This worksheet is split into two sections: Part A is a set of “traditional” maths questions to complete without a computer, while Part B involves using code to answer the same or similar questions. You can complete either section first, or swap between them; you may find that tackling the same problem using a different approach enhances your understanding of it.

# Part A

Answer the following questions on 2D vectors using pen(cil) and (graph) paper.

Pro tip: show your working – diagrams can be helpful!

1. Let and . Draw the following:
2. Evaluate the following expressions:
3. Find the angles between the following pairs of vectors and using trigonometry/`SOHCAHTOA’:
   1. and

You may find the following table useful:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

* 1. and

Exercises may include some modified questions from  
Dunn, F & Parberry, I 2011, *3D Math Primer for Graphics and Game Development*, CRC Press, Boca Raton, FL

# Part B

Fork the following Bitbucket repository so you can compile and make changes to the code:

<https://gamesgit.falmouth.ac.uk/projects/COMP270/repos/comp270-2d-geometry-workshop>

This is a basic drawing tool that allows simple shapes to be displayed against a set of axes. You don’t need to worry about what most of the code does (there are comments you can read if you’re interested); for now, the files we care about are:

* *Vector2.h* contains a class that represents a 2D vector (which might sometimes also be used to store points).
* *Scene.cpp* contains the functions that add shapes to the drawing; note the setup() function in particular.
* *Curve.h/cpp* contain a skeleton implementation for a parametric curve.

Follow the instructions below to expand the functionality of the tool.

1. The Vector2 class is not fully implemented; the addition, subtraction and scalar multiplication functions are incomplete. Finish these off and you should be able to see the answers to question 1 of part A! (Note: you’ll also need to complete the implementation in Scene::week2\_exercise1() to see parts (g) and (h)).  
   Hint 1: you can access a member variable of a class in C++ either just by using its name, or you can specify that it belongs to this instance using the keyword this, which gives a [pointer](https://www.learncpp.com/cpp-tutorial/67-introduction-to-pointers/) to the current instance – e.g. to access the member x from inside a Vector2 class, you can either just type x or use this->x.  
   Hint2: If you see a parameter to a function with an & next to it, that parameter is being [passed by reference](https://www.learncpp.com/cpp-tutorial/73-passing-arguments-by-reference/); you can treat it as a normal variable (i.e. not a pointer; you can access its members using . ) but remember that changing its value will have effects outside the function, unless it is const (in which case the value [cannot be changed](https://www.learncpp.com/cpp-tutorial/6-11a-references-and-const/)).
2. The function to find the vector magnitude is also incomplete; finish it and check the results against the answers to question 2 of part A using the completed calculations in Scene::week2\_exercise2().  
   Hint: to see the values, you’ll need to either print them or set breakpoints.
3. The Curve class is set up to allow a parametric curve to be approximated by a set of drawable lines, which should be generated in the Curve::getLines() function. Finish the implementation to create a unit circle by evaluating its parametric formula (, ) at specified intervals (the drawing code is set up so if you get the lines right, you should see it straight away).  
   Hint: you may find it helpful to add a separate function to evaluate the formula. If you’re struggling to get the lines, try implementing getPoints() to plot individual points first.
   1. A circle with radius is given by , . Add a variable to allow different sized circles to be drawn (in the Curve class and possible the Scene::addCurve() method, too).
4. Now see if you can draw some more interesting curves! You can use the same function as for the circle, or if you like, use derived classes to create different types of curves (in which case you might also like to add some variations of the Scene::addCurve() method). Some examples of curves you could try are:
   1. The [heart curve](https://mathworld.wolfram.com/HeartCurve.html)
   2. The [butterfly curve](https://mathworld.wolfram.com/ButterflyCurve.html)
   3. The [Lissajous curve](https://mathworld.wolfram.com/LissajousCurve.html), which is actually a family of curves with additional (fixed) parameters to describe different shapes.

Hint: you might need to increase the range of the axes to see some of these fully; use the c\_max\_x and c\_max\_y settings in *Application.h*.

**Post your screenshots in the forum/on your wiki page!**