PHYS 2030
Lab 6

October 27, 2014

Finish as many parts of the problem set as possible (and do not worry if you cannot finish all of them, since there is a lot to do). You can start from any problem, since they are somewhat unrelated.

Suggestions:

- Put lots of comments in your code. You might use some parts of the code for your next assignments (or projects), and one thing you do not want to do is to spend time on trying to recall after couple of months what does your code do and why does it do it in that particular way. In addition, commenting will help you track down any logical errors, which are very hard to find in general.
- Use meaningful names for your variables (even if they turn out to get somewhat long). The reason for this is the same as above.
- Do not just comment the particular lines of your code, but you can break your code into logical sections and give clear explanation of what is this section for.

You do not have to hand in anything and this problem set is not going to be graded.

1 Generating fractals with L-system

In class you saw the code for constructing the fractal, Koch snowflake, using recursive approach. To be more specific, you had a certain set of commands (such as 'move forward', 'rotate by some angle clockwise or counterclockwise') and after you modified this set of commands in some way (for Koch snowflake you replaced each occurrence of 'move forward' command by some other set of commands). This new (in principle longer, more complicated) set of commands can be modified again and again (by using the same transformation rule), i.e. the transformation can be applied as many times as one desires. Using similar approach, but different starting set of commands and transformation rules, we can construct fractals of various shapes.

In general, this method of producing fractals by applying some certain set of commands is called the L-system (`http://en.wikipedia.org/wiki/L-system`). From the link provided one can see that for some types of fractal construction, we should have some way of producing a set of commands for given initial string of commands and the transformation rule. For instance, we can call this function `[f]=LSystem(s,sRec,n)`, where `s` is the initial string of commands, `sRec` is the rule of transformation, `n` is the number of times to perform the transformation on the whole set of commands and `f` is the resulting string of commands after `n` iterations. For example, for Koch snowflake we would write the following

```
n=1;
sRec={'F' 'F-F++F-F';'+' '+';'-' '-';'T' 'T'};
sRec={'F' 'F-F++F-F';'+' '+';'-' '-';'T' 'T'};
[sCMD]=LSystem('F++F++F',sRec,n)
```

Here `sRec` is a matrix with each row representing one transformation rule with two columns, where first column represents the particular command (string element in this case) to which the rule is applied, and the second column is the rule itself. For instance, the first row here means that each 'F' has to be replaced by 'F-F++F-F' and the second row states for the rule that each '-' is replaced by '-' (which is the same thing as to say, that no transformation is applied).

If one runs this code, then the output is

```
sCMD = 'TF-F++F-F++F-F++F-F++F-F++F-F'
```

- Implement this function and **check** if its output is correct. Recursion will be very useful here.

Now we need to convert this string of commands into actions. You can think of the set of commands, which may include rotations and moving forward, as some particular trajectory of a particle in xy plane. I will leave the rest to you.

- Implement the function that transforms the set of commands into the set of points that you can plot. You will need some initial conditions for your 'particle' that can be given by initial coordinates $\{x_0, y_0\}$ and initial angle θ_0, which is the angle of the 'particle's velocity' with the x-axis.
- Test your function by plotting Koch snowflake for different values of n. Experiment with values for angles of rotation ('+' and '-') and see if you can produce new shapes.
- Plot Sierpinski triangle (Example 5 in the link given above).
- Show the plot of a fractal tree (Example 2 in the link above). This is a trickier example, because two new commands are added here, and so you would have to add some more code to your function.

- Produce a fractal plant (Example 7). This is an obvious way to see why fractals are useful in computer graphics.

2 Period doubling and Chaos

- Implement the code for discrete form similar to logistic equation,

$$x_{n+1} = x_n r(1 - x_n)$$

  Plot $\{x_i\}$ vs r. This exercise is identical to what you have already seen in the class (but it is essential to try to write this code on your own). Play with initial condition, x_0 and make sure you understand when the period doubling arises.

- Now write the code to plot cobweb diagram (`http://en.wikipedia.org/wiki/Cobweb_plot`). You will most likely need to use `line` command for this. Show the cobweb diagram for different values of r. Can you see how and when does period doubling arise from these diagrams?

- Are the any values of r for which the behavior become chaotic? Can you use cobweb diagram to see this?