

# Python Setup and Introduction to Matrix Algebra

## Numerical Structure and Stability

Michael R. Gosz

Department of Mechanical, Materials, and Aerospace Engineering  
Illinois Institute of Technology

## Lecture Objectives

By the end of this lecture, you should be able to:

- ▶ Recognize what a correct Python environment looks like
- ▶ Understand why matrix algebra is central to computational mechanics
- ▶ Appreciate that numerical issues can arise even in simple systems

## Why We Start with the Python Environment

- ▶ All computational examples rely on a working Python setup
- ▶ Errors early in the semester tend to compound later
- ▶ A correct setup enables reproducibility and confidence

### Key Message

If your environment works today, it will work all semester.

## What a Successful Setup Looks Like

A working environment allows you to:

- ▶ Activate a virtual environment
- ▶ Launch Jupyter Notebook
- ▶ Select the correct kernel
- ▶ Import core libraries without errors

### Minimal Verification

```
import numpy, sympy, matplotlib
```

# Why Matrix Algebra Matters

Almost every computational mechanics problem reduces to:

$$\mathbf{K}\mathbf{x} = \mathbf{f}$$

- ▶ Unknowns → vector
- ▶ Physical laws → matrix
- ▶ Loads and sources → right-hand side

## Big Picture

Matrix algebra is the language of computational mechanics.

## Matrices Encode Structure

A matrix represents more than numbers:

- ▶ Coupling between variables
- ▶ Physical constraints
- ▶ Geometry and material behavior

Matrix–vector multiplication describes how information propagates through a system.

## Numerical Sensitivity

- ▶ Computers use finite-precision arithmetic
- ▶ Small numerical errors are unavoidable
- ▶ Some systems amplify these errors dramatically

### Terminology

This sensitivity is called *conditioning*.

## Well-Conditioned vs Ill-Conditioned Systems

- ▶ **Well conditioned:** small input errors produce small output errors
- ▶ **Ill conditioned:** small input errors produce large output errors

### Important

Numerical instability is often a property of the problem, not the code.

# The Hilbert Matrix

The Hilbert matrix is defined by:

$$H_{ij} = \frac{1}{i + j - 1}$$

$H_{5 \times 5}$

1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$
$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$
$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$
$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{8}$
$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{8}$	$\frac{1}{9}$

- ▶ Simple definition
- ▶ Smooth entries
- ▶ Extremely ill conditioned

Larger entries cluster near the top-left.

# Why the Hilbert Matrix Matters

- ▶ Appears harmless
- ▶ Defeats standard numerical solvers
- ▶ Demonstrates limits of floating-point arithmetic

## Notebook Demonstration

In the accompanying notebook, we:

- ▶ Construct Hilbert matrices of increasing size
- ▶ Solve linear systems numerically
- ▶ Observe loss of accuracy

### Key Observation

As matrix size increases, numerical reliability collapses.

## What Went Wrong?

- ▶ The mathematical problem is well defined
- ▶ The algorithm is correct
- ▶ Floating-point arithmetic limits accuracy

### Critical Lesson

The computer did not fail — the mathematics did.

## Implications for This Course

- ▶ Matrix structure matters
- ▶ Solver choice matters
- ▶ Blind trust in numerical output is dangerous

These themes will reappear throughout the semester.

## Looking Ahead

In upcoming lectures, we will:

- ▶ Define condition numbers formally
- ▶ Study direct and iterative solvers
- ▶ Connect conditioning to stiffness matrices
- ▶ Develop strategies for numerical robustness

## Summary

- ▶ A correct Python setup is essential
- ▶ Matrix algebra underpins computational mechanics
- ▶ Numerical instability is unavoidable but manageable

## Next Step

Work through the Hilbert matrix notebook carefully.