Review Assignment: Part 1

Computer Science ICS4U

For this assignment you will be reviewing topics covered in ICS3U and exploring some interesting properties of sinusoidal functions from a visual perspective.

Prerequisite Knowledge

1. Java topics from ICS3
2. You will need to be able to draw lines and circles on the screen using JavaFX.

Background

Recall from Grade 11 math, that there are a group of functions called sinusoidal functions that have the form:

$$f(t) = a\,sin(k(t-h)) + c \tag{1}$$

where: a is the amplitude

k is related to the frequency of the wave ($f = 2\pi k/360$)

h is a horizontal shift by h units

c is a vertical shift by c units

In this assignment we will not by shifting the graph vertically or horizontally, so we will use the following simplified version:

$$f(t) = a\,sin(kt) \tag{2}$$

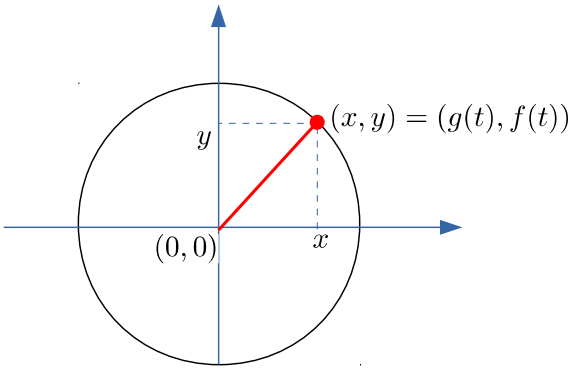
By replacing k with $\frac{360f}{2\pi}$, we get the final version:

$$f(t) = a\,sin(\frac{360f}{2\pi}t) \tag{3}$$

Similarly we will also define:

$$g(t) = a\,cos(\frac{360f}{2\pi}t) \tag{4}$$

Using these two functions, we can plot a line of length a that spins around the origin at a rotation frequency f, using the two functions defined above, one for the x value and one for the y value:



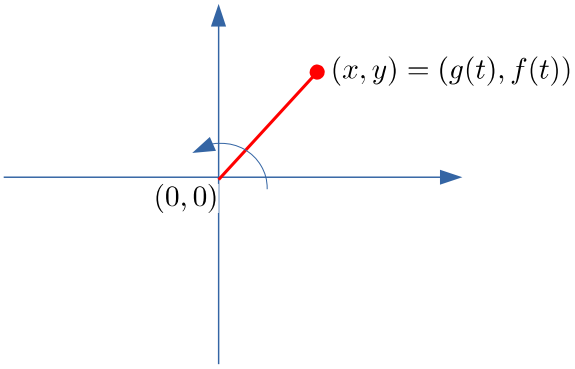
Part 1 – Drawing Circular Graphs

Using Eclipse, write a Java class called Assignment1 with the following method signatures:

public double **sin**(double a, double f, double t) : returns the function $f(t)$ as defined in eq. (3).

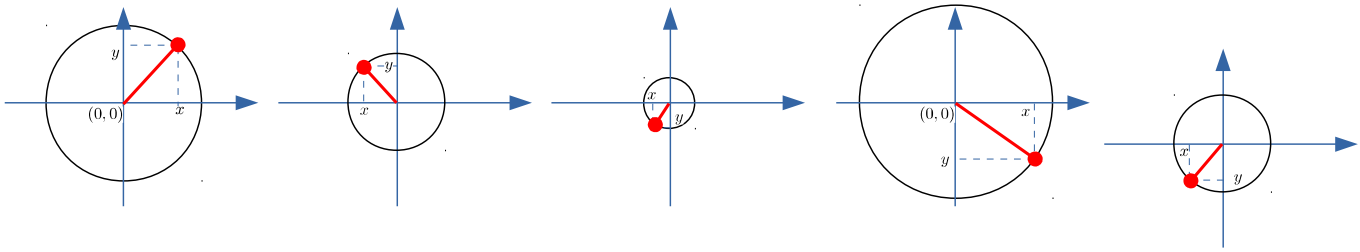
public double **cos**(double a, double f, double t) : returns the function $f(t)$ as defined in eq. (4).

Now, using JavaFX and the above functions, write a main method that will open a 600x600 pixel window and animate a spinning line of length a at a frequency f. Animate your line so that it spins around the origin which is to be located at the centre of your window.



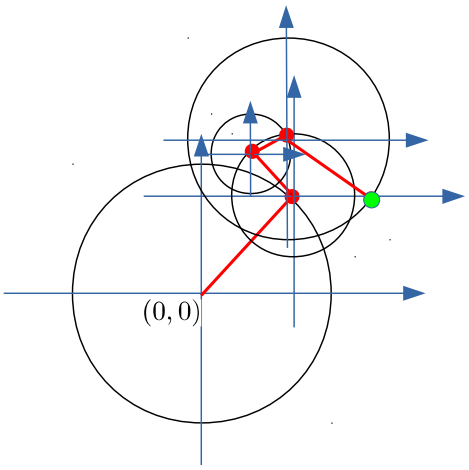
Part 2 – Drawing Using Spinning Lines

In this part you will create an array of spinning lines each with a different length and frequency of rotation:

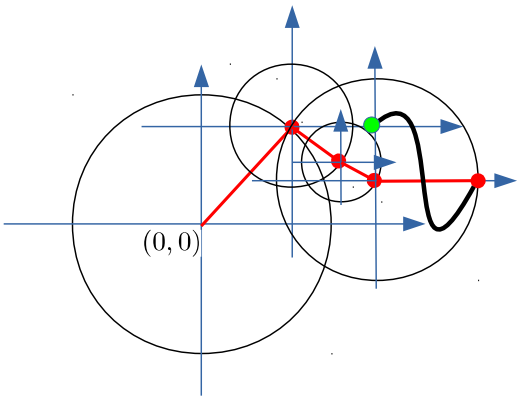


freq = 0 freq = 1 freq = 2 freq = 3 freq = 4

Then, you will add up these lines by placing the start of the next line on the tail of the previous line. The following drawing shows four such lines with frequencies $\{0, 1, 2, 3\}$ and lengths $\{a_0, a_1, a_2, a_3\}$ for a particular value of time t. Notice the end point of the last line shown in green.



If we now run this arrangement for some time, the end point will trace some path as shown below:



Modify your Java program from part 1 so that you animate each of the lines and also show the path travelled by the end point. You can randomly choose the number and lengths of each line to see what picture is drawn.

Part 3 – Drawing Functions Using Spinning Lines

In the previous part you used an array of lines to draw a random picture. However, it would be cool to be able to choose the lengths of each line so that it ends up drawing a picture that we want. Luckily there is a way to find these lengths!

The (x, y) coordinates of the approximated function evaluated at time t is:

$$x = \sum_{f=-\infty}^{\infty} c_{fx} \cdot \cos(2\pi f t) - c_{fy} \cdot \sin(2\pi f t) \qquad y = \sum_{f=-\infty}^{\infty} c_{fx} \cdot \sin(2\pi f t) + c_{fy} \cdot \cos(2\pi f t)$$

where (c_{fx}, c_{fy}) is the starting position (tail end) of a line spinning at frequency f. If needed, it's length can be easily derived using the Pythagorean theorem as:

$$|c_f| = \sqrt{c_{fx}^2 + c_{fy}^2}$$

Note that the sums are infinite but in reality you will be approximating them by the number of spinning lines that you want to use. As before in part 2) you will obtain the head position of the last spinning line by placing each line's tail on the previous line's head until you get to the last line.

Each line's starting end point (c_{fx}, c_{fy}), can be computed by the following formulas:

$$c_{fx} = \sum_{t=0}^{1} \Big[\Big(x\cos(2\pi f t) + y\sin(2\pi f t)\Big)\Big]\Delta t \qquad c_{fy} = -\sum_{t=0}^{1} \Big(x\sin(2\pi f t) - y\cos(2\pi f t)\Big)\Big]\Delta t$$

where Δt is the step size that you want to use in going from 0 to 1 and $f(t) = (x, y)$ is the function that you want to approximately reproduce using the spinning lines. The value for Δt will depend on how many samples from the original function $f(t)$ that you will be providing.

Your program will need to have a function $f(t)$ to approximate. You will have two ways of providing this function:

1. **public ArrayList loadFunction()** - creates an array of (x, y) values from a hardcoded mathematical function such as $f(t) = x^2$.
2. **public ArrayList loadFunction(String file)** – creates an array of (x, y) values by reading data coordinates from a text file of the following format:

   x_1, y_1
   x_2, y_2
   $\vdots$

Additionally you will create a Java method to draw the desired function to be approximated so that you can compare the original function with the approximated one:

public void drawFunction()

Example:
Say that we want to draw an approximation for the function $f(x) = x^2$ in the domain 0 to 2. We decide that we are going to have 11 samples as follows:

x	y
0.0	0.0
0.2	0.04
0.4	0.16
0.6	0.32
0.8	0.64
1.0	1.0
1.2	1.44
1.4	1.96
1.6	2.56
1.8	3.24
2.0	4.0

First we compute all of the $[C_{fx}, C_{fy}]$ values. For our example, to compute C_{fx} for the spinning line of frequency f, we would do something like the following:

$$c_{fx} = \sum_{t=0}^{1} \Big[\Big(x\cos(2\pi ft) + y\sin(2\pi ft)\Big)\Big]\Delta t$$

The Δt value will be the time increment that will take us from $t = 0$ to $t = 1$ in exactly n steps, where n is the number of samples in our function table. In the above example there are 11 rows so

$$\Delta t = \frac{1}{n-1} = \frac{1}{11-1} = 0.1$$

Now we can calculate C_{fx} and C_{fy}. For example, if we are interested in computing the value for a line with frequency 2, we could do it like this:

$$c_{2x} = \sum_{t=0}^{1} \Big[\Big(x\cos(2\pi 2t) + y\sin(2\pi 2t) \Big) \Big] \Delta t$$

This means that we compute the sum of all terms $\Big(x\cos(2\pi 2t) + y\sin(2\pi 2t) \Big) \Delta t$ as we vary t from 0 to 1 in increments of Δt. So the idea is something like this:

```
c_2x = 0
i = 0
Delta_t = 0.1
for all t from 0 to 1 in increments of 0.1
{
      c_2x = c_2x + (x[i]*cos(2*PI*2*t) + y[i]*sin(2*PI*2*t)) * Delta_t
      i++
}
```

You could also simultaneously calculate the value of c_{2y} using the formula for that variable:

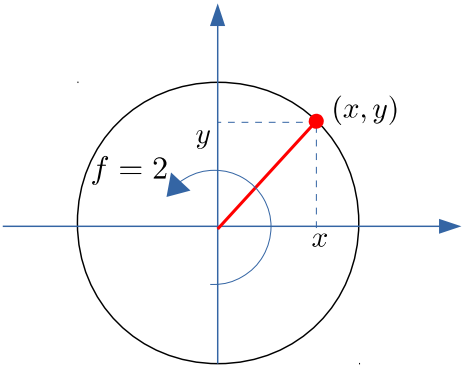
$$c_{fy} = -\sum_{t=0}^{1} \Big(x\sin(2\pi ft) - y\cos(2\pi ft) \Big) \Big] \Delta t$$

Once you have computed $[C_{fx}, C_{fy}]$, you can then calculate the starting point of the line spinning at a frequency of 2 cycles/sec:

The starting point occurs when $t = 0$, So,

$$x = c_{fx} \cdot \cos(2\pi(2)(0)) - c_{fy} \cdot \sin(2\pi(2)(0))$$
$$y = c_{fx} \cdot \sin(2\pi(2)(0)) + c_{fy} \cdot \cos(2\pi(2)(0))$$

The point (x, y) gives us the starting position of the line with frequency = 2.



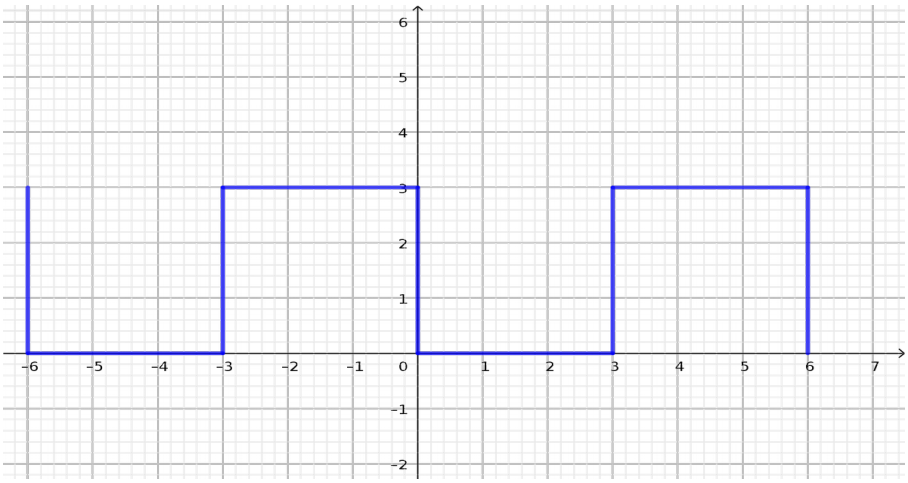
Once we have computed the above for all frequencies, we will essentially have a bunch of lines spinning at different frequencies, each with its own length and starting point. Now we just have to place them all head to tail and trace out the path of the end point of the very last line.

What to Hand In:

You will electronically submit to the Google classroom your Assignment1.java file and a Google doc answering the following questions:

Questions to Answer

1. Does the order in which you add the spinning lines make a difference? Explain.
2. What happens to your approximation if you have too few spinning lines?
3. What happens to your approximation if you have too many spinning lines?
4. What happens at the ends of your path if it is not closed?
5. What happens if your function has sharp abrupt edges, such as in the following square wave:



6. What effect does the number of original function samples have on the approximation?