Princeton University
Department of Operations Research
and Financial Engineering

ORF 201
Computer Methods in Problem Solving

Lab 8: The Guassian Eliminator

Due Sunday, Apr 15, 11:59 pm

---

1. PROBLEM DESCRIPTION

It is common in engineering and science to need to solve a system of equations such as this one:

$$\begin{array}{rcrcrcr} -x_1 & & & + & 3x_3 & = & -1 \\ -x_1 & - & 2x_2 & - & x_3 & = & -2 \\ & & x_2 & + & 3x_3 & = & 1 \end{array}$$

(for this example, the solution is $x_1 = 5/2$, $x_2 = -1/2$, and $x_3 = 1/2$).

If each equation is linear, as in the example above, then the system is called a *system of linear equations*. For systems of linear equations, there is a systematic solution procedure which will solve the system if possible or make it clear that no solution exists. This procedure is called *Gaussian elimination*. It is based on three simple transformations:

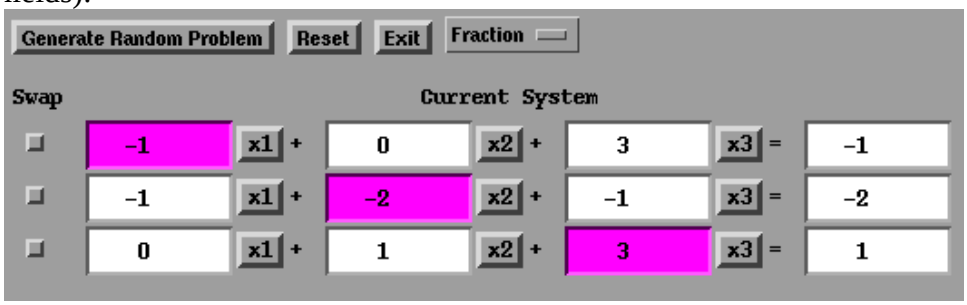
- Swapping two equations.
- Dividing an equation by a (nonzero) number.
- Replacing an equation with itself minus a constant times another equation.

For fairly obvious reasons, these transformations are called *row operations*.

Think of the coefficients multiplying the x_j's in the system of equations as elements of an array of numbers. Gaussian elimination proceeds in two stages. The first stage works from the upper-left diagonal element down to the lower-right diagonal element and step-by-step modifies the system to make the diagonal elements equal to one and the elements below the diagonal equal to zero. The second stage goes in the reverse direction, from the lower-right to the upper-left, and step-by-step makes the elements above the diagonals equal to zero.

For this lab, you will write a Java applet to facilitate the task of solving systems of linear equations by Gaussian elimination. With this applet you will be able to gain a fuller understanding of how and why Gaussian elimination works.

The applet will display two textfields in which to specify the number of equations and the number of variables. It will also have a *Go Eliminating* button, which when pushed will cause a new window to open, an *eliminator frame*, which will look something like this (after entering data into its number fields):



Each coefficient is contained in a `NumberField`. The variables are shown on `Buttons`. The boxes in the column labelled `Swap` are `Checkboxes`. Three types of row operations are provided as follows:

- Clicking on one of the *diagonal* variable buttons causes that equation to be divided by a constant so that the coefficient in front of the button just pushed becomes one.
- Clicking on an *off-diagonal* variable button, say the j-th variable in the i-th equation, causes the i-th equation to be replaced by itself minus a constant times the j-th equation where the constant is chosen so that the coefficient in front of the button just pushed becomes zero.
- Clicking on two checkboxes in the *swap* column causes the two corresponding equations to swap positions and the check-marks to disappear.

2. Programming Notes

As usual, you need to create some directories and copy some files. Begin like this:

```
cd public_html/JAVA/ORF201
mkdir gauss_elim
cd gauss_elim
```

Then copy the following files:

```
cp /u/orf201/public_html/JAVA/ORF201/gauss_elim/index.html .
cp /u/orf201/public_html/JAVA/ORF201/gauss_elim/gauss_elim.html .
cp /u/orf201/public_html/JAVA/ORF201/gauss_elim/GaussElim.java .
```

Compile the java code that you just copied over:

```
javac GaussElim.java
```

Then use appletviewer to see what the code currently does:

```
appletviewer gauss_elim.html
```

An applet window should pop up but there will be nothing showing since the working program doesn't do anything at the moment.

Essentially every place `GaussElim.java` where you need to add code is marked with a comment of the following form:

```
/***** Blah, blah, blah, ...
and more blah.   ******/
```

These comments give fairly detailed instructions about what needs to be done.

2.1. **Remembering Your Roots.** In a few places, you will need to tell a method the name of the class from which it was called. This is done as follows:

```
class upAbove
{
    void tellEmBelow()
    {
        downBelow db = new downBelow();

        db.setUp(this);
    }

    void doItUpHere()
    {
        System.out.println("This gets printed");
    }
}

class downBelow
{
    upAbove ua;

    void setUp(upAbove ua)
    {
        this.ua = ua;
    }

    void doSomething()
```

```
        {
            ua.doItUpHere();
        }
    }
```

2.2. **Checkbox methods.** With checkboxes, there are two methods `getState()` and `setState()` that you will need to use. The first takes no arguments but returns a boolean: true if the box is checked, false otherwise. The second takes a boolean as argument and returns nothing. It is used to set the checkbox's state to checked (true) or unchecked (false).

2.3. **Extension Classes.** Most of the classes you will define need to extend some existing class. In particular, `myCheckbox` and `myButton` must extend `Checkbox` and `Button`, respectively. In addition to the usual functionality of these GUI classes, they must store away information about which boxes/buttons they are and how to call methods defined "up above".

2.4. **Static Variables.** Your `myCheckbox` class will need to use static variables to store certain information that will then be available to every instance of `myCheckbox`. An example that is similar to what you need to do was covered in the Thursday lecture of week 10. The source code for this example can be viewed/downloaded by visiting

```
http://www.sor.princeton.edu/~rvdb/201/lectures/code/
```

(there is a link to this location just below the list of course lectures on the ORF 201 home page) and clicking on `CheckBoxDemo.java`.

2.5. **NumberField.** The class `NumberField` is defined in `myutil`. It is an extension of the class `TextField`, which is part of `java.awt`. I created `NumberField` to make getting and setting a double precision number slightly easier. Here is a small example illustrating how it works:

```
double a;
NumberField A;

A = new NumberField("0.0", 9);

a = A.fromNumberField(); // takes text from NumberField,
                         // converts to double, stores in a

/* do some computations that change a */

A.toNumberField(a);      // takes double, converts to text,
                         // puts on NumberField
```

The first argument in the constructor (the string ”0.0” above) is the initial text that will appear in the NumberField. The second argument (9 above) tells how many characters wide to make the NumberField.

2.6. **Decimal vs. Fraction.** The class `NumberField` has a method called `setFormat()` which takes a string as an argument. To tell a `NumberField` that you want output format in decimal notation, use

```
A.setFormat("Decimal");
```

To specify fraction notation, use

```
A.setFormat("Fraction");
```

These are the only two strings that `setFormat()` understands.

3. GAUSS_ELIM.HTML

Don’t forget to edit `gauss_elim.html` to make the following changes:

- *Change* `codebase` *from* `"../.."` *to* `http://www.princeton.edu/~yourname/JAVA`.
- *Add the honor code pledge:*
  This program represents my own work in accordance with University regulations.

4. GOING PUBLIC

If your umask is set for restricted access (i.e., 077), don’t forget to make your files public:

```
chmod a+rx public_html/JAVA/ORF201/gauss_elim
chmod a+r public_html/JAVA/ORF201/gauss_elim/index.html .
chmod a+r public_html/JAVA/ORF201/gauss_elim/gauss_elim.html .
chmod a+r public_html/JAVA/ORF201/gauss_elim/GaussElim.class .
```

After you’ve made your files public check to see if you can bring your applet up in a browser. Fire up netscape and go to the following address:

```
http://www.princeton.edu/~yourname/JAVA/ORF201/gauss_elim/gauss_elim.html
```

If your code works and permissions are set correctly, you should see the maze applet in the browser.

5. Minimum Requirements

At a minimum your program must:

(1) implement a GUI interface to accomodate the three types of row operations.
(2) correctly implement the three types of row operations.