Numerical Methods

Math 3338 – Spring 2022

Worksheet 11

Solving Systems of Linear Equations

1 Reading

CP	6.1
NMEP	2.1, 2.2, 2.3, 2.5

Table 1: Sections Covered

Our goal is to solve a system of equations with the form,

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

Or, more simply, equations of the form $Ax = b$ where A is an $n \times n$ matrix. Before any of this, we need to know more about Numpy arrays.

2 Gauss-Jordan Elimination

Gauss-Jordan elimination is a systematic method to reduce a matrix into upper triangular form with ones on the diagonal. The basic process is as follows,

1. Divide the first row by the upper left coordinate. This makes a 1 in that location.
2. Add the first row to the lower rows, making 0's beneath the leading 1.
3. Repeat for the remaining rows.

The book does a spectacular job of explaining this.

Once your system is in upper triangular form, it's very easy to solve using back substitution. This is exactly what it sounds like, try to do it by hand you'll figure it out.

The major "problem" that arises with Gauss-Jordan is what if one of the rows has a leading 0? In that case you need to pivot your matrix. Or swap rows to ensure this doesn't happen.

Here is an example of a 3×4 system.

$$\begin{aligned} \left[\begin{array}{ccc|c} 2 & 1 & 4 & 1 \\ 1 & -1 & 2 & 2 \\ 2 & 1 & 3 & -1 \end{array}\right] &\to \left[\begin{array}{ccc|c} 1 & .5 & 2 & .5 \\ 1 & -1 & 2 & 2 \\ 2 & 1 & 3 & -1 \end{array}\right] \\ &\to \left[\begin{array}{ccc|c} 1 & .5 & 2 & .5 \\ 0 & -1.5 & 0 & 1.5 \\ 0 & 0 & -1 & -2 \end{array}\right] \\ &\to \left[\begin{array}{ccc|c} 1 & .5 & 2 & .5 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 2 \end{array}\right] \end{aligned}$$

The final step is in upper triangular form. Now you can use back substitution to solve the system. What is back substitution? Think about this as a system of equations.

$$\begin{aligned} x_1 + .5x_2 + 2x_3 &= .5 \\ x_2 + 0x_3 &= -1 \\ x_3 &= 2 \end{aligned}$$

We know $x_3 = 2$ so plug that into the above two equations. Then $x_2 = -1$ and we can plug that in to the top equation to find $x_1 = -3$. The point is that once you get the system to upper triangular form it's very easy to get the solution.

Numerical Methods

Math 3338 – Spring 2022

Homework 11 (Due: Tuesday, February 22)

Note: numpy.concatenate might be a useful command in these problems.

Problem 1 (1 pt) Write a function called `backsub(A,b)` that takes an upper-triangular matrix A and a column vector b and returns a vector x so that $Ax = b$.

Problem 2 (1 pt) Write a program called `gaussian_elim` that solves a system $Ax = b$ using Gaussian elimination. Your input should be `A,b` where A is a matrix (as a nested array) and b is a column array. Return a column array containing the solution x.

You should test this with a few matrices. How does your function act on the given matrix?

$$\begin{bmatrix} 0 & 1 & 2 \\ 4 & -1 & 3 \end{bmatrix}$$

Numpy, of course, has methods to solve linear systems. If you are unable to complete the above, use the package `numpy.linalg`. This has a bunch of useful methods you can take advantage of. You will need to find a list of methods this package contains and learn how to use them.

Problem 3 (1 pt) The Hilbert Matrix is an extremely famous and troublesome matrix. It's defined as,

$$A = \begin{bmatrix} 1 & 1/2 & 1/3 & \cdots \\ 1/2 & 1/3 & 1/4 & \cdots \\ 1/3 & 1/4 & 1/5 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Write a program called `hilbert` that solves $Ax = b$ where

$$b_i = \sum_{j=1}^{n} A_{ij}.$$

Your program should take and input of n, the size of A. The output should be `x,A,b`.

Solve this system for $n = 5, 10, 20, 50, 100$. Discuss the solutions.

Something you should notice is that there is one solution to $Ax = b$ and it's

$$x = \begin{bmatrix} 1 & 1 & 1 & \cdots \end{bmatrix}^T.$$

You can test this with matrix multiplication (you should)[1] or just by thinking about how b is defined. Does your answer to the previous parts match? Why or why not?

Solve this multiple ways. If you created the above program, use it and `numpy.linalg`. Compare all the solutions.

[1]numpy.dot(A,b) to multiply matrices