Sheet 3

Root finding and the Manipulate command

3.1 Numerical Root Finding

While **NSolve** (which we met in sheet 1) works well at finding the solution to polynomial equations, it is not able to find solutions of more complicated functions. For example, **NSolve** is unable to find a solution to the equation

$$q(x) = e^{\sin x - \cos(3x)} - x = 0,$$

, so it gives an error message and returns the input unchanged:

In[69]:=
```
q[x_] := Exp[Sin[x] - Cos[3 x]] - x
NSolve[q[x] == 0,x]
```

NSolve::nsmet: This system cannot be solved with the methods available to NSolve. ≫

Out[69]=
```
NSolve[e^(-Cos[3 x] + Sin[x]) - x == 0, x]
```

The function **FindRoot** searches for a root using the Newton-Raphson method, here starting from $x = 5$:

In[70]:=
```
root = FindRoot[q[x], {x, 5}]
```

Out[70]=
```
{x → 1.72401}
```

Note that you only provide an expression **q[x]** to **FindRoot**, rather than a full equation **q[x]==0** as we did for **NSolve**.We can check that a root was found:

In[71]:=
```
q[x /. root]
```

Out[71]=
```
2.22045*10^-16
```

The value of q here is less than 10^{-15}, which is the default working precision for numerical operations in Mathematica. The function **Chop** can be used to replace very small numbers (by default less than 10^{-10}) by zero:

In[72]:=
```
q[x /. root] // Chop
```

Out[72]=
```
0
```

Sometimes **FindRoot** will get fail to find a root because the iteration gets 'stuck' in a local minimum, and in this case it will return an error as well as a numerical value:

In[73]:=
```
notRoot = FindRoot[q[x], {x, -2}]
```

FindRoot::lstol: The line search decreased the step size to within tolerance specified by AccuracyGoal and PrecisionGoal but was unable to find a sufficient decrease in the merit function. You may need more than MachinePrecision digits of working precision to meet these tolerances. ≫

Out[73]= {x → 0.136425}

Note that in this case **FindRoot** returns some value of x, but it is not a root since $q(x) \neq 0$:

```
In[74]:= q[x /. notRoot]
```

Out[74]= 0.321336

We can see at which points x the function was evaluated at by using the **EvaluationMonitor** option (with a delayed rule **:>** instead of **->**):

```
In[75]:= x0 = 4;
         {res, {evx}} = Reap[FindRoot[q[x], {x, x0},
         EvaluationMonitor :> Sow[x]]]
```

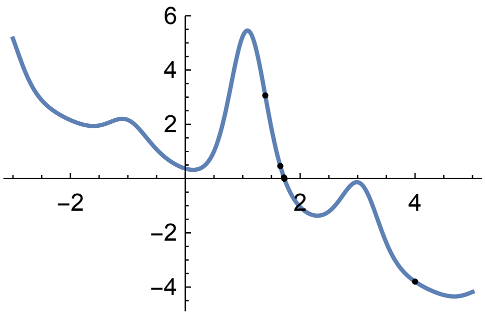
```
Out[75]= {{x → 1.72401},
         {{4., 1.39251, 1.65405, 1.71706, 1.72393, 1.72401, 1.72401}}}
```

Here the pair of functions **Sow** and **Reap** are used to propagate expressions during the evaluation of the **FindRoot**, and the result is stored in two variables, **res** (the result) and **evx** (the points where q was evaluated at).

We can then plot these points to see where the function was evaluated:

```
In[76]:= Show[Plot[q[x], {x, -3, 5}],
         ListPlot[Transpose[{evx, q[evx]}], PlotStyle ->{Small, Black}] ]
```

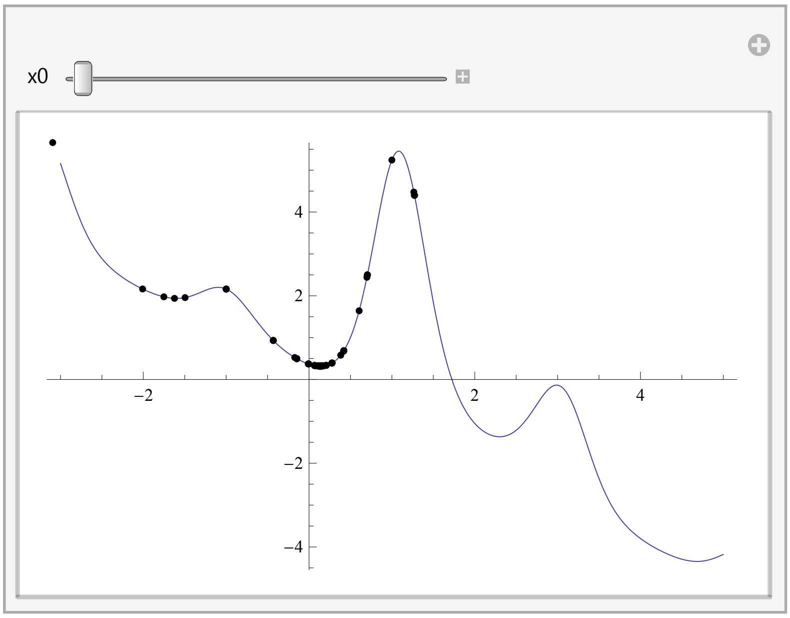
Out[76]=



Taking these two bits of code, we can use the Manipulate environment to vary x_0 dynamically (copy this code into your Mathematica workbook, but it is not necessary to understand it all!)

```
In[77]:= Manipulate[{res, {evx}} =
         Reap[FindRoot[q[x], {x, x0}, EvaluationMonitor :> Sow[x]]] // Quiet;
         Show[Plot[q[x], {x, -3, 5}, Epilog -> Point[{x, q[x]} /. res]],
            ListPlot[{Transpose[{evx, q[evx]}]}, PlotStyle -> {{Small, Black}}],
            Epilog -> {Red, PointSize[Large], Point[{x, q[x]} /. res]},
            GridLines -> {{x0}, None}], {x0, -3, 5}]
```

Out[77]=



The function **Quiet** at the end of the first line is used to suppress the error message when **FindRoot** gets stuck in the local minima and the semicolon is needed to join multiple expressions into a sequence of instructions. You can see that for this function, the default settings will often get stuck in a local minimum.

Exercises

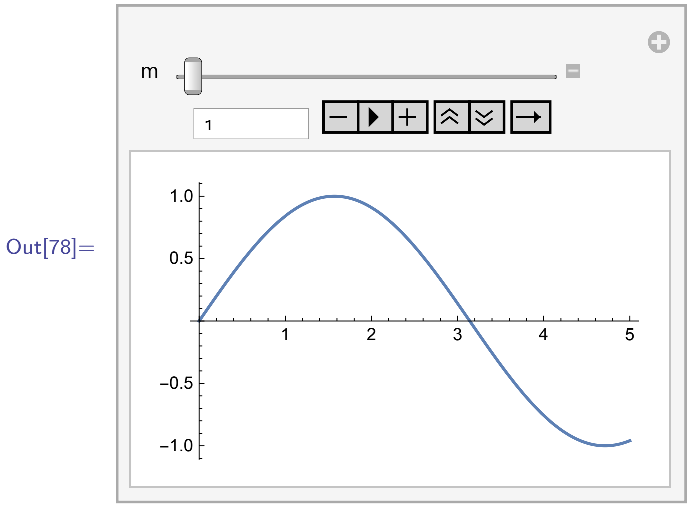
Consider the function $f(x) = x^3 - x^2 - 6x - 2$.

1. Define the function `f[x_]`.
2. Plot a graph of $f(x)$ for $-4 \leq x \leq 4$.
3. By observing your graph, estimate where the solutions to $f(x) = 0$ are.
4. Use the command `NSolve[f[x]==0,x]` to find values for the positions of the roots.
5. Use `FindRoot[f[x]==0,{x,x0}]` to search for the roots of $f(x) = 0$ by choosing different values of `x0`, ensuring that you find all the roots.
6. Repeat this process for a different cubic, e.g $g(x) = 2x^3 + x^2 - 5x + 1$, although you can choose your own.
7. Find all the roots of $\sin x - \cos x$ that lie within $-5 \leq x \leq 5$.

3.2 Manipulate

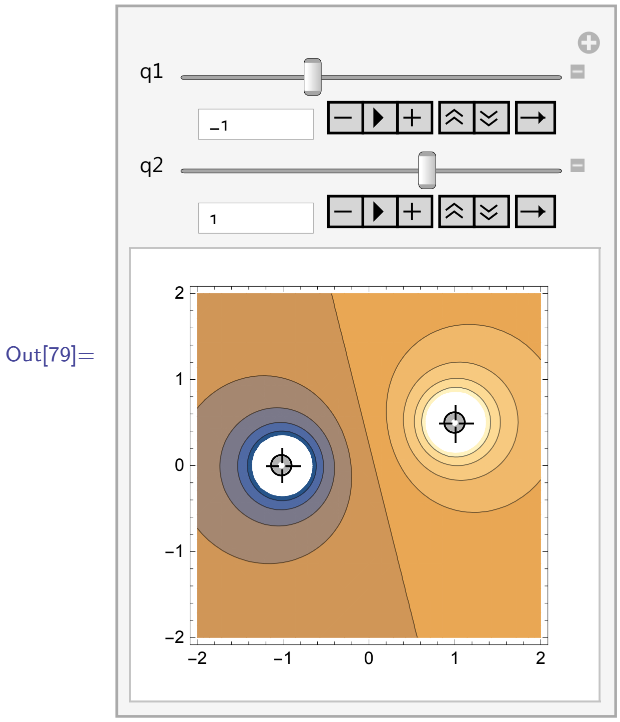
Manipulate can be used in simpler cases, for example:

In[78]:=
```
Manipulate[Plot[Sin[m x], {x, 0, 5}], {m, 1, 20}]
```



Manipulate is a very useful tool, allowing you to see how varying parameters affect the solution, particularly when used with graphics. For instance, here is a **ContourPlot** of an electrostatic potential built from point charges, where you can vary the strength of the charges and move them around:

In[79]:=
```
Manipulate[
 ContourPlot[q1/Norm[{x,y} - p1] + q2/Norm[{x,y} - p2], {x,-2,2},
 {y,-2,2}],
 {{q1, -1}, -3, 3}, {{q2, 1}, -3, 3},
 {{p1, {-1, 0}}, Locator}, {{p2, {1, 0.5}}, Locator}]
```



ContourPlot shows the contours of the function of x and y. As you move the sliders or locators around, a 'rough' version of the plot is generated, which is refined when you let go. Note that both the variables given numbers in the **ContourPlot** (**x,y**) and those given values in the **Manipulate** (**p1,p2,q1,q2**) are coloured green to show they have values within the context of that piece of code.

Advanced Exercise

Adapt the above code to handle a third point charge. You'll need to edit the function inside the **`ContourPlot`**, as well as add two more variables **q3** and **p3** into the Manipulate. Make sure you put the new point **p3** at a new position!

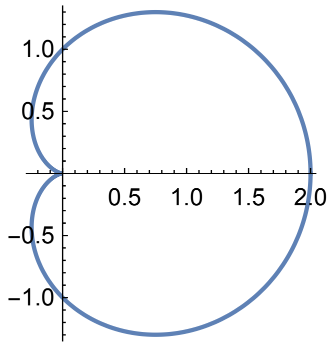
3.3 Areas, Lengths and Volumes : Polar Coordinates

Consider the polar curve known as the cardioid, which is given by $r = k(1 + \cos\theta)$. We define a function for this curve, `h[th_]:=k(1+Cos[th])`.

We can plot this curve for a fixed value of k by using a replacement rule to specify k,

```
In[80]:= PolarPlot[h[th]/.k->1,{th,0,2 Pi}]
```

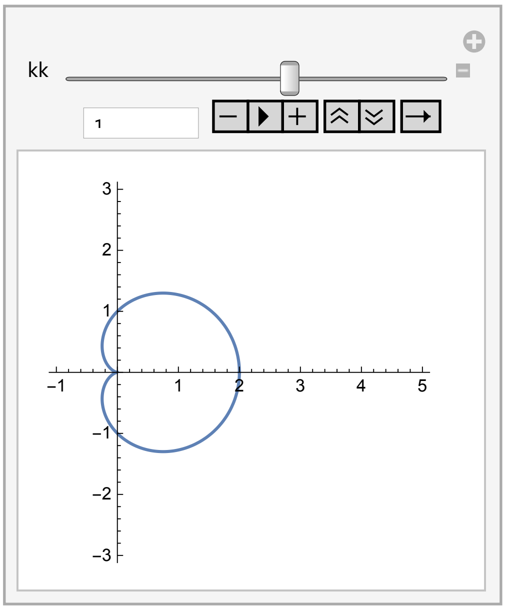
Out[80]=



Or we might wish to use **`Manipulate`** to vary it. Due to the way that **`Manipulate`** works we need to make a dummy variable to use the replacement rule (as it can't 'see' that **h** contains **k**),

```
In[81]:= Manipulate[PolarPlot[h[th] /. k -> kk, {th, 0, 2 Pi},
           PlotRange -> {{-1, 5}, {-3, 3}}], {{kk, 1}, 0, 3}]
```

Out[81]=



Changing the value of k scales the size of the cardioid. How could you change the definition of **h** to rotate the cardioid as k is varied?

The area enclosed by a polar curve is given by

$$A = \frac{1}{2}\int_a^b r^2 d\theta,$$

and so for the general cardioid, we can use **`Integrate`** to find the area:

```
In[82]:= 1/2 Integrate[h[th]^2,{th,0,2 Pi}]
```

Out[82]= $\frac{3\ k^2\ Pi}{2}$

The arclength of a polar curve is given by

$$S = \int_a^b \sqrt{r^2 + (dr/d\theta)^2}\, d\theta.$$

Again, we can use **Integrate** to find the arclength:

```
In[83]:= Integrate[Sqrt[h[th]^2 + h'[th]^2] , {th, 0, 2 Pi}]
```

Out[83]= $8\ \sqrt{k^2}$

Note that Mathematica returns the most general form, which involves $\sqrt{k^2}$. It seems like it should be able to automatically simplify that to be k, but for negative numbers that is not true! So maybe it should be $|k|$, but that is only true for real numbers, not complex numbers. If we explicitly tell Mathematica that k is positive, it will make the appropriate simplifications:

```
In[84]:= Integrate[Sqrt[h[th]^2 + h'[th]^2] , {th,0,2 Pi},
         Assumptions-> k > 0]
         Simplify[Integrate[Sqrt[h[th]^2 + h'[th]^2] , {th, 0, 2 Pi}],
         k > 0]
```

Out[84]= 8 k
8 k

Exercises

1. Plot the section of a logarithmic spiral, $r = e^{\theta/10}$, for $0 \leq \theta \leq \frac{3\pi}{2}$. Find the area under the curve and the arclength.

2. Plot the section of an Archimedes' spiral, $r = \theta/2$, for $0 \leq \theta \leq \frac{3\pi}{2}$. Find the area under the curve and the arclength.

3.4 Areas, Lengths and Volumes : Cartesian Coordinates

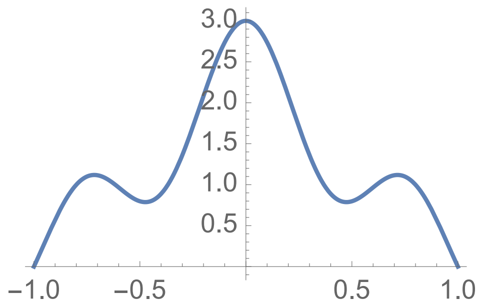
Consider the curve $y = (1 - x^2)(2 + \cos(7x))$ for $-1 \leq x \leq 1$. We first define a function

```
In[85]:= y[x_] := (1 - x^2) (2 + Cos[7 x])
```

We can plot the curve with

```
In[86]:= Plot[y[x], {x, -1, 1}]
```

Out[86]=



The arclength of a curve $y = f(x)$ in Cartesian coordinates is given by

$$S = \int_a^b \sqrt{1 + \left(\frac{df}{dx}\right)^2}\, dx$$

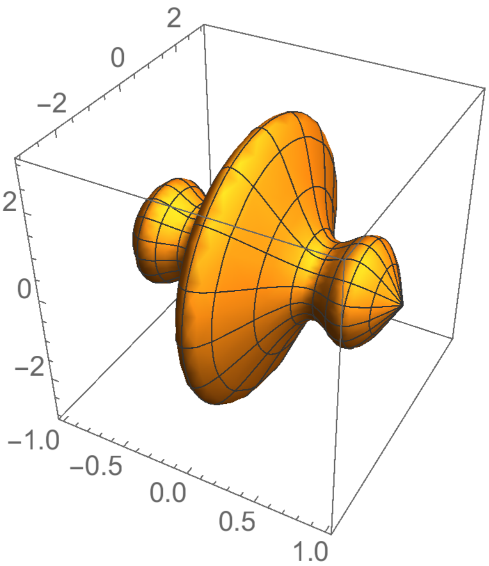
We can use the **NIntegrate** function to find the arclength:

```
In[87]:= NIntegrate[Sqrt[1+y'[x]^2], {x, -1, 1}]
Out[87]= 7.78374
```

We can generate a 3-dimensional solid by revolving the area beneath this portion of curve around the $x-$axis:

```
In[88]:= RevolutionPlot3D[y[x], {x, -2, 2}, RevolutionAxis -> {1, 0, 0},
         BoxRatios -> {1, 1, 1}, PlotRange -> All]
```

Out[88]=



The **RevolutionAxis** option here specifies that the $x-$axis is the one to revolve around (as the default is the z axis), while **BoxRatios** ensures the surrounding box is a cube.

The volume of a solid of revolution formed by revolving a curve $y = f(x)$ around the $x-$axis is given by

$$V = \pi \int_a^b f(x)^2 \, dx.$$

We can use **NIntegrate** to work out the enclosed volume,

```
In[89]:= Pi NIntegrate[y[x]^2, {x, -1, 1}]
Out[89]= 14.5195
```

The curved surface area of the solid of revolution formed by revolving a curve $y = f(x)$ around the $x-$axis

$$A = 2\pi \int_a^b f(x)\sqrt{1 + \left(\frac{df}{dx}\right)^2} \, dx.$$

Again, we use **NIntegrate** to calculate this (using the short form for the derivative):

```
In[90]:= 2 Pi NIntegrate[y[x] Sqrt[1+ y'[x]^2], {x, -2, 2}]
Out[90]= 68.0435
```

Exercise

Repeat the above process for the curve $y = \sin(3x)^2 e^{-x}$ for $0 \leq x \leq \pi$.

End

You now have all the material needed to answer the remaining three questions on the coursework.