Mathematica Project 1 Help

Throughout the semester, we will learn to use *Mathematica* to solve various problems related to MULTI-VARIATE CALCULUS. First, we will mention some basic features of *Mathematica,* and then do some exercises with **parametric curves.**

Let's consider the function f(x) = $x^3 + 2x^2 - 3$. We can enter functions into *Mathematica* with the syntax below.

```
f[x_] := x^3 + 2 x^2 – 3
```

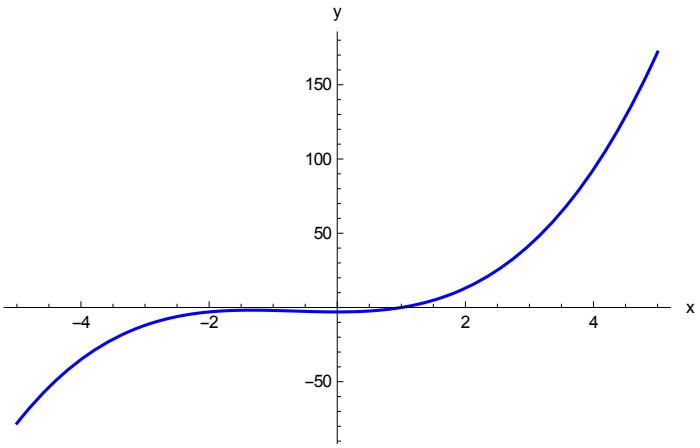
Notice that function arguments are ALWAYS placed inside square brackets. When we first define a function, each function argument is followed by an underscore. After we declare the function name and arguments, we follow the up with a semi-colon and an equal sign. This tells *Mathematica* that we are defining a new function. After this, we simply tell *Mathematica* what to do with the function arguments! We will learn some of the basic functions built into Mathematica as we go. Also, you have to press SHIFT + ENTER simultaneously to enter a command in *Mathematica*. If you only press ENTER, you will get a new input line, but *Mathematica* won't evaluate anything until SHIFT + ENTER are pressed together.

The first thing to note about *Mathematica* is that EVERY BUILT-IN FUNCTION STARTS WITH A CAPITAL LETTER. As mentioned above, the arguments for this function are placed inside square brackets (for instance, Sin[x], Cos[x], Exp[x], etc).

The Solve command instructs *Mathematica* to solve an equation (or a system of equations) for the specified variables. Notice the use of the double equal sign in the equation. In *Mathematica*, = (a single equals) is used to assign values to variables. We have already seen the use of := to define functions. The double equals, ==, tells *Mathematica* to check whether one expression equals another. One of the most common mistakes in *Mathematica* programming is using a single = when you should really use ==. If you make this mistake, often the only thing to do is to correct it, save your file, close the program completely, and then reload your file.

We will now produce a plot of our function f. Conveniently, the command to produce a plot is Plot.

```
Plot[f[x], {x, −5, 5}, PlotStyle → Blue, AxesLabel → {"x", "y"}]
```



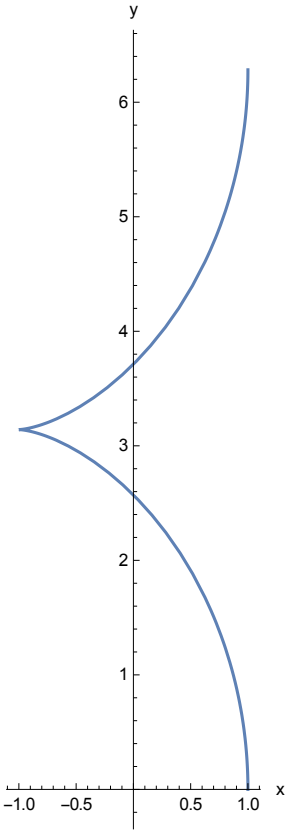
Once again, EVERY BUILT-IN FUNCTION IN *Mathematica* STARTS WITH A CAPITAL LETTER. The first argument of Plot is the function (or functions) you want to plot. The second argument is a list containing the function variable followed by the a specification of the range you want plotted. Officially, these are the only arguments you need to give Plot to produce a graph. The arguments after the first two are optional. The first specifies the color of the plot while the second gives labels for the axes. Notice that these options are specified by replacement rules. The way to get the arrow symbol is to type a dash, - , directly followed by greater than, >. Mathematica will then convert -> to the arrow symbol. By the way, copying and pasting code into Mathematica often results in errors because the arrow (and other special symbols) may not paste correctly!

Parametric Curves.

Exercise 1.

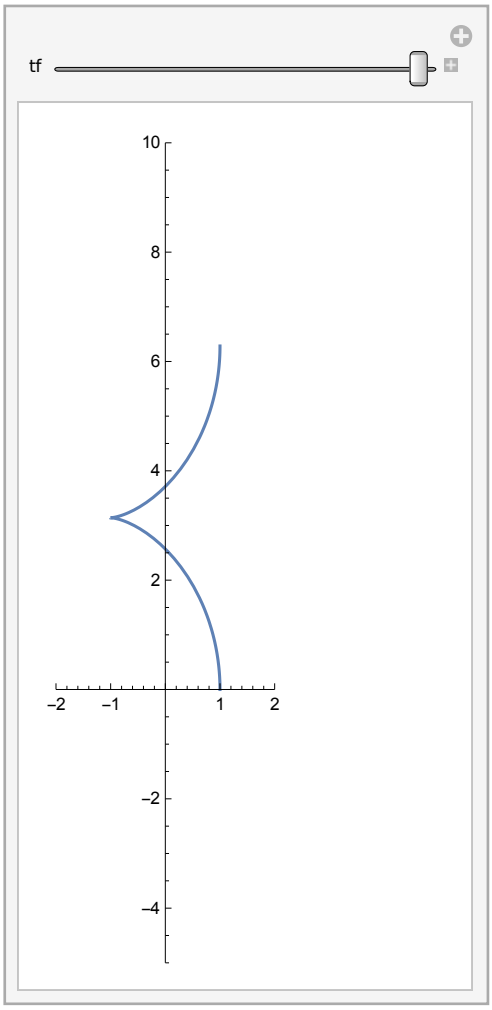
Consider the parametric curve x = cos t, y = t + sin t, $0 \le t \le 2\pi$, and produce its plot in *Mathematica*:

```
(* put your comments here *)
ParametricPlot[{x = Cos[t], y = t + Sin[t]}, {t, 0, 2 Pi}, AxesLabel → {"x", "y"}]
```



Now plot the same curve embedded in a manipulate box with a slider -- it is useful to allow us to trace the path of the particle described by the parametric equations.

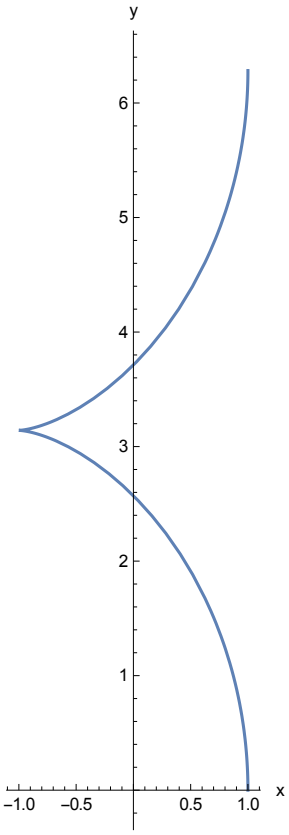
```
Manipulate[
 ParametricPlot[
   {x = Cos[t], y = t + Sin[t]}, {t, 0, tf}, PlotRange → {{-2, 2}, {-5, 10}}
 ],
   {tf, .01, 2 Pi}
]
(* move the slider to trace the path *)
```



Exercise 2.

Consider the same parametric curve as in the previous exercise. You can define a curve in *Mathematica* like this:

```
Clear[x, y, t];
(* clears variables from the listed symbols if you used them before  -
 not needed here, it is just an example *)
x[t_] := Cos[t];
y[t_] := t + Sin[t];
(* plot the curve again: *)
ParametricPlot[{x[t], y[t]}, {t, 0, 2 Pi}, AxesLabel → {"x", "y"}]
```



Now let us calculate the slope at t = π/2 using formula for the derivative of a parametric curve (confirm this result by hand):

```
y'[Pi / 2] / x'[Pi / 2]
```

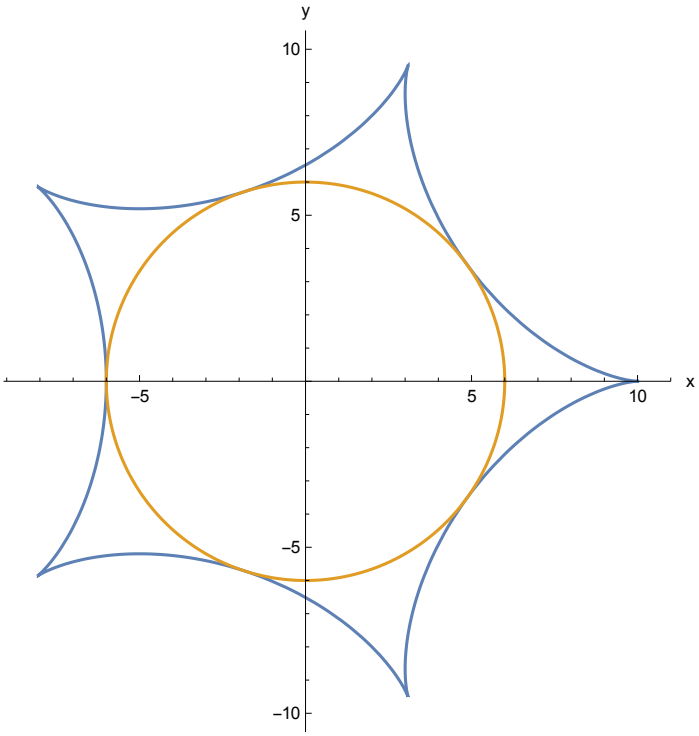
```
-1
```

Exercise 3. (Intersection and Collision Points.)

Points where two parametric curves cross each other are called intersection points. Collision points are intersection points at which the parameter in both sets of parametric equations have the same value.

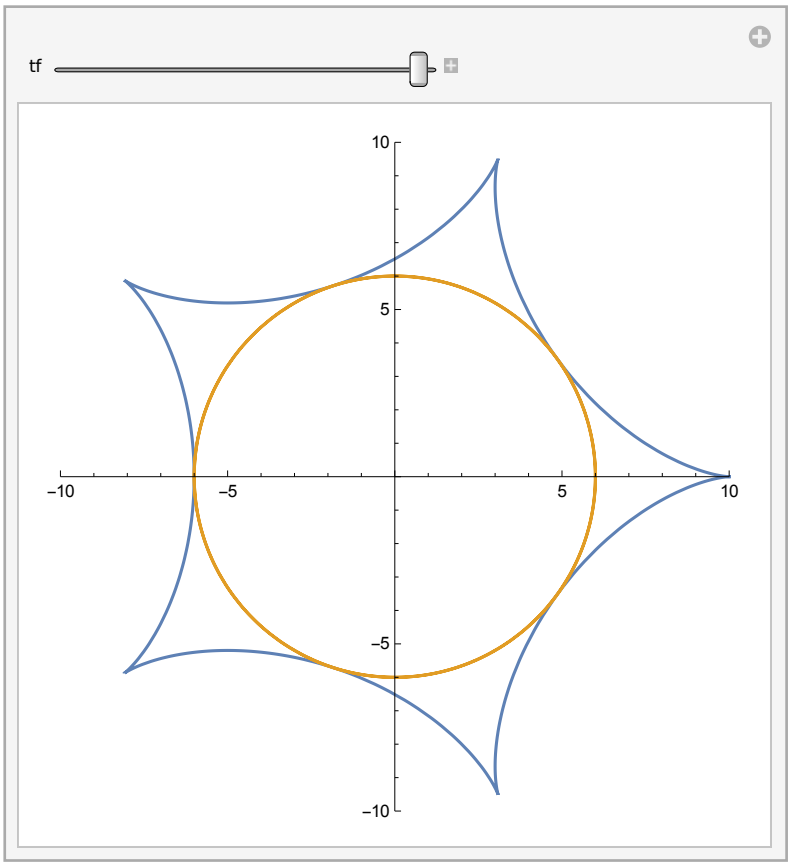
Consider two curves: the hypocycloid given by x = 8 cos t + 2 cos 4t, y = 8 sin t - 2 sin 4t, and the circle given by x = 6 cos t, y = 6 sin t. Let us plot them together:

```
ParametricPlot[
 {
  (* hypocycloid *){x = 8 * Cos[t] + 2 * Cos[4 * t], y = 8 * Sin[t] - 2 * Sin[4 * t]},
  (* circle *){x = 6 * Cos[t], y = 6 * Sin[t]}
 },
 {t, 0, 2 Pi}, AxesLabel → {"x", "y"}
]
```



Now let us draw the same curves in a box with a slider and decide which points are intersection points and which ones are collision points by tracing the curves using the slider:

```
Manipulate[
 ParametricPlot[
  {
   {x = 8 * Cos[t] + 2 * Cos[4 * t], y =  8 * Sin[t] - 2 * Sin[4 * t]},
   {x = 6 * Cos[2 t], y = 6 * Sin[2 t]}
  },
  {t, 0, tf}, PlotRange → {{-10, 10}, {-10, 10}}
 ],
   {tf, .01, 2 Pi}
]
```



(There are 5 intersection points and 0 collision points.)

Another example: consider two lines given by x = t - 2, y = t - 1, and x = 2 - t, y = 2t - 3. Repeat the experiment with the function “Manipulate” and determine any intersection and collision points.
(it is easy to see that the lines have one intersection point (0,1) which is also a collision point. Confirm this conclusion algebraically.)

```
Manipulate[
 ParametricPlot[
  {
   {x = t - 2, y = t - 1}, {x = 2 - t, y = 2 t - 3}
  },
  {t, 0, tf}, PlotRange → {{-3, 3}, {-5, 5}}
 ],
 {tf, 0.01, 5}
]
```