TDT4127 Programming and Numerics
Week 41

Gaussian elimination

Plotting with Python

Learning goals

- Goals
  - Solving linear systems
  - Algorithm:
    - ***Gaussian elimination***
  - Plotting functions
    - Requires **matplotlib** library

- Curriculum
  - Exercise sets 7 (and 10)
  - Programming for Computations - Python
    - Ch. 1.4, 1.5.7

Exercise set 7

- Two numerics exercises
  - One on **plotting** (relevant parts covered in this lecture)
  - One on **Newton's method for systems of equations**
    - *This is not covered before next week*
    - Leaves 1½ weeks after the lecture to finish the exercise
  - You can still do the Newton's exercise before next week's lecture, there is a note explaining it in the exercise
    - If you prefer having the lecture first, do the rest of the exercise set and save the Newton's exercise

Gaussian elimination, recap

To solve $A\boldsymbol{x} = \boldsymbol{b}$, first write it in augmented form.

Start with pivot row 0 and pivot column 0, then:

1. **Swap** the entries of the pivot row with the row **below** with *largest absolute value* in the pivot column
    1. If impossible, move pivot column to the right
2. **Reduce** the rows below the pivot row by adding multiples of the pivot row to zero out the pivot column
3. **Move** the pivot row down and pivot column to the right. If on the **last row** or the augmented column, stop. Else, repeat from 1.

Partial pivoting

- In step no. **1**, we **swap** the entries of the pivot row with the row below with largest entry in the pivot column.
- Swapping like this is called *partial pivoting*.

- Why is this necessary?
  - It’s not what is taught in non-numerical linear algebra courses 🤔
- **Answer:** Partial pivoting reduces numerical errors due to round-off (floating point precision).

Partial pivoting – example of error

- Consider the system

$$\left[\begin{array}{cc|c} 10^{-5} * 1 & 1 & 1 \\ 1 & 1 & 2 \end{array}\right]$$

and assume we have 4 digits of precision.

- **No pivoting:** Subtract 10^5 times the first row from the second to get

$$\left[\begin{array}{cc|c} 10^{-5} * 1 & 1 & 1 \\ 0 & -10^4 * 9.9999 & -10^4 * 9.9998 \end{array}\right]$$

Partial pivoting – example of error

- **No pivoting:** Subtract 10^5 times the first row from the second to get

$$\begin{bmatrix} 10^{-5} * 1 & 1 & | & 1 \\ 0 & -10^4 * 9.9999 & | & -10^4 * 9.9998 \end{bmatrix}$$

- With 4 digits of precision, this rounds to

$$\begin{bmatrix} 10^{-5} * 1 & 1 & | & 1 \\ 0 & -10^5 * 1.000 & | & -10^5 * 1.000 \end{bmatrix}$$

- This can be easily solved: $x_2 = 1, x_1 = 0$. This is wrong!
  - *The correct solution is* $x_2 = 99998/99999$, $x_1 = 100000/99999$*!*
  - Sensitivity to roundoff errors is an example of *numerical instability*
    - *Small calculation errors cause big changes in the solution*
  - Double-precision floats (Python) have ~16 digit precision, but numerical instability can still be an issue

Partial pivoting

- What happens if we do partial pivoting? After swapping:

$$\begin{bmatrix} 1 & 1 & | & 2 \\ 10^{-5} * 1 & 1 & | & 1 \end{bmatrix}$$

- Subtract 10^{-5} times the first row from the second to get

$$\begin{bmatrix} 1 & 1 & | & 2 \\ 0 & 1.000 & | & 1.000 \end{bmatrix}$$

  with 4 digits of precision
- This solves to $x_2 = 1, x_1 = 1$, a more precise solution.
- Adding *large* multiples of rows causes numerical errors by «drowning out» the information in the other rows
  - Due to roundoff errors
- Adding *smaller* multiples of rows is safer since it leaves less chance of information loss
  - Partial pivoting means all row multiplications are ≤ 1.

Complete pivoting

- One can also do *complete pivoting*, looking through both rows **and columns** for the maximal element
- Requires a **swap** for the **column** of the maximal element
  - And the row of the maximal element
- Only necessary in the worst cases
- Takes more time. For a matrix with $n \times n$ entries, we need to look at $\sim n^2$ entries to find the max, compared to n entries with partial pivoting.
  - This is not really an issue for small (1000 x 1000) matrices, but becomes a real problem with larger matrices.

When does Gaussian elimination work?

- As long as the problem has a solution!
  - …and as long as *partial/complete pivoting* is enough to avoid accuracy problems (which is almost always!)
  - There is **no need** for analysis of convergence or error estimates
    - When you run it all the way, you get the **exact solution**
    - If you stop without letting the algorithm finish, you get **nothing**
- If the problem does not have a solution? Examples:

$$\begin{bmatrix} 1 & 0 & | & 1 \\ 0 & 0 & | & 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 0 & 1 & | & 1 \\ 0 & 0 & | & 1 \end{bmatrix}$$

  - One can add checks in the code to look for these under/overdetermined situations and act accordingly.

Alternatives to Gaussian elimination

- The below is not curriculum
- Gaussian elimination is **slow for large systems**
  - For an $n \times n$ system, each row reduction requires ~n^2 operations. With n rows, this is ~n^3 operations, i.e. n^2 operations per unknown in $\boldsymbol{x}$. As n grows, this quickly becomes too much.
- Some large systems have **special structures**
  - *Triangular*, *banded* ,*Toeplitz, sparse*
  - These structures can be **exploited** to make GE faster
- Otherwise, one should use faster, **inexact** methods that do not give the *exact* solution (similar to Newton's)
  - **Krylov subspace methods** are used a lot in practice
  - These are often what you get when using packages or MATLAB

Plotting in Python

- Use the **matplotlib** library https://matplotlib.org/gallery.html
- Why use **matplotlib**?
  - Same reason as we use **Python**: **free** to use, lots of possibilities
  - Plenty of examples available online
- Why not **MATLAB**?
  - Matplotlib mimics MATLAB's plotting, but MATLAB costs **money**
  - MATLAB may have more tools, especially in 3D
- Why not use **Excel**?
  - Excel: Easy to make one-off figures, not lots of figures
  - Data handling is then often easier (and more general) in Python
  - If we want a certain style of plot, matplotlib lets us use others' setups very easily by just cloning their code
    - Instead of spending time trying to reproduce the exact Excel settings

The matplotlib library

- Installing the **matplotlib** library
- Some Mac users may have it installed already
- https://matplotlib.org/users/installing.html
- An installation guide is in the works

How does it work?

- For those familiar with GeoGebra: In GeoGebra, we just input the function and it magically draws it.
  - Matplotlib gives us a more fine-grained tool
- Include matplotlib using the command

```
include matplotlib.pyplot as plt
```

- Given lists `x` and `y` of equal length, we plot the points (`x[i],y[i]`) with the command `plt.plot(x,y)`
  - Same as when drawing a graph from hand if you have no idea how it looks: put dots on the coordinates and draw lines between
- To see the figure, use `plt.show()`

Example

```
#Import plotting library
import matplotlib.pyplot as plt
#Inform about data points to plot
plt.plot([1,2,3,4], [1,4,9,16])
#Inform about label on the y axis
plt.ylabel('some numbers')
#Axis range: [x_min, x_max, y_min, y_max]
plt.axis([0,4,0,16])
#Show the plot in a pop-up window
plt.show()
```

Plotting styles

- The default behaviour of `plt.plot()` is to connect the points with lines
- We can change this using additional arguments after the `x/y` coordinates
  - For example, to plot `y` over the `x` points as red **circles**:

    `plt.plot(x,y,'ro')`
  - To plot `y` over the `x` points as green **triangles**:

    `plt.plot(x,y,'g^')`
  - More options can be found here: https://matplotlib.org/users/pyplot_tutorial.html

Plotting several graphs in one figure

- If we want to generate several graphs, plot all of them first using `plt.plot()`, then use `plt.show()`

```
#Import plotting library
import matplotlib.pyplot as plt
x = …
y1 = f(x)
Y2 = g(x)
plt.plot(x,y1)
plt.plot(x,y2)
plt.show()
```

Summary

- We use partial pivoting in Gaussian elimination to avoid issues with floating point precision
- Except potential precision issues, Gaussian elimination is a safe and stable method for solving linear problems
  - But not necessarily the fastest – inexact methods can be *good enough* and much faster. Not curriculum, though.
- Plotting in Python can be done using the **matplotlib** library
  - We will not be very fancy with it, but it exists and is versatile

Questions?