is $\mathcal{O}(2N^3)$ (where N is the number of equations)
  - Exercise: verify this for our script
  - LU decomposition takes $\mathcal{O}(2N^3/3)$ flops, 3 times less!
  - Forward and backward substitution each take $\mathcal{O}(N^2)$ flops (both cases)

Today's outline

- Introduction
- Gauss elimination
- Partial Pivoting
- LU decomposition
- Summary

LU Decomposition

Suppose we want to solve the previous set of equations, but with several right hand sides:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots \\ x_1 & x_2 & x_3 \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ b_1 & b_2 & b_3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

LU Decomposition

Suppose we want to solve the previous set of equations, but with several right hand sides:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots \\ x_1 & x_2 & x_3 \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ b_1 & b_2 & b_3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Factor the matrix A into two matrices, L and U, such that $A = LU$:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \times & 1 & 0 \\ \times & \times & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}$$

Now we can solve for each right hand side, using only a forward followed by a backward substitution!

Substitutions

- Define a lower and upper matrix L and U so that $A = LU$
- Therefore $LUx = b$
- Define a new vector $y = Ux$ so that $Ly = b$
- Solve for y, use L and forward substitution
- Then we have y, solve for x, use $Ux = y$
- Solve for x, use U and backward substitution
- But how to get L and U?

Decomposing the matrix (1)

When we eliminate the element A_{21} we can keep multiplying by a matrix that undoes this row operations, so that the product remains equal to A.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} - d_{21}A_{12} & A_{23} - d_{21}A_{13} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

Decomposing the matrix (2)

When we eliminate the element A_{31} we can keep multiplying by a matrix that undoes this row operations, so that the product remains equal to A.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{22} = A_{22} - d_{21}A_{12} & A'_{23} = A_{23} - d_{21}A_{13} \\ 0 & A'_{32} = A_{32} - d_{31}A_{12} & A'_{33} = A_{33} - d_{31}A_{21} \end{bmatrix}$$

Decomposing the matrix (3)

When we eliminate the element A_{32} we can keep multiplying by a matrix that undoes this row operations, so that the product remains equal to A.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & d_{32} & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A''_{33} = A'_{33} - d_{32}A'_{23} \end{bmatrix}$$

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Decomposing the matrix (3)

When we eliminate the element A_{32} we can keep multiplying by a matrix that undoes this row operations, so that the product remains equal to A.

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & d_{32} & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A''_{33} = A'_{33} - d_{32}A'_{23} \end{bmatrix}$$

We now have a lower matrix L and an upper matrix U. This finishes the LU decomposition!

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Pivoting during decomposition

Suppose we have arrived at the situation below, where $A'_{32} > A'_{22}$:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & \boxed{A'_{22}} & A'_{23} \\ 0 & \boxed{A'_{32}} & A'_{33} \end{bmatrix}$$

Pivoting during decomposition

Suppose we have arrived at the situation below, where $A'_{32} > A'_{22}$:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & A'_{32} & A'_{33} \end{bmatrix}$$

Exchange rows 2 and 3 to get the largest value on the main diagonal. Use a permutation matrix to store the swapped rows:

Pivoting during decomposition

Suppose we have arrived at the situation below, where $A'_{32} > A'_{22}$:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & A'_{32} & A'_{33} \end{bmatrix}$$

Exchange rows 2 and 3 to get the largest value on the main diagonal. Use a permutation matrix to store the swapped rows:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{31} & 0 & 1 \\ d_{21} & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{32} & A'_{33} \\ 0 & A'_{22} & A'_{23} \end{bmatrix}$$

Pivoting during decomposition

Suppose we have arrived at the situation below, where $A'_{32} > A'_{22}$:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{21} & 1 & 0 \\ d_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & A'_{32} & A'_{33} \end{bmatrix}$$

Exchange rows 2 and 3 to get the largest value on the main diagonal. Use a permutation matrix to store the swapped rows:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ d_{31} & 0 & 1 \\ d_{21} & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A'_{32} & A'_{33} \\ 0 & A'_{22} & A'_{23} \end{bmatrix}$$

Multiplying with a permutation matrix will swap the rows of a matrix. The permutation matrix is just an identity matrix, whose rows have been interchanged.

Recipe for LU decomposition

When decomposing matrix A into $A = LU$, it may be beneficial to swap rows to get the largest values on the diagonal of U (pivoting). A permutation matrix P is used to store row swapping such that:

$$PA = LU$$

- Write down a permutation matrix and the linear system
- Promote the largest value in the column diagonal
- Eliminate all elements below diagonal
- Move on to the next column and move largest elements to diagonal
- Eliminate elements below diagonal
- Repeat 5 and 6
- Write down L,U and P

EINDHOVEN UNIVERSITY OF TECHNOLOGY

LU decomposition example (1)

Write down a permutation matrix, which starts as the identity matrix, and the linear system:

$$PA = LU$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

LU decomposition example (1)

Write down a permutation matrix, which starts as the identity matrix, and the linear system:

$$PA = LU$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

Promote the largest value into the diagonal of column 1 — swap row 1 and 2:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

LU decomposition example (2)

Eliminate all elements below the diagonal — row 2 already contains a zero in column 1, row 3 = row 3 - 0.5 row 1. Record the multiplier 0.5 in *L*:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1.5 & -0.5 \end{bmatrix}$$

LU decomposition example (2)

Eliminate all elements below the diagonal — row 2 already contains a zero in column 1, row 3 = row 3 - 0.5 row 1. Record the multiplier 0.5 in L:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1.5 & -0.5 \end{bmatrix}$$

Elimination of column 1 is done. Now step to the next column, and move the largest value be lower triangle of L accordingly:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & -0.5 \\ 0 & 1 & 1 \end{bmatrix}$$

LU decomposition example (3)

Eliminate all elements below the diagonal —
row 3 = row 3 - $\frac{2}{3}$row 2. Record the multiplier $\frac{2}{3}$ in L:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & \frac{2}{3} & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & -0.5 \\ 0 & 0 & \frac{4}{3} \end{bmatrix}$$

EINDHOVEN UNIVERSITY OF TECHNOLOGY

LU decomposition example (3)

Eliminate all elements below the diagonal —
row 3 = row 3 - $\frac{2}{3}$row 2. Record the multiplier $\frac{2}{3}$ in L:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & \frac{2}{3} & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & -0.5 \\ 0 & 0 & \frac{4}{3} \end{bmatrix}$$

We have obtained the matrices from $PA = LU$:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & \frac{2}{3} & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1.5 & -0.5 \\ 0 & 0 & \frac{4}{3} \end{bmatrix}$$

Proceed with solving for x.

Substitutions

$$Ax = b \quad \Longrightarrow \quad PAx = Pb \equiv d$$
$$PA = LU \quad \Longrightarrow \quad LUx = d$$

- Define a new vector $y = Ux$
  - $Ly = b \quad \Longrightarrow \quad Ly = d$
  - Solve for y, forward substitution:

$$y_0 = \frac{d_0}{L_{00}}$$
$$y_i = \frac{d_i - \sum_{j=0}^{i} L_{ij} y_j}{L_{ii}}$$

- Then solve $Ux = y$:
  - Solve for x, backward substitution:

$$x_N = \frac{y_N}{U_{NN}}$$
$$x_i = \frac{y_i - \sum_{j=i+1}^{N} U_{ij} x_j}{U_{ii}}$$

How to use the solver in Python

```python
import numpy as np
from scipy.linalg import lu
from gaussjordan import backsubstitution_v1 as backwardSub
from gaussjordan import forwardsubstitution as forwardSub

# Example usage
A = np.random.rand(5, 5) # Get random matrix
P, L, U = lu(A) # Get L, U and P
b = np.random.rand(5) # Random b vector
d = P @ b # Permute b vector
y = forwardSub(L, d) # Can also do y=L\d
x = backwardSub(U, y) # Can also do x=U\y
rnorm = np.linalg.norm(A @ x - b) # Residual
```

How to use the solver in Python

```python
import numpy as np
from scipy.linalg import lu
from gaussjordan import backsubstitution_v1 as backwardSub
from gaussjordan import forwardsubstitution as forwardSub

# Example usage
A = np.random.rand(5, 5) # Get random matrix
P, L, U = lu(A) # Get L, U and P
b = np.random.rand(5) # Random b vector
d = P @ b # Permute b vector
y = forwardSub(L, d) # Can also do y=L\d
x = backwardSub(U, y) # Can also do x=U\y
rnorm = np.linalg.norm(A @ x - b) # Residual
```

- Use this as a basis to create a function that takes A and b, and returns x.
- Use the function to check the performance for various matrix sizes and inspect the performance.

Today's outline

- Introduction
- Gauss elimination
- Partial Pivoting
- LU decomposition
- Summary

Summary

- This lecture covered direct methods using elimination techniques.
- Gaussian elimination can be slow ($\mathcal{O}(N^3)$)
- Back substitution is often faster ($\mathcal{O}(N^2)$)
- LU decomposition means that we don't have to do Gaussian elimination every time (saves time and effort), but the matrix has to stay the same.
- Python's libraries have built in routines for solving linear equations and LU decomposition.
- Advanced techniques such as (preconditioned) conjugate gradient or biconjugate gradient solvers are also available.
- Next part covers iterative approaches