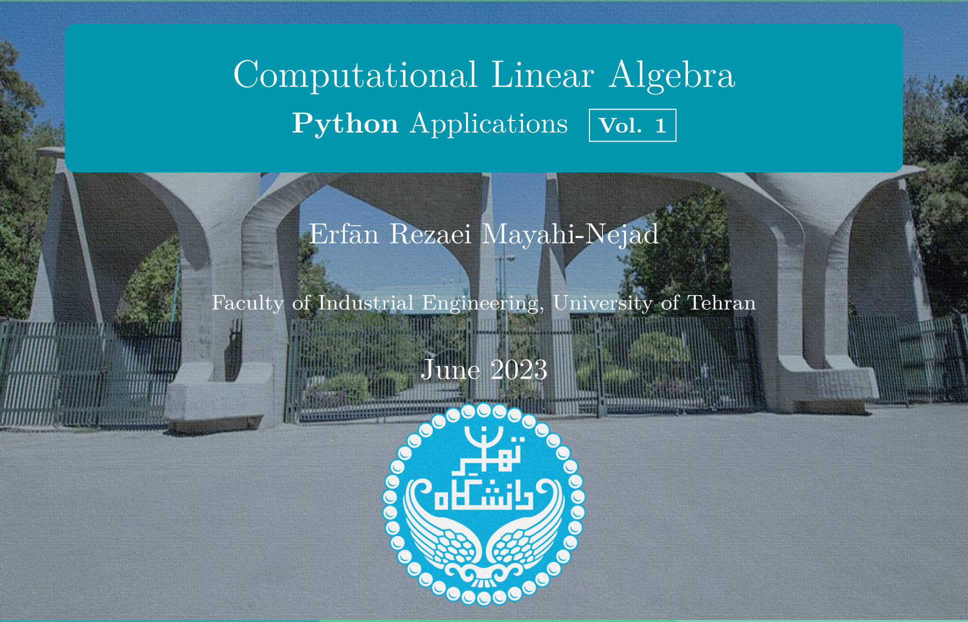


Computational Linear Algebra

Python Applications **Vol. 1**

Erfān Rezaei Mayahi-Nejad

Faculty of Industrial Engineering, University of Tehran

June 2023

What is This Course All About?

- This course is an **extension** of the *Linear Algebra* course taught by Dr. Rahbari at the Faculty of Industrial Engineering at the University of Tehran during the 2023 academic year.

- The main goal of this course is to acquaint students with some widely-used libraries in Python programming, such as SymPy, NumPy, SciPy, and others.

- In this course, the Jupyter Notebook IDE is utilized through Anaconda.

How to Import & Use Libraries?

Before using a library, it is necessary to import it into the Notebook.

```
# !pip install (library name)
# import (library name)
# for instance:

!pip install sympy
import sympy
```

The library names can also be abbreviated.

```
# import (library name) as (abbr. name)
# for instance:

import numpy as np
import sympy as sp
import scipy as sc
```

How to Import & Use Libraries? (cont'd)

A more efficient way to use functions is demonstrated below by abbreviating the library name to another arbitrary name.

```
# first form

import numpy

numpy.arange(1,10,2)
↪ array([1, 3, 5, 7, 9])
```

```
# second form

import numpy as np

np.arange(1,10,2)
↪ array([1, 3, 5, 7, 9])
```

Vectors & Matrices in SymPy

```python
import sympy as sp
```

$$\begin{bmatrix} 1 & 3 & 0 \\ 0 & -1 & 2 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ -1 \end{bmatrix}$$

```python
# this will be shown as a vector (Constant Matrix)
b=sp.Matrix([[3],[2],[-1]]); display(b)
↪ Matrix([[3], [2], [-1]])
```

```python
# this will be shown as a matrix (Coefficient Matrix)
A=sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]); display(A)
↪ Matrix([[1, 3, 0], [0, -1, 2], [-4, 0, 1]])
```

```python
# this will be shown as a vector (Variable Matrix)
x,y,z=sp.symbols('x y z')
X=sp.Matrix([[x],[y],[z]]); display(X)
↪ Matrix([[x], [y], [z]])
```

Arithmetic Operators in SymPy

```
# addition
display(b+b, A+A)
↪ Matrix([[6], [4], [-2]])
  Matrix([[2,6,0], [0,-2,4], [-8,0,2]])
```

```
# subtraction
sp.Matrix([[4],[-5],[2]])-b
↪ Matrix([[1], [-7], [3]])
```

```
# multiplication
# this also could be done by: A.multiply(X)
A*X
↪ Matrix([[x+3y], [-y+2z], [-4x+z]])
```

```
# elementwise multiplication
X.multiply_elementwise(b)
↪ Matrix([[3x], [2y], [-z]])
```

Special Matrices in SymPy

(1) Identity Matrix:

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

```
sp.eye(3)      # 3 shows the dimention here
↪ Matrix([[1,0,0], [0,1,0], [0,0,1]])
```

Special Matrices in SymPy (cont'd)

(2) Zero Matrix:

$$\mathbf{0}_{n\times m} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}_{n\times m}$$

```
sp.zeros(3,2)      # 3 is no. rows, and 2 is no. columns
↪ Matrix([[0,0], [0,0], [0,0]])
```

Special Matrices in SymPy (cont'd)

(3) All-Ones Matrix:

$$\mathbf{J}_{n\times m} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}_{n\times m}$$

```
sp.ones(2,3)     # 2 is no. rows, and 3 is no. columns
↪ Matrix([[1,1,1], [1,1,1]])
```

Special Matrices in SymPy (cont'd)

(4) Diagonal Matrix:

$$\mathbf{D}_n = \begin{bmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \end{bmatrix}_{n\times n}$$

```
x,y,z=sp.symbols('x y z')

sp.diag(x,y,z)
↪ Matrix([[x,0,0], [0,y,0],[0,0,z]])
```

Special Matrices in SymPy (cont'd)

(5) Random-Valued Matrix:

$$\mathbf{R}_{n\times m} = \begin{bmatrix} \mathtt{A} & \alpha & \cdots & \aleph \\ \text{♌} & \text{👤} & \cdots & \text{🦇} \\ \vdots & \vdots & \ddots & \vdots \\ \text{♅} & \text{☯} & \cdots & \mathfrak{z} \end{bmatrix}_{n\times m}$$

```
sp.randMatrix(2,4,1,10)
↪ Matrix([[4,9,5,4], [4,10,1,4]])
```

1. Linear Systems: Inverse Matrix

Trace

The trace of a square matrix is the sum of its diagonal elements.

$$tr(\mathbf{A}_n) = \sum_{i=1}^{n} a_{ii}$$

```
sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).trace()
↪ 1
```

1. Linear Systems: Inverse Matrix (cont'd)

Determinant

The determinant of a square matrix is a single number. That number contains an amazing amount of information about the matrix.

$$det(\mathbf{A}_n) = |\mathbf{A}_n| = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

```
sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).det()
↪ -25
```

1. Linear Systems: Inverse Matrix (cont'd)

Transpose

A matrix is transposed when elements of rows and columns are switched interchangeably.

$$\mathbf{A}_{m\times n} = \mathbf{A}^{\mathrm{T}}_{n\times m} = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \dots & a_{mn} \end{bmatrix}$$

```
sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).T
↪ Matrix([[1,0,-4],[3,-1,0],[0,2,1]])
```

1. Linear Systems: Inverse Matrix (cont'd)

Cofactor Matrix

The cofactor matrix, also known as the matrix of cofactors, is a square matrix obtained from a given square matrix by calculating the cofactor of each element. The cofactor of an element is determined by taking the determinant of the submatrix obtained by removing the row and column containing that element and multiplying it by a sign factor.

```
sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).cofactor_matrix()
↪ Matrix([[-1,-8,-4],[-3,1,-12],[6,-2,-1]])
```

1. Linear Systems: Inverse Matrix (cont'd)

Adjugate Matrix

The adjugate matrix, also known as the classical adjoint matrix, is a square matrix associated with another square matrix.

$$cofactor(\mathbf{A})^{\mathrm{T}} = adjugate(\mathbf{A}_n) = \begin{bmatrix} \mathrm{M}_{11} & \mathrm{M}_{21} & \dots & \mathrm{M}_{n1} \\ \mathrm{M}_{12} & \mathrm{M}_{22} & \dots & \mathrm{M}_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{M}_{1m} & \mathrm{M}_{2m} & \dots & \mathrm{M}_{mn} \end{bmatrix}$$

```
sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).adjugate()
↪ Matrix([[-1,-3,6],[-8,1,-2],[-4,-12,-1]])
```

1. Linear Systems: Inverse Matrix (cont'd)

Inverse Matrix

The inverse matrix, also known as the multiplicative inverse or reciprocal matrix, is a concept in linear algebra that is closely related to square matrices. Given a square matrix $\mathbf{A}$, if there exists another square matrix $\mathbf{B}$ such that the product of $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{I}$, then $\mathbf{B}$ is said to be the inverse of $\mathbf{A}$.

$$\mathbf{A}_n^{-1} = \frac{adjugate(\mathbf{A}_n)}{|\mathbf{A}_n|}$$

```
sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).inv()
↪ Matrix([[1,3,-6],[8,-1,2],[4,12,1]])/25
```

2. Linear Systems: Gauss-Jordan Elimination

- Row Echelon Form:

$$\mathrm{REF}(\mathbf{A}_{n\times m}) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ 0 & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nm} \end{bmatrix}$$

```
A=sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]])

A.echelon_form()
↪ Matrix([[1, 3, 0],[0, -1, 2],[0,0,-25]])
```

2. Linear Systems: Gauss-Jordan Elimination (cont'd)

Gauss-Jordan Method

Gauss-Jordan elimination is a method used to solve systems of linear equations and perform row reduction on matrices. It is an extension of the Gauss elimination method and aims to bring a given matrix to its reduced row-echelon form.

$$\left[\ \mathbf{A}_{n\times m}\ \middle|\ \mathbf{b}\ \right] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1m} & b_1 \\ a_{21} & a_{22} & \dots & a_{2m} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} & b_n \end{array}\right]$$

```
sp.Matrix([[1,3,0,3],[0,-1,2,2],[-4,0,1,-1]]).rref()
↪ Matrix([[1, 0, 0, 3/5],[0, 1, 0, 4/5],[0, 0, 1, 7/5]]),(0, 1, 2)
```

3. Linear Systems: LU Decomposition

LU Method

LU decomposition, also known as LU factorization, is a matrix factorization method used in numerical linear algebra. It decomposes a square matrix into the product of two matrices: an upper triangular matrix (**U**), and a lower triangular matrix (**L**).

$$\mathbf{A}_n = \mathbf{L}_n\mathbf{U}_n = \underbrace{\begin{bmatrix} 1 & 0 & \dots & 0 \\ \mathtt{L}_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathtt{L}_{n1} & \mathtt{L}_{n2} & \dots & 1 \end{bmatrix}}_{\mathbf{L}_n} \underbrace{\begin{bmatrix} \mathtt{U}_{11} & \mathtt{U}_{12} & \dots & \mathtt{U}_{1n} \\ 0 & \mathtt{U}_{22} & \dots & \mathtt{U}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathtt{U}_{nn} \end{bmatrix}}_{\mathbf{U}_n}$$

```
L, U, _=sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).LUdecomposition()
display(L,U)
sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]]).LUsolve(sp.Matrix
    ([[3],[2],[-1]]))
↪ Matrix([[1,0,0],[0,1,0],[-4,-12,1]]),Matrix([[1,3,0],[0,-1,2],[0,0,25]])
↪ Matrix([[3/5],[4/5],[7/5]])
```

Row Space of a Matrix

Row Space

The row space of a matrix refers to the vector space spanned by the rows of the matrix. It represents all possible linear combinations of the rows.

```
A=sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]])

A.rowspace()
↪ [Matrix([[1, 3, 0]]),Matrix([[0,-1,2]]),Matrix([[0, 0, -25]])]
```

Column Space of a Matrix

Column Space

The column space of a matrix refers to the vector space spanned by the columns of the matrix. It represents all possible linear combinations of the columns.

```
A=sp.Matrix([[1,3,0],[0,-1,2],[-4,0,1]])

A.columnspace()
↪ [Matrix([[1],[0],[-4]]),Matrix([[3],[-1],[0]]),Matrix([[0],[2],[1]])]
```

Null Space of a Matrix

Null Space

The null space of a matrix, also known as the kernel, is the set of all vectors that, when multiplied by the matrix, result in the zero vector. In other words, it is the solution space of the homogeneous equation $\mathbf{A}\mathbf{x} = 0$, where $\mathbf{A}$ is the matrix and $\mathbf{x}$ is the vector of unknowns.

```
sp.Matrix([[1,3,0,3],[0,-1,2,2],[-4,0,1,-1]]).nullspace()
↪ [Matrix([[-3/5],[-4/5],[-7/5],[1]])]
```

Left Null Space of a Matrix

Left Null Space

The left null space of a matrix is the set of all vectors that, when multiplied by the transpose of the matrix, result in the zero vector. In other words, it is the solution space of the homogeneous equation $\mathbf{A}^{\mathrm{T}}\mathbf{y} = 0$, where $\mathbf{A}^{\mathrm{T}}$ is the transpose of the matrix and $\mathbf{y}$ is the vector of unknowns.

```
sp.Matrix([[1,3,0,3],[0,-1,2,2],[-4,0,1,-1]]).T.nullspace()
↪ []
```

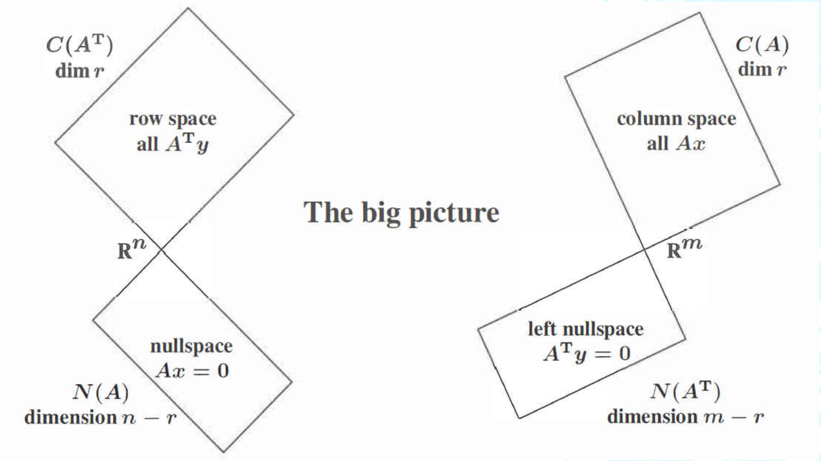
Rank of a Matrix

Rank

The rank of a matrix is the maximum number of linearly independent rows or columns in the matrix. It represents the dimension of the vector space spanned by the rows or columns of the matrix. In other words, it is the number of non-zero rows or columns in the matrix's row or column echelon form.

```
sp.Matrix([[1,3,0,3],[0,-1,2,2],[-4,0,1,-1]]).rank()
↪ 3
```

The Big Picture



Gilbert Strang, *Introduction to Linear Algebra*, 2016

Eigenvalues in SymPy

Eigenvalues

An eigenvalue of a square matrix is a scalar λ that, when multiplied by a corresponding non-zero vector $\mathbf{v}$, yields the equation $\mathbf{Av} = \lambda\mathbf{v}$. Also:

$$\begin{cases} det(\mathbf{A}_n) = \prod_{i=1}^{n} \lambda_i \\ tr(\mathbf{A}_n) = \sum_{i=1}^{n} \lambda_i \end{cases}$$

```
A=sp.Matrix([[0, 1, 1], [1, 0, 0],[1, 1, 1]])
A.eigenvals()
↪ {2: 1, -1: 1, 0: 1}
```

Eigenvectors in SymPy

Eigenvectors

Eigenvectors are non-zero vectors that, when multiplied by a square matrix, result in a scaled version of themselves. In other words, for a square matrix $\mathbf{A}$ and an eigenvector $\mathbf{v}$, the equation $\mathbf{Av} = \lambda\mathbf{v}$ holds, where λ is the corresponding eigenvalue.

```
A=sp.Matrix([[0, 1, 1], [1, 0, 0],[1, 1, 1]])
A.eigenvects()
↪ [([Matrix([[-1],[1],[0]])]),([Matrix([[0],[-1],[1]])]),([Matrix([[2/3],[1/3],[1]])])]
```

Diagonalization in SymPy

Diagonalization

Diagonalization is the process of finding a diagonal matrix Λ that is similar to a given square matrix $\mathbf{A}$. This means that there exists an invertible matrix $\mathbf{X}$ such that $\mathbf{X}^{-1}\mathbf{A}\mathbf{X} = \Lambda$, where Λ is a diagonal matrix.

$$\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

```
A=sp.Matrix([[0, 1, 1], [1, 0, 0],[1, 1, 1]])
A.diagonalize()
↪ (Matrix([[-1,0,2],[1,-1,1],[0,1,3]]),Matrix([[-1,0,0],[0,0,0],[0,0,2]]))
```

Matrix Definiteness in SymPy

Matrix Definiteness

Matrix definiteness refers to the properties of a square matrix based on the signs of its eigenvalues.

Positive Definite All eigenvalues are positive.	**Negative Definite** All eigenvalues are negative.
Positive Semidefinite All eigenvalues are non-negative.	**Negative Semidefinite** All eigenvalues are non-positive.
Indefinite Both positive and negative eigenvalues exist.	

Matrix Definiteness in SymPy (cont'd)

```
A=sp.Matrix([[0,1,1],[1,0,0],[1,1,1]])
A.is_positive_definite
↪ False
```

```
A=sp.Matrix([[0,1,1],[1,0,0],[1,1,1]])
A.is_positive_semidefinite
↪ False
```

```
A=sp.Matrix([[0,1,1],[1,0,0],[1,1,1]])
A.is_negative_definite
↪ False
```

```
A=sp.Matrix([[0,1,1],[1,0,0],[1,1,1]])
A.is_negative_semidefinite
↪ False
```

```
A=sp.Matrix([[0,1,1],[1,0,0],[1,1,1]])
A.is_indefinite
↪ True
```

Jordan Form in SymPy

Jordan Form

The Jordan form, also known as the Jordan canonical form, is a canonical representation of a square matrix that reveals its underlying structure and eigenvalues. It is particularly useful for understanding and analyzing the behavior of linear transformations, especially in the context of eigenvalues and eigenvectors.

$$\mathbf{J}_1 \xrightarrow{\text{Jordan Block}} \begin{bmatrix}\lambda_1 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_1\end{bmatrix}$$

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{B} = \begin{bmatrix}\mathbf{J}_1 & & \\ & \ddots & \\ & & \mathbf{J}_n\end{bmatrix}$$

Jordan Form in SymPy (cont'd)

```
A=sp.Matrix([[0,1,1],[1,0,0],[1,1,1]])
A.jordan_form()
↪ (Matrix([[-1,0,2/3],[1,-1,1/3],[0,1,1]]),Matrix([[-1,0,0],[0,0,0],[0,0,2]]))
```

Matrix Manipulation in SymPy

Coming Soon :)

Thank you for your attention!