Solving Systems Of Linear Equations

by Loïc Quertenmont, PhD

LINFO1113 - 2019-2020

Programme

Cours	1	Librairies mathématiques, représentation des nombres en Python et erreurs liées
Cours	**2,3,4**	**Résolution des systèmes linéaires**
Cours	5,6	Interpolation et Régression Linéaires
Cours	7	Zéro d'équation
Cours	8,9	Différentiation numérique
Cours	10	Intégration numérique
Cours	11,12	Introduction à l'optimisation
Cours	13	Rappel / Répétition

Outline

- **Matrix Algebra (Reminder)**
- **Introduction**
- **Gauss Elimination Method**
- **LU Decomposition Methods**
- **Symmetric and Banded Coefficient Matrices**
- **Iterative Methods**
- **Other Remarks**

Real life example

- We food product and we want to find it's recipe
- We know the pizza is made of flour, cheese and tomato,
  - but we don't know in which proportion ?
- We can look at the energy, sugar and fat contained in each product to estimate the quantities

	x 	+	y 	+	z 	= 
Energy	x 6	+	y 4	+	z 2	= 28
Sugar	x 5	+	y 1	+	z 1	= 18
Fat	x 1	+	y 6	+	z 1	= 16

- We only need to solve this system of equations
  - Not always easy… → x=3, y=2, z=1

Example of system of linear equation

- System of linear equations of the form

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3x - y + z &= 4 \end{aligned}$$

- We are looking for a solution to the system
  - Find a value for x, y, and z that solves the system

- That type of system appears naturally in many type of engineering and physical problems

Examples

- **Example in 1-dimension**

$$5\,x = 3 \qquad\qquad a\,x = b$$

Solution is easy to find:

$$x = 3/5 \qquad\qquad x = b/a$$

- **Example in 3 dimensions**

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3x - y + z &= 4 \end{aligned} \qquad\qquad A\,\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$$

- *Solution is already harder to find:*

$$\begin{aligned} x &= 1 \\ y &= 2 \\ z &= 3 \end{aligned} \qquad\qquad \vec{\boldsymbol{x}} = \boldsymbol{A^{-1}}\vec{\boldsymbol{b}}$$

Extending to N-dimensions

- **A system of algebraic equations has the form**

$$A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n = b_1$$
$$A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n = b_2$$
$$\vdots$$
$$A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nn}x_n = b_n$$

where
- A_{ij} are known
- b_i are known
- X_i are the unknowns

- **OR equivalently with matrix notation**

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- **OR simply:**

$$\mathbf{Ax} = \mathbf{b}.$$

Matrix algebra will be heavily used in this lecture,

so let’s take a few minutes to review

Basis of matrix algebra

Matrix Algebra (reminder)

- A matrix is a rectangular array of numbers.
- A matrix of size (or dimension) m x n has m rows and n columns
  - The number of rows is always listed first
- A square matrix has m=n

- A column or a column vector, or simply a vector
  - matrix of dimensions n × 1
- A row or a row vector
  - matrix of dimensions 1 × n

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

- **Notation:**
  - For Matrices: boldfaced uppercase letters
  - For Vectors: boldfaced lowercase letters
  - (For scalar: lowercase letters)

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

  - Indices of the elements of a matrix:
    The row number comes first, followed by the column number
  - Indices of the elements of a vector:
    - only one index is needed

Matrix Operations

Transpose:

- The transpose of a matrix **A** is noted $\mathbf{A}^T$
- Defined as:
  - $\mathbf{A}^T_{ij} = \mathbf{A}_{ji}$
  - Rows and Columns of the matrix are exchanged
    - Column vector → Row vector
- A square matrix is **symmetric** if $\mathbf{A}^T = \mathbf{A}$

Transpose

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{A}^T = \begin{bmatrix} A_{11} & A_{21} & A_{31} \\ A_{12} & A_{22} & A_{32} \\ A_{13} & A_{23} & A_{33} \end{bmatrix} \qquad \mathbf{b}^T = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

Matrix Operations

Addition:

The sum $\mathbf{C} = \mathbf{A} + \mathbf{B}$ of two $m \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ is defined as

$$C_{ij} = A_{ij} + B_{ij}, \quad i = 1, 2, \ldots, m; \quad j = 1, 2, \ldots, n \tag{A10}$$

Thus the elements of $\mathbf{C}$ are obtained by adding elements of $\mathbf{A}$ to the elements of $\mathbf{B}$. Note that addition is defined only for matrices that have the same dimensions.

Matrix Operations

Vector inner (dot) product:

The *dot* or *inner product* $c = \mathbf{a} \cdot \mathbf{b}$ of the vectors $\mathbf{a}$ and $\mathbf{b}$, each of size m, is defined as the scalar

$$c = \sum_{k=1}^{m} a_k b_k \tag{A11}$$

It can also be written in the form $c = \mathbf{a}^T\mathbf{b}$. In NumPy the function for the dot product is `dot(a,b)` or `inner(a,b)`.

The output of an inner/dot product is **a scalar**

Vector outer product:

The *outer product* $\mathbf{C} = \mathbf{a} \otimes \mathbf{b}$ is defined as the matrix

$$C_{ij} = a_i b_j$$

An alternative notation is $\mathbf{C} = \mathbf{a}\mathbf{b}^T$. The NumPy function for the outer product is `outer(a,b)`.

The output of an outer product is **a matrix**

Matrix Operations

Matrix product:

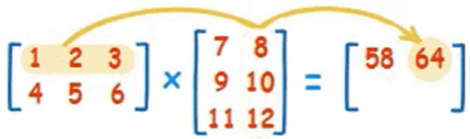
The *matrix product* $\mathbf{C} = \mathbf{AB}$ of an $l \times m$ matrix $\mathbf{A}$ and an $m \times n$ matrix $\mathbf{B}$ is defined by

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}, \quad i = 1, 2, \ldots, l; \quad j = 1, 2, \ldots, n \qquad \text{(A12)}$$

- **The number of columns in A (the dimension m) must be equal to the number of rows in B.**
- **The resulting matrix has dimension : l x n**

- The matrix product can also be defined in terms of the dot product.
  Representing the ith row of **A** as the vector a_i and the jth column of **B** as the vector b_j , we have

$$\mathbf{AB} = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \mathbf{a}_1 \cdot \mathbf{b}_2 & \cdots & \mathbf{a}_1 \cdot \mathbf{b}_n \\ \mathbf{a}_2 \cdot \mathbf{b}_1 & \mathbf{a}_2 \cdot \mathbf{b}_2 & \cdots & \mathbf{a}_2 \cdot \mathbf{b}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_\ell \cdot \mathbf{b}_1 & \mathbf{a}_\ell \cdot \mathbf{b}_2 & \cdots & \mathbf{a}_\ell \cdot \mathbf{b}_n \end{bmatrix}$$



matrix-multiplying ضرب ماتریس ها

Matrix Operations

Note on Numpy Array products

- NumPy **dot product** for arrays:
  - dot(**A**,**B**) → matrix product of **A** and **B**.

$$\mathbf{AB} = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \mathbf{a}_1 \cdot \mathbf{b}_2 & \cdots & \mathbf{a}_1 \cdot \mathbf{b}_n \\ \mathbf{a}_2 \cdot \mathbf{b}_1 & \mathbf{a}_2 \cdot \mathbf{b}_2 & \cdots & \mathbf{a}_2 \cdot \mathbf{b}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_\ell \cdot \mathbf{b}_1 & \mathbf{a}_\ell \cdot \mathbf{b}_2 & \cdots & \mathbf{a}_\ell \cdot \mathbf{b}_n \end{bmatrix}$$

- NumPy **inner product** for arrays:
  - Inner(**A**,**B**) = dot(**A**,$\mathbf{B}^T$) → matrix product of **A** and $\mathbf{B}^T$.

- NumPy **outer product** for arrays **A** (size k × l) and **B** (size m × n)
  Construction similar to matrix product, but with an outer vector product
  outer(**A**,**B**) =

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_1 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_1 \otimes \mathbf{b}_m \\ \mathbf{a}_2 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_2 \otimes \mathbf{b}_m \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_k \otimes \mathbf{b}_1 & \mathbf{a}_k \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_k \otimes \mathbf{b}_m \end{bmatrix}$$

The submatrices $\mathbf{a}_i \otimes \mathbf{b}_j$ are of dimensions $\ell \times n$. As you can see, the size of the outer product is much larger than either **A** or **B**.

Identity and Inverse

Identify Matrix

A square matrix of special importance is the identity or *unit matrix*:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

It has the property $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$.

Inverse Matrix

The inverse of an $n \times n$ matrix $\mathbf{A}$, denoted by $\mathbf{A}^{-1}$, is defined to be an $n \times n$ matrix that has the property

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I} \tag{A16}$$

Determinant

Determinant

The determinant of a square matrix $\mathbf{A}$ is a scalar denoted by $|\mathbf{A}|$ or $\det(\mathbf{A})$. There is no concise definition of the determinant for a matrix of arbitrary size. We start with the determinant of a 2×2 matrix, which is defined as

$$\begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} = A_{11}A_{22} - A_{12}A_{21} \tag{A17}$$

The determinant of a 3×3 matrix is then defined as

$$\begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix} = A_{11} \begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} - A_{12} \begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix} + A_{13} \begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix}$$

Determinant

Having established the pattern, we can now define the determinant of an $n \times n$ matrix in terms of the determinant of an $(n-1) \times (n-1)$ matrix:

$$|\mathbf{A}| = \sum_{k=1}^{n} (-1)^{k+1} A_{1k} M_{1k} \tag{A18}$$

where M_{ik} is the determinant of the $(n-1) \times (n-1)$ matrix obtained by deleting the ith row and kth column of $\mathbf{A}$. The term $(-1)^{k+i} M_{ik}$ is called a *cofactor* of A_{ik}.

$$\det\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = a \bullet \det\begin{bmatrix} e & f \\ h & i \end{bmatrix} - b \bullet \det\begin{bmatrix} d & f \\ g & i \end{bmatrix} + c \bullet \det\begin{bmatrix} d & e \\ g & h \end{bmatrix}$$

The matrix $\mathbf{A}$ is said to be *singular* if $|\mathbf{A}| = 0$.

Positive Definiteness

An $n \times n$ matrix A is said to be positive definite if

$$\mathbf{x}^T\mathbf{A}\mathbf{x} > 0 \tag{A20}$$

for all nonvanishing vectors $\mathbf{x}$. It can be shown that a matrix is positive definite if the determinants of all its leading minors are positive. The leading minors of $\mathbf{A}$ are the n square matrices

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ A_{12} & A_{22} & \cdots & A_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{bmatrix}, \quad k = 1, 2, \ldots, n$$

Therefore, positive definiteness requires that

$$A_{11} > 0, \quad \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} > 0, \quad \begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix} > 0, \ldots, |A| > 0$$

Useful Theorems

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$$

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$$

$$\left|\mathbf{A}^T\right| = |\mathbf{A}|$$

$$|\mathbf{AB}| = |\mathbf{A}|\,|\mathbf{B}|$$

$$\text{if } \mathbf{C} = \mathbf{A}^T\mathbf{BA} \text{ where } \mathbf{B} = \mathbf{B}^T, \text{ then } \mathbf{C} = \mathbf{C}^T$$

Introduction

Notations

- **A system of algebraic equations has the form**

$$A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n = b_1$$
$$A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n = b_2$$
$$\vdots$$
$$A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nn}x_n = b_n$$

where

- A_{ij} are known
- b_i are known
- X_i are the unknowns

- **OR equivalently with matrix notation**

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- **OR simply:**

$$\mathbf{Ax} = \mathbf{b}.$$

Augmented coefficient matrix

- A useful representation of the equations for computational purposes is the **augmented coefficient matrix:**
  - obtained by **adjoining the constant vector b to the coefficient matrix A**

$$\left[\mathbf{A} \mid \mathbf{b}\right] = \left[\begin{array}{cccc|c} A_{11} & A_{12} & \cdots & A_{1n} & b_1 \\ A_{21} & A_{22} & \cdots & A_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{n3} & b_n \end{array}\right]$$

- **This matrix contains all the "knowns" of the system of equations**

Uniqueness of Solution

- For a system of n equations with n unknowns
  - Has a **unique solution**, IF
  - The coefficient matrix is **non-singular**
    → determinant of A is not null
    → $|A| \neq 0$
    → Row and Columns are linearly independent

- If the coefficient matrix is **singular**
  - There might be an **infinite number of solutions or None**
  - **Example:**

$$2x + y = 3$$
$$4x + 2y = 6$$

→ The two equations are equivalent, so any pair (x,y) satisfying the first equation is valid
→ **Number of solution is infinite**

$$2x + y = 3$$
$$4x + 2y = 0$$

→ The two equations contradict each other
→ There is **no solution** satisfying both equations

“ill Conditioning “

- What happens when the coefficient matrix is almost singular ?
  - if $|\mathbf{A}|$ is very small

- **What is “small”…. ?**
  - We need a reference → Norm of the matrix (denoted $\|\mathbf{A}\|$)
  - Then, determinant is small IF $|\mathbf{A}| \ll \|\mathbf{A}\|$
  - Euclidian Norm: $$\|\mathbf{A}\|_e = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}^2}$$
  - Row-Sum Norm: $$\|\mathbf{A}\|_\infty = \max_{1\le i\le n} \sum_{j=1} |A_{ij}|$$

- **Formal Measure of conditioning**:
  - Matrix condition number: $\mathrm{con^d}(\mathbf{A}) = \|\mathbf{A}\|\left\|\mathbf{A^{-1}}\right\|$
  - If $\mathrm{con^d}(\mathbf{A})$ is close to unity, the matrix is well conditioned
  - If $\mathrm{con^d}(\mathbf{A})$ >> 1 the matrix is ill conditionned

“ill Conditioning “

- Remarks on condition number
  - It depends on the choice for the norm
  - It is heavy to calculate on large matrices

- In most cases it is sufficient to gauge conditioning by comparing the determinant with the magnitudes of the elements in the matrix

- If the equations are ill conditioned,
  small changes in the coefficient matrix → large changes in the solution

“ill Conditioning “

$$2x + y = 3 \qquad 2x + 1.001y = 0 \quad \rightarrow \quad x = 1501.5,\ y = -3000.$$

$$|\mathbf{A}| = 2(1.001) - 2(1) = 0.002 \quad \rightarrow \ |A| << \text{coefficients (1)}$$

$$2x + y = 3 \qquad 2x + 1.002y = 0 \quad \rightarrow \quad x = 751.5,\ y = -1500.$$

Note that a 0.1% change in the coefficient of y produced a 100% change in the solution!

- Numerical solutions of ill-conditioned equations are not to be trusted.
- Roundoff errors during the solution process are equivalent to small changes in the coefficient matrix.
- This in turn introduces large errors into the solution
- In suspect cases the determinant of the coefficient matrix should be computed so that the degree of ill conditioning can be estimated.
  - This can be done during or after the solution with only a small computational effort.

Solving Linear Systems

Two classes of methods for solving linear systems:

- **Direct Methods:**
  - Transform the original equations into equivalent equations easier to solve
  - The transformation do not change the solution, but could change **|A|**
  - Transformations :
    - Exchanging two equations
      $|A| \rightarrow -|A|$
    - Multiplying an equation by a nonzero constant c
      $|A| \rightarrow c|A|$
    - Multiplying an equation by a nonzero constant
      and then subtracting it from another equation
      $|A| \rightarrow |A|$
- **Indirect (or iterative) Methods**
  - Start with a guess of the solution **x** and then repeatedly
    refine the solution until a certain convergence criterion is reached
  - **Generally slower, but could be faster if A is large or sparse**

Overview of Direct Methods

Method	Initial form	Final form
Gauss elimination	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{Ux} = \mathbf{c}$
LU decomposition	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{LUx} = \mathbf{b}$
Gauss-Jordan elimination	$\mathbf{Ax} = \mathbf{b}$	$\mathbf{Ix} = \mathbf{c}$

Table 2.1. Three Popular Direct Methods

$$\mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \qquad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

L and **U** are triangular matrices:
Solving a system once it has a triangular form is easy
Example with the form **L x = b**

$$\begin{aligned} L_{11}x_1 &= c_1 & x_1 &= c_1/L_{11} \\ L_{21}x_1 + L_{22}x_2 &= c_2 \quad \rightarrow & x_2 &= (c_2 - L_{21}x_1)/L_{22} \\ L_{31}x_1 + L_{32}x_2 + L_{33}x_3 &= c_3 & x_3 &= (c_3 - L_{31}x_1 - L_{32}x_2)/L_{33} \end{aligned}$$

→ Forward substitution

Similarly, we can solve system of the form **U x = b** with backward substitution

Gauss Elimination Method

Gauss Elimination

- Given the system of linear equation:

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ 3x - y + z &= 4 \end{aligned} \quad \leftrightarrow \quad \begin{pmatrix} 1 & 1 & -2 \\ 0 & 1 & -1 \\ 3 & -1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -3 \\ -1 \\ 4 \end{pmatrix}$$

augmented coeficient matrix

$$\left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 3 & -1 & 1 & 4 \end{array}\right)$$

We can solve a system of linear equations by doing a sequence of row manipulations.

Gauss Elimination

Multiply first eq. with -3 and add to third eq.

$$x + y - 2z = -3 \quad \times -3$$
$$y - z = -1$$
$$3x - y + z = 4 \quad +$$

$$\left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 3 & -1 & 1 & 4 \end{array}\right)$$

Multiply second eq. with 4 and add to third eq.

$$x + y - 2z = -3$$
$$y - z = -1 \quad \times 4$$
$$-4y + 7z = 13 \quad +$$

$$\left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & -4 & 7 & 13 \end{array}\right)$$

Multiply third eq. with $\frac{1}{3}$ to obtain z

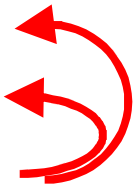
$$x + y - 2z = -3$$
$$y - z = -1$$
$$3z = 9 \quad \times 1/3$$

$$\left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 3 & 9 \end{array}\right)$$

Backward substitution

Substitute $z = 3$ in first and second equation

$$\begin{aligned} x + y - 2z &= -3 \\ y - z &= -1 \\ z &= 3 \end{aligned}$$



$$\left(\begin{array}{ccc|c} 1 & 1 & -2 & -3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Substitute $y = 2$ in first equation

$$\begin{aligned} x + y &= 3 \\ y &= 2 \\ z &= 3 \end{aligned}$$

$$\left(\begin{array}{ccc|c} 1 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

The solution:

$$\begin{aligned} x &= 1 \\ y &= 2 \\ z &= 3 \end{aligned}$$

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array}\right)$$

Algorithm

- To solve $AX = B$ start with the matrix $(A|B)$
  1. We will do forward substitutions until we get a *upper triangular* matrix:

$$\left(\begin{array}{ccc|c} 1 & \times & \times & \times \\ 0 & 1 & \times & \times \\ 0 & 0 & 1 & \times \end{array}\right)$$

  2. Then we do backward substitutions to get

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & \times \\ 0 & 1 & 0 & \times \\ 0 & 0 & 1 & \times \end{array}\right)$$

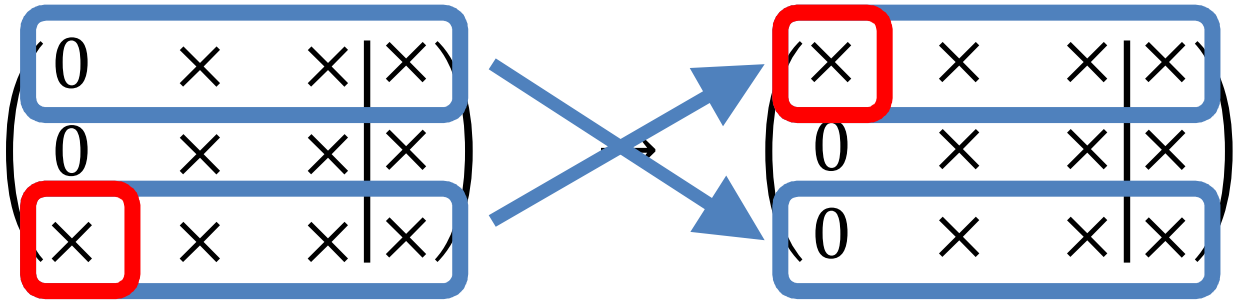
- We assume that A is $n \times n$ and the solution x exists

Gauss Elimination Algorithm – forward (1)

$$(A|B) = \left(\begin{array}{ccc|c} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{array}\right)$$

Step 1: Choose the first element as *pivot* element

If the first element is 0, first find another row with non-zero pivot and permute the rows



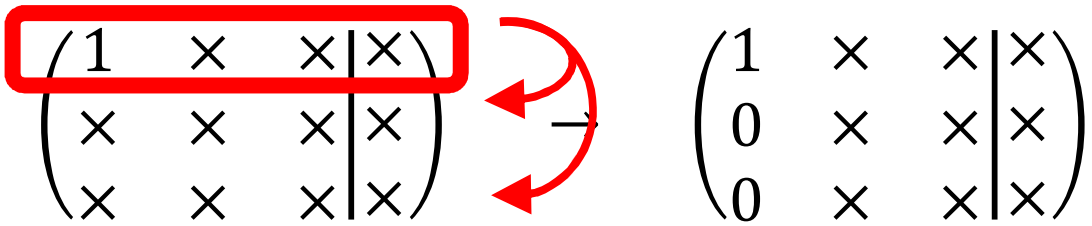
Gauss Elimination Algorithm – forward (2)

Step 2: Divide the first line by the pivot element

$$\left(\begin{array}{ccc|c} a & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & \times/a & \times/a & \times/a \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{array}\right)$$

Gauss Elimination Algorithm – forward (3)

Step 3: Use the row to eliminate the other values below the pivot in the same column

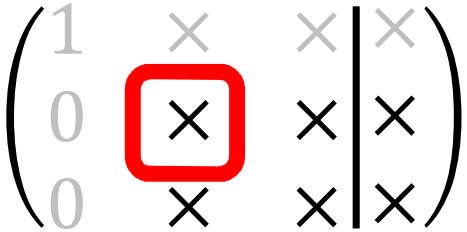


Example:

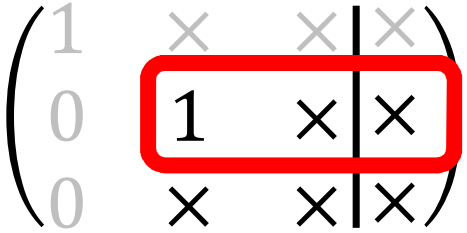
$$\left(\begin{array}{ccc|c} 1 & a & b & c \\ d & e & f & g \\ h & i & j & k \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & a & b & c \\ 0 & e-da & f-db & g-dc \\ 0 & i-ha & j-hb & k-hc \end{array}\right)$$

Gauss Elimination Algorithm – forward (4)

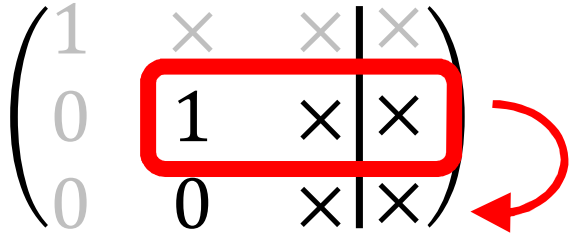
- We are finished with row 1 and column 1
- Now do the same with the second row/column:
  - Step 1: choose pivot



  - Step 2: divide the row by the pivot

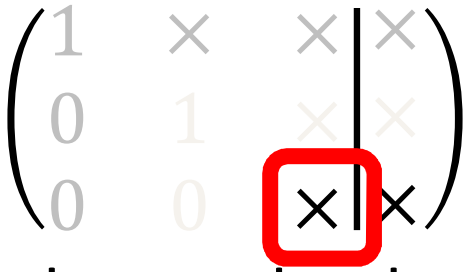


  - Step 3: eliminate values below pivot

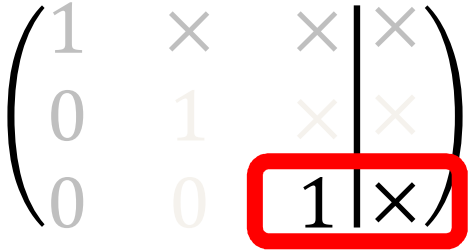


Gauss Elimination Algorithm – forward (5)

- Repeat forward substitutions for all remaining rows and columns
  - Step 1: choose pivot



  - Step 2: divide the row by the pivot



  - Step 3: eliminate values below pivot.
    - Nothing to do here because we are already in the last row. We are finished.

Gauss Elimination Algorithm – forward (6)

- Final result of forward substitutions: A upper triangular matrix of the form

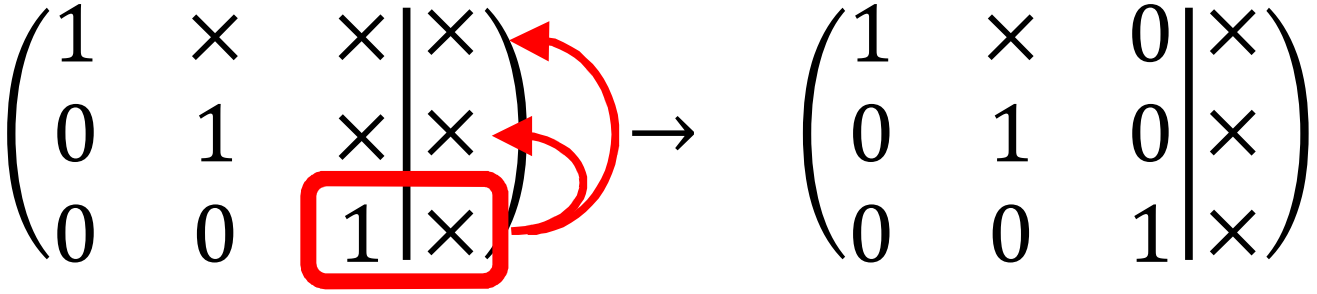
$$\left(\begin{array}{ccc|c} 1 & \times & \times & \times \\ 0 & 1 & \times & \times \\ 0 & 0 & 1 & \times \end{array}\right)$$

- If we decompose the augmented coeficient matrix, we get a simplified problem of the form:

$$\mathbf{U\,x = b'}$$

Algorithm – Back substitution

- Use the last row to eliminate the elements in the rows above it

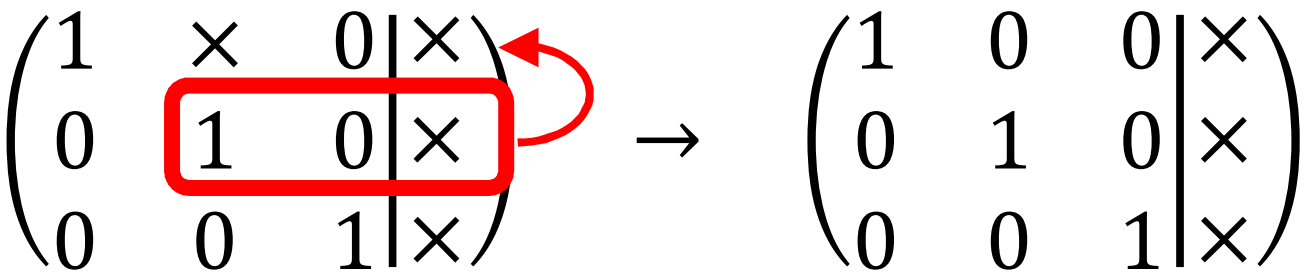


Example:

$$\left(\begin{array}{ccc|c} 1 & a & b & c \\ 0 & 1 & d & e \\ 0 & 0 & 1 & f \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & a & 0 & c-bf \\ 0 & 1 & 0 & e-df \\ 0 & 0 & 1 & f \end{array}\right)$$

Back substitution (2)

- Do the same with the row above the last row:



- Continue with all rows until the first row has been reached. Final result: Form $(I \mid \vdots)$

Python Implementation

```
import numpy as np

def gaussElimin(a,b):
    n = len(b)
  # Elimination Phase
    for k in range(0,n-1): #iterate on rows
        for i in range(k+1,n): #iterate on rows below pivot
           if a[i,k] != 0.0:
               lam = a [i,k]/a[k,k]
               a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
               b[i] = b[i] - lam*b[k]
  # Back substitution
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
    return b
```

```
A = np.array([[1,1,-2],[0,1,-1],[3,-1,1]])
display(A)

b = np.array([-3,-1,4])
display(b)

gaussElimin(A,b)
```

executed in 19ms, finished 14:44:24 2019-09-06

→

```
array([[ 1,  1, -2],
       [ 0,  1, -1],
       [ 3, -1,  1]])

array([-3, -1,  4])

array([1, 2, 3])
```

Complexity of Gaussian Elimination

- We have three types of operation
  1. Permuting rows
  2. Multiplying a row by its pivot element
  3. Eliminating a row
- For a $n \times n$ matrix, each operation takes $\rightarrow O(n)$

- Each forward transformation changes all rows below the current row $\rightarrow O(n^2)$
- Each back substitution changes a column $\rightarrow O(n)$
- n forward and backward substitutions $\rightarrow O(n^3)$

Gauss-Jordan Elimination

- Gauss elimination pushed to its limit

- **Gauss Elimination**
  - Transform only the rows bellow the pivot
  - Output of elimination → **U x = c**
  - Complexity →$\sim n^3 \ /3$
- **Gauss-Jordan Elimination**
  - Transform rows bellow and above the pivot
  - Output of elimination → **I x = c**
  - Complexity → $\sim n^3 \ /2$

Numerical Stability

- Let's take this system

$$\left(\begin{array}{cc|c} a & 1 & \times \\ 1 & 0 & \times \end{array}\right)$$

- We have seen that with floating-point values sometimes calculations are not exact. Let's assume a is very close to zero but not exactly zero: $0 < a \ll 1$
- Forward substitution with a as pivot does this:

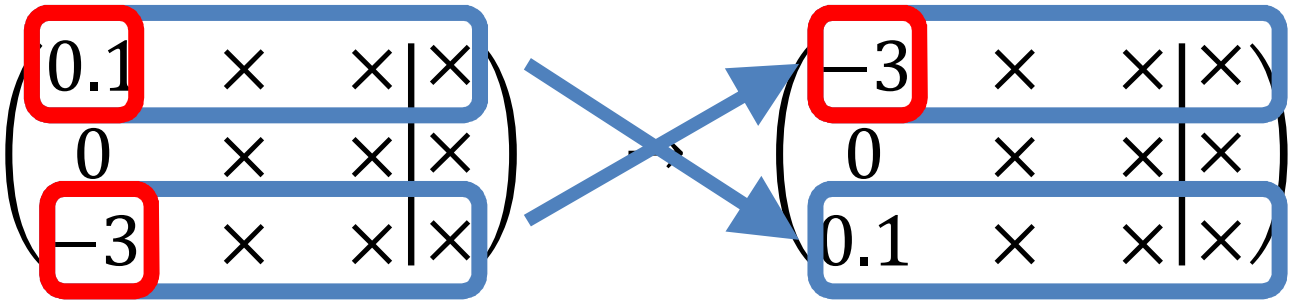
$$\left(\begin{array}{cc|c} 1 & 1/a & \times \\ 0 & -1/a & \times \end{array}\right)$$

  Now we have two huge numbers $-1/a$ and $1/a$!
  Small errors $\mathrm{a} + \varepsilon$ will result in large differences

- Good pivot = larger (positive or negative) number

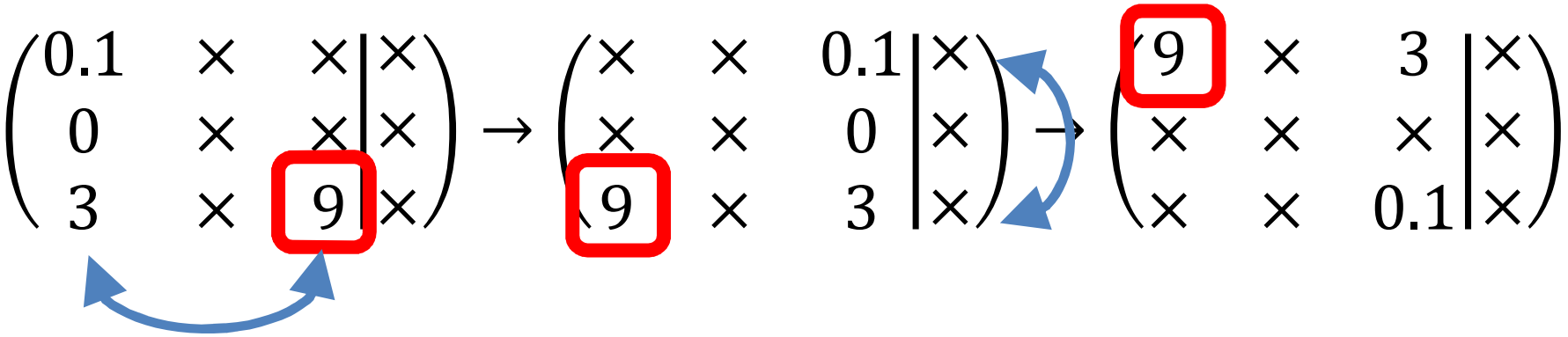
Numerical Stability (2)

- How to get a large pivot?
- Two methods: *Partial Pivoting* and *Full Pivoting*
- Partial Pivoting:
  - That's what we already do in forward substitution
  - Search the column to find other pivots. Take the row with the largest absolute value and permute.



Full Pivoting

- Full Pivoting
  - Search the entire matrix for the largest possible pivot and permute rows and/or columns



- Full Pivoting gives better results, but is slower
- Permuting columns changes the order of the variables! We have to remember them.

Solving Two Problems Simultaneously

- Note that the steps in the Gaussian Elimination for $\mathbf{AX} = \mathbf{B}$ only depend on the elements of $\mathbf{A}$, not on $\mathbf{B}$!
- That means that two problems

$$\mathbf{AX_1} = \mathbf{B_1} \quad \text{and} \quad \mathbf{AX_2} = \mathbf{B_2}$$

need the same manipulation steps on $(\mathbf{A} \mid \mathbf{B_1})$ and $(\mathbf{A} \mid \mathbf{B_2})$ to find $\mathbf{X_1}$ and $\mathbf{X_2}$

- We can solve them both at the same time by working with $(\mathbf{A} \mid \mathbf{B_1}\ \mathrm{B}_2)$

Solving n Problems Simultaneously

- Let's solve these special linear systems:

$$\mathbf{AX_1} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \mathbf{AX_2} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \cdots \quad \mathbf{AX_n} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

- We can solve them together by Gaussian Elimination on

$$\left(\begin{array}{cccc|cccc} \times & \times & \times & \times & 1 & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 1 & 0 & 0 \\ \times & \times & \times & \times & \vdots & \vdots & \ddots & \vdots \\ \times & \times & \times & \times & 0 & 0 & 0 & 1 \end{array}\right)$$

$\mathbf{A}$

Solving n Problems Simultaneously (2)

$$\mathbf{AX_1} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \mathbf{AX_2} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \cdots \quad \mathbf{AX_n} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

can be written as

$$\mathbf{A} \begin{pmatrix} | & & | \\ \mathbf{X_1} & \cdots & \mathbf{X_n} \\ | & & | \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

What are we computing here?

$$\mathbf{A}\,\mathbf{A^{-1}} = \mathbf{I}$$

A Remark on Inverse Matrices

- Calculating $\mathbf{A^{-1}}$ is rarely needed and should be avoided because of numerical instabilities

- In many problems, inverse matrices appear in equations of the form $\mathbf{C = A^{-1}B}$

- Instead of inverting A we find C by solving

$$\mathbf{A\,C = B}$$