Fawaz Shah

System Life Cycle – Simultaneous Equation Solver For 3 Unknowns Python Program

Analysis

After having studied the FP1 module of Further Maths AS Level, I was inspired by the simultaneous equations methods involved to create my own simultaneous equation solver. The program takes a set of 3 equations and (if a unique solution exists) prints the solution. If no unique solution exists then an adequate message is printed (there are either infinite solutions or no solutions).

Design

Each equation is presented in the form ***Ax + By + Cz = D***
A, B, C and D for each equation are stored in an augmented (3x4) matrix. The matrix is transformed into **RREF** (**R**educed **R**ow **E**chelon **F**orm), using Gaussian elimination (otherwise known as row operations). The rules for transforming a matrix into RREF are fairly complex, but can be viewed here: https://en.wikipedia.org/wiki/Row_echelon_form

To check if a unique solution exists to the system, the determinant of the original 3x3 matrix must be found (i.e. the augmented 3x4 matrix but with the last column removed). This can be done by **expansion of the determinant by the first row** (http://mathworld.wolfram.com/DeterminantExpansionbyMinors.html). If there is no unique solution, we have to determine whether there are no solutions or infinite solutions. This is slightly trickier, and we need to transform the matrix into RREF for this.

In the following examples, * denotes any **non-zero** constant.

In RREF, a system with no solutions looks like this:

[1 0 0 *]
[0 1 0 *]
[0 0 0 *]

The important thing to notice here is that the last row is implying that **0x + 0y + 0z = ***, which is impossible. Therefore there are no solutions.

In RREF, a system with infinite solutions looks like this:

[1 0 0 *]
[0 1 0 *]
[0 0 0 0]

Here, the last row implies that **0x + 0y + 0z = 0**. Since this is true in all situations, this system has infinite solutions.

For further clarity, a system with a unique solution would look like this:

```
[1    0    0    *]
[0    1    0    *]
[0    0    1    *]
```

You would just read off the constants on the right to get the unique values of x, y and z that make up the solution.

Implementation

The program was written in Python, with the matrix being stored in a 3x4 2D array. The first dimension of each array represents a row, and the second dimension is the column. In this way operations can be performed upon each element in an entire row (row operations are necessary for Gaussian elimination). I split the program up into functions which handle different tasks (e.g. getting user input, switching rows around, adding multiples of one row onto another etc.). At the end of the program I used the limit_denominator() method from the ‘fractions’ Python library to round any fractional answers to a sensible number of digits.

Testing

I tested the program with some matrices from my FP1 textbook, making sure to test matrices with different types of solutions (unique, infinite, none). The program achieved the correct answer in all cases.

Evaluation

I feel this project went very well, since I got to explore different areas of both programming and maths in detail (two of my favourite things). The program turned out to be successful. If I had more time I would have added error-handling for if the user does something unintended (e.g. enters a string instead of an integer).