COMP270
Mathematics for 3D Worlds and Simulations

Week 2 Exercises: 2D Vectors

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 $\mathbf{a} = \binom{2}{3}$ and $\mathbf{b} = \binom{1}{-2}$. Draw the following:
   a. $\mathbf{a}$
   b. $\mathbf{b}$
   c. $2\mathbf{a}$
   d. $-\frac{1}{2}\mathbf{b}$
   e. $\mathbf{a} + \mathbf{b}$
   f. $\mathbf{a} - \mathbf{b}$
   g. $\frac{1}{2}\mathbf{a} + 2\mathbf{b}$
   h. $-2\mathbf{a} - \frac{1}{3}\mathbf{b}$

2. Evaluate the following expressions:
   a. $2\binom{9}{-3}$
   b. $\frac{1}{2}\binom{4}{5}$
   c. $-\binom{-7}{1}$
   d. $\left\|\binom{-12}{5}\right\|$
   e. $\left\|\frac{1}{3}\binom{27}{-12} - \frac{1}{2}\binom{-6}{24}\right\|$
   f. $\left\|\frac{-1}{\sqrt{2}}\binom{\sqrt{2}}{2} + \frac{1}{2}\binom{1}{\frac{-2}{\sqrt{2}}}\right\|$

3. Find the angles between the following pairs of vectors $\mathbf{a}$ and $\mathbf{b}$ using trigonometry/`SOHCAHTOA':
   a. $\mathbf{a} = \binom{3}{\sqrt{3}}$ and $\mathbf{b} = \binom{1}{\sqrt{3}}$
   b. $\mathbf{a} = \binom{3}{3}$ and $\mathbf{b} = \binom{-2}{2}$

You may find the following table useful:

θ	0°	30°	45°	60°	90°
$\sin\theta$	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1
$\cos\theta$	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0
$\tan\theta$	0	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$	$\pm\infty$

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

COMP270
Mathematics for 3D Worlds and Simulations

Week 2 Exercises: 2D Vectors

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)).

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 ($x = \cos t$, $y = \sin t$) 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.
   a. A circle with radius r is given by $x = r\cos t$, $y = r\sin t$. 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:
   a. The heart curve
   b. The butterfly curve
   c. The Lissajous curve, 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!