Math340: Programming in Mathematics

Instructor: Dr. Bo-Wen Shen

Email: sdsu.math340.shen@gmail.com

Web: http://bwshen.sdsu.edu

Lecture #12a:

Symbolic Mathematics in Python (SymPy) for Computational Algebra

Department of Mathematics and Statistics

San Diego State University

Spring 2025

an example for ill-conditioning systems

Ill-Conditioning: An Example

An Ill-Conditioned System

condition #: 2.0001E4 (20,001)

You may verify that the system

$$0.9999x - 1.0001y = 1$$
$$x - y = 1$$

The solution is: $(x, y) = (0.5, -0.5)$

has the solution $x = 0.5, y = -0.5$, whereas the system

$$0.9999x - 1.0001y = 1$$
$$x - y = 1 + \epsilon$$

has the solution $x = 0.5 + 5000.5\epsilon, y = -0.5 + 4999.5\epsilon$. This shows that the system is ill-conditioned

The solution is: $(x, y) = (0.5, -0.5) + (5000.5\epsilon, 4999.5\epsilon)$

$\epsilon = 1e - 4$ can change the answer significantly. (Kreyszig, 2011)

Basic Python Functions for Linear Algebra

Basic Linear Algebra

NumPy contains the **linalg** module for:

- Solving linear systems
- Computing the determinant of a matrix
- Computing the inverse of a matrix
- Computing eigenvalues and eigenvectors of a matrix
- Solving least-squares problems
- Computing the singular value decomposition of a matrix
- Computing the Cholesky decomposition of a matrix

Conaider the following system, which is ill-conditioned:

$$0.9999x - 1.0001y = 1,$$

$$x - y = 1.$$

Express the above system in a matrix form:

$$A = \begin{bmatrix} 0.9999 & -1.0001 \\ 1 & 1 \end{bmatrix}; \quad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$A\vec{x} = b; \quad \vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Solving the above yields the following:

$$\vec{x} = A^{-1}b$$

$$\vec{x} = (x, y) = (0.5, -0.5)$$

Linear Algebra (LA) in numpy and sympy

np.linalg.solve

sympy.solve_linear_system

In [1]:
```
import numpy as np
A = np.array([[0.9999, -1.0001], [1, -1]])
b = np.array([1, 1])
x = np.linalg.solve(A, b)
print(x)
```

```
[ 0.5 -0.5]
```

What does the condition number tell us?

- A large condition number indicates that the matrix is ill-conditioned , meaning small changes in the input can lead to large changes in the output. This can cause problems in numerical computations, such as solving linear systems or inverting matrices.
- A small condition number indicates that the matrix is well-conditioned, meaning it is more stable for numerical operations.

Ill-conditioning, Sensitivities, & Hallucination

- A large condition number indicates ill-conditioning, which implies (numerical) sensitivity.
- The condition number $\kappa(C)$ is is defined in terms of norm of the matrix C:

$$\kappa(C) = \|C\|\|C^{-1}\|$$

here $\|C\|$ represents the matrix form of C. It can be written as follows:

$$\kappa(C) = \left|\frac{\lambda_{max}}{\lambda_{min}}\right|$$

λ: eigenvalue

- Based on the analysis of conduction numbers, Shen et al. (2022a) showed that the Lorenz 1969 multiscale system is ill-conditioning. Such a feature is referred to as the 3rd kind of butterfly effect.
- Since the condition number is defined as the ratio of the largest to the smallest eigenvalue, systems with very large matrices (such as the attention matrix) may have a wide range of eigenvalues, leading to ill-conditioning.
- We hypothesize that ill-conditioning may occur in large language models, potentially contributing to hallucinations.

In [2]:
```
#import numpy.linalg as LA
eigen=np.linalg.eig(A)
eigen_val=eigen[0]
print(eigen_val)
print(np.linalg.cond(A))
```

```
[-5.e-05+0.01414205j -5.e-05-0.01414205j]
20000.000049995837
```

In [3]:
```
U, S, Vh = np.linalg.svd(A)
print(S)
print(S[0]/S[1])  #lagest singular value/smallest singular value
```

```
[2.00000000e+00 9.99999999e-05]
20000.000049995837
```

In [4]:
```
from sympy import Matrix, Symbol, solve_linear_system
x = Symbol('x')
y = Symbol('y')


system = Matrix (([0.9999, -1.0001,  1.0],\
                  [ 1.0,   -1.0,     1.0]))\


sol=solve_linear_system(system, x, y)

print (sol)
```

```
{x: 0.500000000000000, y: -0.500000000000000}
```

Now, we conaider the following system with a tiny perturl

$$0.9999x - 1.0001y = 1$$

$$x - y = 1 + \epsilon$$

Once again, we express the above system in a matrix form

$$A = \begin{bmatrix} 0.9999 & -1.0001 \\ 1 & 1 \end{bmatrix}; \quad b = \begin{bmatrix} 1 \\ 1+\epsilon \end{bmatrix}$$

$$A\vec{x} = b; \quad \vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Solving the above yields the following:

$$\vec{x} = A^{-1}b$$

$$\vec{x} = (x, y) = (5000.5\epsilon + 0.5, \;\; 4999.5\epsilon - 0.5)$$

In [5]:
```
from sympy import Matrix, Symbol, solve_linear_system
x = Symbol('x')
y = Symbol('y')
e = Symbol('e')

system = Matrix (([0.9999, -1.0001,  1.0],\
                  [ 1.0,   -1.0,     1.0+e]))\


sol=solve_linear_system(system, x, y)
```

```
print (sol)
```

```
{x: 5000.50000000055*e + 0.5, y: 4999.50000000055*e - 0.5}
```

Solve a Boundary Value Problem (see Details in Lecture #11)

Consider the folowing 2nd-order ODE with two boundary conditions:

$$\frac{d^2y}{dt^2} = 4y; \quad y(0) = 1.1752; \quad y(1) = 10.0179$$

Applying a finite difference method, we obtain the following system in a matrix form:

$$Ax = b,$$

$$A = \begin{bmatrix} -2.25 & 1.0 & 0 \\ 1.0 & -2.25 & 1 \\ 0 & 1.0 & -2.25 \end{bmatrix};$$

$$b = \begin{bmatrix} -1.1752 \\ 0 \\ -10.0179 \end{bmatrix}$$

$$or \;\; b^T = (-1.1752, 0, 10.0179).$$

Below, we illustrate how to apply solve_linear_system to compute the solution to the above system.

```
from sympy import Matrix, Symbol, solve_linear_system
x  = Symbol('x')
y  = Symbol('y')
z  = Symbol('z')

                                  A                b
system = Matrix (([-2.25,   1.0,  0.0,  -1.1752],
                  [ 1.0, -2.25,  1.0,   0.],
                  [0.0,   1.0, -2.25, -10.0179]))
#print (system)


sol=solve_linear_system(system, x, y, z)

print (sol)
```

$Ay = b$

In [4]:
```
from sympy import Matrix, Symbol, solve_linear_system
x = Symbol('x')
y = Symbol('y')
z = Symbol('z')

system = Matrix (([-2.25,   1.0,  0.0,   -1.1752],\
                  [ 1.0,   -2.25, 1.0,    0.],\
                  [ 0.0,    1.0, -2.25, -10.0179]))

sol=solve_linear_system(system, x, y, z)

print (sol)
```

```
{x: 2.14670657596372, y: 3.65488979591837, z: 6.07679546485261}
```

Numerical Results for the Example

$$\frac{d^2y}{dt^2} = 4y \qquad y(0) = 1.1752 \qquad y(1) = 10.0179$$

	t	x	numerical	analytical	error
	0	1.0	(1.1752)	1.1752	n/a
X	0.25	1.5	2.1467	2.1293	-0.0174
Y	0.50	2.0	3.6549	3.6269	-0.0280
Z	0.75	2.5	6.0768	6.0502	-0.0266
	1.0	3.0	(10.0179)	10.0179	n/a

Below, we illustrate the advantage of symbolic computing by adding a tiny perturbation into the boundary condition as follows:

$$\frac{d^2y}{dt^2} = 4y; \quad y(0) = 1.1752 + \epsilon; \quad y(1) = 10.0179$$

Applying a finite difference method, we obtain the followi

$$Ax = b$$

$$A = \begin{bmatrix} -2.25 & 1.0 & 0 \\ 1.0 & -2.25 & 1 \\ 0 & 1.0 & -2.25 \end{bmatrix}; \quad b = \begin{bmatrix} -1.1752 + \epsilon \\ 0 \\ 10.0179 \end{bmatrix}$$

Below, we solve the above system using solve_linear_system again.

```
from sympy import Matrix, Symbol, solve_linear_system
x   = Symbol('x')
y   = Symbol('y')
z   = Symbol('z')
e   = Symbol('e')

system = Matrix (([-2.25,   1.0,  0.0,  -1.1752-e],
                  [ 1.0, -2.25,  1.0,   0.],
                   [0.0,   1.0, -2.25, -10.0179]))
#print (system)


sol=solve_linear_system(system, x, y, z)

print (sol)
```

A

b

$Ay = b$

In [5]:
```
from sympy import Matrix, Symbol, solve_linear_system
x = Symbol('x')
y = Symbol('y')
z = Symbol('z')
```

```
e = Symbol('e')

system = Matrix (([-2.25,   1.0,  0.0,   -1.1752-e],\
                  [ 1.0,   -2.25, 1.0,    0.],\
                  [ 0.0,    1.0, -2.25, -10.0179]))

sol=solve_linear_system(system, x, y, z)

print (sol)
```

```
{x: 0.589569160997732*e + 2.14670657596372, y: 0.326530612244898*e + 3.65488
979591837, z: 0.145124716553288*e + 6.07679546485261}
```

$$\frac{\partial^2 u}{\partial x^2} = f(x)$$

$$f(x_i) = f_i$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}$$

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} = f_i$$

backup

In [3]:
```
from sympy import Matrix, Symbol, solve_linear_system
x = Symbol('x')
y = Symbol('y')
e = Symbol('e')

system = Matrix (([0.9999, -1.0001,  0.0],\
                  [ 1.0,   -1.0,     0.0+e]))\


sol=solve_linear_system(system, x, y)

print (sol)
```

```
{x: 5000.50000000055*e, y: 4999.50000000055*e}
```

In [ ]:
```

```