

# SGF Tutorial 1

(Simulation Generation Framework) (<http://circe.engr.wisc.edu>)

What is SGF?

It's basically software that solves PDEs (Partial Differential Equations). It can be used in a variety of situations (Electromagnetics, Semiconductor devices and just about anywhere where you use PDEs). It converts code that you write into C/C++ code and gives you results in simpler formats. (I believe it can also plot results, but I'm not really sure how easy that is)

Why SGF?

No specific reason. SGF can solve equations that you use in boundary conditions, field calculations etc. You can plot the results and that helps a lot in visualizing the fields, flux and how they change.

Also it's free (you can download it from the site and install it in your PC) and it's pretty simple to work with. Once you get to doing more advanced programs, you will realize that it makes life a lot easier.

## SGF Programming Syntax

- All code that you write needs to be saved in a .sg file (Example: sample.sg)
- You can edit the file using any text editor (notepad, wordpad etc [in Windows]; emacs, pico, vi etc [in Unix])
- Functions:
  - ✓ Every code needs to have a function that is executed. The primary function in the code is "main". So when you compile code and execute it, SGF is going to look for the "main" function and run it first.
  - ✓ Every function starts with "begin" and ends with "end"

Example:

```
begin main  
// Code for function main in here  
end
```

- Comments:

- ✓ Comments are used to make programs more understandable and explain their purpose.
- ✓ It's always a good idea adding comments to your code so as to help you or others understand what you've written. It's good programming practice, so do place comments in your code wherever necessary.
- ✓ All comments in your SGF code have to begin with “ // ”

Example:

```
// Comments for function main
begin main

//Comments for code inside the main function

end
```

- Every line of actual code (excluding the comments) in your .sg file needs to end with a semicolon “;”

Example:

```
begin main
var A = 5;
end
```

- Constants:

- ✓ Constants are elements (or values) which are not changed during the program
- ✓ It's always a good idea declaring constants to hold values which do not change. That way when you want to change their value, all you have to do is modify just one line of code.
- ✓ It's standard programming practice to use all capital letters while naming constants (Eg. PI, LENGTH, DIM etc)

Example:

```
const PI = 3.14159265358979323846 ;
const LENGTH = 10;
```

- Variables:

- ✓ Variables are values that are to be modified or calculated during the course of a program
- ✓ Variables can also be an array of elements (could have more than one dimension also)

Example:

```
var a; // a is declared to be a variable
var b[10]; // b is an array that consists of 10 elements
const DIM = 20;
var c[DIM]; // c is an array of DIM(20) number of elements
var d[DIM,DIM]; // d is a 2D array having DIM x DIM(400)elements
```

- ✓ For arrays we follow 0 indexing (i.e. in the example above b[0] is the first element & b[9] is the last element of b)

- Assignment Statements:

- ✓ These are statements that you use to assign values to variables

Example:

```
const DIM = 10;
var a;
var b[DIM];
assign a = 5.0; // assign a value 5.0 to 'a'
assign b[0..DIM-1] = 0.5; // assign a value 0.5 to all the elements of b
assign b[i = 0..DIM-1] = i * 2.0; // assigns a value of 2i to the ith element
```

- Equations:

- ✓ You can declare equations in SGF using “equ”
- ✓ You need to give a name for an equation so that you can reference it

Example:

```
var x1, x2;
equ x1 -> x1 + x2 = 4; // Use the equation  $x1 + x2 = 4$  to solve for x1
equ x2 -> 3*x1 + x2 = 7; // Use the equation  $3*x1 + x2 = 7$  to solve for x2
```

- ✓ The arrow between x1 and x1 is just a combination of a hyphen “-“ and a greater than symbol “>”

- Sample Code:

```
// Filename: lin_eq2.sg
// Solving 2 linear equations with 2 unknowns

var x1, x2;           // x1 & x2 are variables
unknown x1, x2;      // x1 & x2 are unknowns
equ x1 -> x1 + x2 = 4; // 1st linear equation
equ x2 -> 3*x1 + x2 = 7; // 2nd linear equation
begin main           // Start of the function
solve;               // Solve the equations for the unknowns
open "answer" write; // Open file "answer" to write
file write x1, x2;   // write x1 & x2 into answer
close;               // close the file
end                  // End of the function
```

## Running the SGF program

- Although a Windows version of SGF is available, we will use the Unix version
- To run Unix on a Windows machine, we will use a remote login software called "Exceed"
  - ✓ For those who've never used Exceed before, you need to add it on to your CAE application list ( Start > CAE Applications > Add Applications to Start Menu )
  - ✓ Once your user application manager comes on, search for Exceed (either in By alphabetical, or it's in By Category under Internet)
  - ✓ Once you've added it on to your CAE applications, run Exceed ( Start > CAE Applications > Internet > Exceed > Connect to Random SUN machine )
  - ✓ Now Exceed brings up a remote login window
  - ✓ Under the tab "options", go to "session" and choose Gnome 2.2 Desktop Environment (If you have any preferred desktop environment , use that)
  - ✓ Now enter your username and then password (same as your CAE login)
- Before you can use SGF, you have to include its directory in the path.
  - ✓ Run a terminal window in Unix (a window with a command prompt)
  - ✓ You need to add in a line in your .cshrc file, so open the file with an editor like pico or emacs

sun-100% **emacs .cshrc** (or pico .cshrc)

- ✓ Type out the following at the end of the file

```
set path = ( ~siwaporn/SGF/SGFramework $path )
```

(don't forget to press "enter" at the end of the line)

- ✓ Save the changes to your .cshrc file
  - ✓ The .cshrc file is something that is run when you login into Unix, so to activate the change that you made, for this session, you can either just type out the whole line again at the command prompt or just logout and login again.
- Now a quick recap of useful Unix commands
    - ✓ To create a directory: sun-100% **mkdir dirname**
    - ✓ To remove a directory: sun-100% **rmdir dirname**
    - ✓ To enter another folder sun-100% **cd foldername**
    - ✓ To list files in a folder sun-100% **ls**
    - ✓ To go to the parent directory sun-100% **cd ..**
    - ✓ To display a file sun-100% **cat filename** (or more filename)
  - Create a folder in your home directory called ece220 (for your SGF files)
  - Create a subfolder in ece220 called lin\_eq
  - In Windows or Unix, go to the course homepage and download the code file for Linear Equations (lin\_eq2.sg) from the SGF folder. Save it in your lin\_eq directory.
  - Now hopefully you should be ready to run the code for SGF. Go to the lin\_eq directory and at the command prompt, type **SG lin\_eq2**
  - Now when you list the files (using ls) in your directory, you should find a lot of additional files (lin\_eq2.cpp, lin\_eq2.h etc) including a file named "answer".
  - Display the contents of the "answer" file. (you should get the results 1.5 and 2.5 which are the values of x1 and x2)

## Additional Work

- Now modify the lin\_eq2 file to solve three linear equations

$$\begin{aligned}x_1 + x_2 + x_3 &= 4; \\x_1 + 2x_2 + 3x_3 &= 7; \\2x_1 + 3x_2 + x_3 &= 8;\end{aligned}$$

- The answers of course are  $x_1 = 2$ ,  $x_2 = 1$ ,  $x_3 = 1$
- Compare your code with lin\_eq3.sg (Three Linear Equations) code on the course homepage