Integrating Equations of Motion in Mathematica

Gary L. Gray
Assistant Professor
Engineering Science and Mechanics
The Pennsylvania State University
227 Hammond Building
University Park, PA 16802
gray@engr.psu.edu

Double-click the cell brackets on the right to open and close them.

Introduction

There is really only one thing you need to know about *Mathematica* — it can do almost *anything* you would want to do mathematically. Of course, getting it to do *anything* is quite another matter. What you will learn here is probably much less than 1% of what *Mathematica* is capable of.

This is a quick tutorial on how to use *Mathematica* for many of problems you will be solving this semester. For example, you will numerically solve ordinary differential equations (equations of motion), solve systems of algebraic equations, and plot many types of functions. If you have suggestions, comments, or corrections, please send them to me at the above email address.

Executing Commands in *Mathematica*

Commands are entered in "cells". The vertical lines you see on the right are cell brackets and they define boundaries between groups of objects. Each cell has associated with it a certain set of attributes, but you only need be concerned with a couple of them. The cell containing this paragraph is a text cell as you can see in the toolbar at the top of the window. A text cell is used for adding comments and explanation to a notebook and is not *executable*. You can only do mathematics in *Mathematica* withing executable cells. Any time you start a new cell, it is an "Input cell" by default and *Mathematica* commands can be executed there since it is executable. Below you see an example of an input cell with a *Mathematica* command and the resulting output cell. The command is executed by placing the cursor *anywhere within the cell* and pressing either **Shift-Return** or **Enter**.

```
2 + 2 Cos[3.1]
```

```
0.0017297
```

When you execute the very first command in a *Mathematica* notebook (this is a *Mathematica* notebook and it runs in an application called the "front end"), *Mathematica* must start a program called the "kernel" which is the application that does all the computation. Therefore, you may experience a small delay the first time you execute an input cell as you wait for the kernel to start up.

Note that *Mathematica* knows about constants such as π and E (the **N** command tells *Mathematica* to return a numerical value). You can get the "pi" from the palettes on the right or using some special keystrokes.

```
N[π]
```

```
3.14159
```

```
N[E]
```

```
2.71828
```

Without the **N**, you simply get

```
π - E
```

```
-E + π
```

since *Mathematica* thinks of these as *exact* constants. This is an important thing to keep in mind about *Mathematica*.

Mathematica Syntax

You can use any name you like as a variable name, but it cannot have spaces, dashes, or begin with a number. The equals sign "=" assigns one thing to another. For example, this command sets *q* equal to 4. A semicolon suppresses the output.

```
q = 4;
```

We now set *p* equal to 5.

```
p = 5;
```

If you ask *Mathematica* what *p* is, it returns what you would expect.

```
p
```

```
5
```

```
p + q
```

```
9
```

Mathematica never forgets these values for *p* and *q* until you quit and restart the program. (Actually, there are ways to get it to forget them, but we won't get into them here.) In addition, I highly recommend that you start all variable names with a lower case letter since ***all*** *Mathematica* commands begin with capital letters and then there is no was you can use a variable name that is also a *Mathematica* command.

Multiplication is represented by a space or an asterisk.

```
p q
```

```
20
```

```
p * q
```

```
20
```

□ Fancy Formatting

Mathematica is capable of some pretty fancy formatting. Note that all *Mathematica* commands start with capital letters (*e.g.*, cosine) and also note that the argument of each *Mathematica* command is contained in square brackets (more on this later).

$$z = \int_{-1}^{\infty} \frac{\text{Cos}[a\,x]}{x^2 + \sqrt{y - z^3}}\, dx$$

You can use the Palettes submenu of the File menu to get a whole bunch of palettes for formatting.

The Double Equals Sign

Mathematica uses the double equals sign "==" to equate one thing to another. This is not the same as an assignment given by the single equals sign. For example, say way want to solve the simultaneous equations $x + y = 3$ and $x - y = 4$. You would enter the equations in the following way (notice that I have put more than one command in a single cell bracket):

```
eq1 = x + y == 1
eq2 = x - y == 2
```

```
x + y == 1
```

```
x - y == 2
```

I then solve the equations using the **Solve** command:

```
xysoln = Solve[{eq1, eq2}, {x, y}]
```

$$\left\{\left\{x \to \frac{3}{2},\ y \to -\frac{1}{2}\right\}\right\}$$

The output from the solution is a list of *replacement rules*. A replacement rule has the form:

```
left -> right
```

That is, anywhere **left** appears, replace it with **right**. Here is how you use them.

```
x /. xysoln
```

$$\left\{\frac{3}{2}\right\}$$

```
y /. xysoln
```

$$\left\{-\frac{1}{2}\right\}$$

```
x y /. xysoln
```

$$\left\{-\frac{3}{4}\right\}$$

This says to multiply *x* by *y*. Then the **/.** command tells *Mathematica* to substitute the replacement rule **xysoln** into *x y*. For now, don't worry about the extra square brackets that are hanging around.

Let's have a look at one more example.

In[1]:=
```
list = {d -> 4, e -> 3.2, f -> 1.2}
```

Out[1]=
```
{d → 4, e → 3.2, f → 1.2}
```

In[2]:=
```
d /. list
```

Out[2]=
```
4
```

In[3]:=
```
d + e + f + g /. list
```

Out[3]=
```
8.4 + g
```

Notice it didn't substitute anything for **g** since it was not in our list.

- **You can get information about any command by entering a question mark followed by the command.**

Here is information on the **Solve** command we used above.

```
? Solve
```

```
Solve[eqns, vars] attempts to solve an equation or set of equations
  for the variables vars. Solve[eqns, vars, elims] attempts to
  solve the equations for vars, eliminating the variables elims.
```

Even more information can be obtained by using the double question mark.

```
?? Solve
```

```
Solve[eqns, vars] attempts to solve an equation or set of equations
  for the variables vars. Solve[eqns, vars, elims] attempts to
  solve the equations for vars, eliminating the variables elims.
```

```
Attributes[Solve] = {Protected}

Options[Solve] = {InverseFunctions -> Automatic,
   MakeRules -> False, Method -> 3, Mode -> Generic, Sort -> True,
   VerifySolutions -> Automatic, WorkingPrecision -> Infinity}
```

Turn off annoying *Mathematica* warnings.

```
fred = 12
```

```
12
```

```
freda = 13
```

```
– General::spell1 :
   Possible spelling error: new symbol name "freda" is similar to existing symbol "fred".
```

```
13
```

These keep *Mathematica* from complaining when you name a variable **freda** when you already have a variable named **fred**. I execute these commands in almost every *Mathematica* notebook I create.

```
Off[General::spell]
Off[General::spell1]
```

Now watch what happens:

```
joe = 3
```

```
3
```

```
joey = 1
```

```
1
```

Doing Vector Operations

To do the type of operations we need to do in this class, you first need to load some *external packages*. These are packages that ship with *Mathematica*, but are not loaded automatically.

```
Needs["Calculus`VectorAnalysis`"]
```

■ Defining a Vector

Vectors are defines as *lists* enclosed in curly brackets.

```
f = {1, 2, 3}
```

```
{1, 2, 3}
```

```
g = {4, 5, 6}
```

```
{4, 5, 6}
```

```
dot = DotProduct[f, g]
```

```
32
```

```
cross = CrossProduct[f, g]
```

```
{-3, 6, -3}
```

These operations also work on more than numbers.

```
DotProduct[{l, m, n}, {r, s, t}]
```

```
l r + m s + n t
```

```
CrossProduct[{l, m, n}, {r, s, t}]
```

```
{-n s + m t, n r - l t, -m r + l s}
```

□ Picking out components of a vector or list.

You can pick out one of the components of the vector (or, equivalently, a part of a list) by using the following notation.

```
cross[[1]]
```

```
-3
```

```
cross[[2]]
```

```
6
```

You get the idea.

Now, recall **xysoln**.

```
xysoln
```

$$\left\{\left\{x \to \frac{3}{2},\ y \to -\frac{1}{2}\right\}\right\}$$

It is really a *list of lists*. Therefore:

```
xysoln[[1]]
```

$$\left\{x \to \frac{3}{2},\ y \to -\frac{1}{2}\right\}$$

Therefore we need to do the following:

```
xysoln[[1, 1]]
```

$$x \to \frac{3}{2}$$

```
xysoln[[1, 2]]
```

$$y \to -\frac{1}{2}$$

```
xysoln[[2, 1]]
```

– Part::partw : Part 2 of $\left\{\left\{x \to \frac{3}{2},\ y \to -\frac{1}{2}\right\}\right\}$ *does not exist.*

$$\left\{\left\{x \to \frac{3}{2},\ y \to -\frac{1}{2}\right\}\right\}[[2,\ 1]]$$

Get it? One more example. Form a another type of list (actually, this is a 2 by 2 matrix):

In[4]:=
```
mat = {{1, 2}, {3, 4}}
```

Out[4]=
```
{{1, 2}, {3, 4}}
```

In[5]:=
```
mat[[1]]
```

Out[5]=
```
{1, 2}
```

In[6]:=
```
mat[[1, 2]]
```
Out[6]=
```
2
```

In[7]:=
```
mat[[2]]
```
Out[7]=
```
{3, 4}
```

In[8]:=
```
mat[[2, 1]]
```
Out[8]=
```
3
```

Solving Systems of Algebraic Equations

We have already seen this, but *Mathematica* is *very* powerful!!!

■ Define 8 equations.

These eight equations are the equations of motion of the slider-crank mechanism inside an IC engine.

```
eq1 = N + Cx == 0
eq2 = Cy - P == mC aC
eq3 = By - Cy == mBC aDy
eq4 = Bx - Cx == mBC aDx
eq5 = Cx L Cos[ϕ] - Cy L Sin[ϕ] == IBC αBC + mBC aDy a Sin[ϕ] - mBC aDx a Cos[ϕ]
eq6 = Ax - Bx == 0
eq7 = Ay - By == 0
eq8 = Bx r Cos[ϑ] + By r Sin[ϑ] + T == 0
```

$$N + C_x == 0$$

$$-P + C_y == a_C\, m_C$$

$$B_y - C_y == a_{D_y}\, m_{BC}$$

$$B_x - C_x == a_{D_x}\, m_{BC}$$

$$L\, \text{Cos}[\phi]\, C_x - L\, \text{Sin}[\phi]\, C_y == -a\, \text{Cos}[\phi]\, a_{D_x}\, m_{BC} + a\, \text{Sin}[\phi]\, a_{D_y}\, m_{BC} + I_{BC}\, \alpha_{BC}$$

$$A_x - B_x == 0$$

$$A_y - B_y == 0$$

$$T + r\, \text{Cos}[\vartheta]\, B_x + r\, \text{Sin}[\vartheta]\, B_y == 0$$

■ Solve the 8 equations of motion.

These are the various forces in the problem.

```
soln =
 Solve[{eq1, eq2, eq3, eq4, eq5, eq6, eq7, eq8}, {Ax, Bx, Ay, By, Cx, Cy, N, T}]
```

$$\left\{\left\{T \to r \text{Sin}[\theta]\ (-P - a_{D_y}\ m_{BC} - a_C\ m_C) + \frac{1}{L}\ (r \text{Cos}[\theta]\ (a\ a_{D_x}\ m_{BC} - L\ a_{D_x}\ m_{BC} - \text{Sec}[\phi]\ I_{BC}\ \alpha_{BC} - L\ P\ \text{Tan}[\phi] - a\ a_{D_y}\ m_{BC}\ \text{Tan}[\phi] - L\ a_C\ m_C\ \text{Tan}[\phi])),\right.\right.$$
$$A_x \to -\frac{1}{L}(a\ a_{D_x}\ m_{BC} - L\ a_{D_x}\ m_{BC} - \text{Sec}[\phi]\ I_{BC}\ \alpha_{BC} - L\ P\ \text{Tan}[\phi] - a\ a_{D_y}\ m_{BC}\ \text{Tan}[\phi] - L\ a_C\ m_C\ \text{Tan}[\phi]),$$
$$A_y \to P + a_{D_y}\ m_{BC} + a_C\ m_C,$$
$$N \to -\frac{-a\ a_{D_x}\ m_{BC} + \text{Sec}[\phi]\ I_{BC}\ \alpha_{BC} + L\ P\ \text{Tan}[\phi] + a\ a_{D_y}\ m_{BC}\ \text{Tan}[\phi] + L\ a_C\ m_C\ \text{Tan}[\phi]}{L}, B_x \to -\frac{1}{L}$$
$$(a\ a_{D_x}\ m_{BC} - L\ a_{D_x}\ m_{BC} - \text{Sec}[\phi]\ I_{BC}\ \alpha_{BC} - L\ P\ \text{Tan}[\phi] - a\ a_{D_y}\ m_{BC}\ \text{Tan}[\phi] - L\ a_C\ m_C\ \text{Tan}[\phi]),$$
$$C_x \to -\frac{a\ a_{D_x}\ m_{BC} - \text{Sec}[\phi]\ I_{BC}\ \alpha_{BC} - L\ P\ \text{Tan}[\phi] - a\ a_{D_y}\ m_{BC}\ \text{Tan}[\phi] - L\ a_C\ m_C\ \text{Tan}[\phi]}{L},$$
$$\left.\left.B_y \to P + a_{D_y}\ m_{BC} + a_C\ m_C,\ C_y \to P + a_C\ m_C\right\}\right\}$$

If I wanted to assign the one of the eight solutions to a variable, I would do it as before (get used to this, because you will be doing it a lot).

```
bX = Bx /. soln[[1]]
```

$$-\frac{a\ a_{D_x}\ m_{BC} - L\ a_{D_x}\ m_{BC} - \text{Sec}[\phi]\ I_{BC}\ \alpha_{BC} - L\ P\ \text{Tan}[\phi] - a\ a_{D_y}\ m_{BC}\ \text{Tan}[\phi] - L\ a_C\ m_C\ \text{Tan}[\phi]}{L}$$

Notice I didn't get the curly braces this time.

Solving a Single Second-Order Ordinary Differential Equation.

■ You begin by defining the equation to be solved.

For a single second-order ODE, you simply define the equation in the manner shown below. Notice, that you can include the initial conditions in the list (for numerical solutions, you *must* include the initial conditions). This equation happens to be a form of an equation called *Duffing's equation*.

```
duffing = {x''[t] + γ x'[t] - x[t] + x[t]^3 == A Cos[t], x[0] == 0.6, x'[0] == 1.25}
```

$$\{-x[t] + x[t]^3 + \gamma\ x'[t] + x''[t] == A\ \text{Cos}[t],\ x[0] == 0.6,\ x'[0] == 1.25\}$$

Notice that derivatives are indicated by "primes" and we have indicated that *x* is a function of time.

■ Now use the command `NDSolve` to get the solution.

□ I now get the solution. Notice that I must assign values to the constants using replacement rules.

The nice thing about using replacements rules is that the variables are *not* permanently assigned the numerical values.

```
duffsoln1 = NDSolve[duffing /. {γ -> 0.15, A -> 0.3}, x, {t, 0, 100}]
```

```
– NDSolve::mxst :
   Maximum number of 1000 steps reached at the point t == 58.9464420548711398`.
```

```
{{x → InterpolatingFunction[{{0., 58.9464}}, <>]}}
```

Mathematica has default limits on almost everything. We can integrate out past 58.9 seconds by setting the **`MaxSteps`** variable to a higher value.

```
duffsoln1 =
 NDSolve[duffing /. {γ -> 0.15, A -> 0.3}, x, {t, 0, 100}, MaxSteps -> 5000]
```

```
{{x → InterpolatingFunction[{{0., 100.}}, <>]}}
```

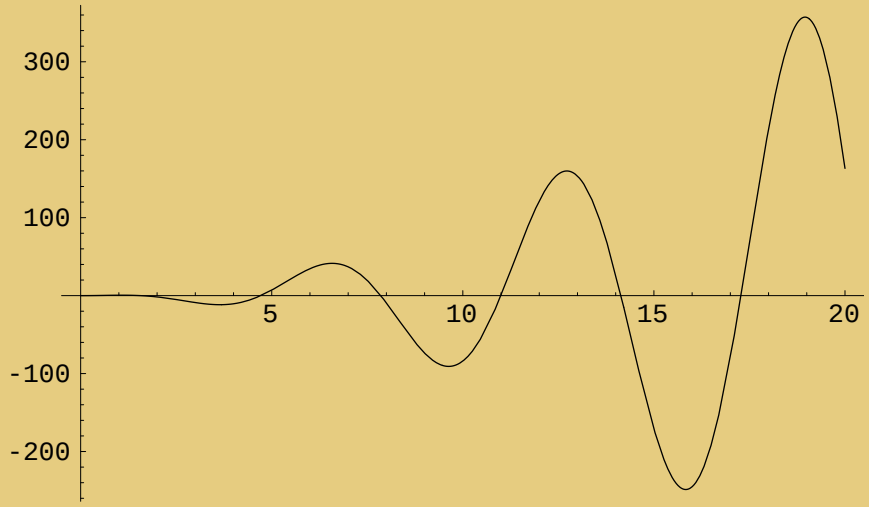
Mathematica returns an **`InterpolatingFunction`** as the solution. You can work with **`InterpolatingFunction`**'s as you would with **`Sin`** or **`Cos`** or any other function.

■ Plot the solution versus time.

We use the Plot command to plot the solution. First, let's look at a couple of simple examples.

□ 2D Plot Example

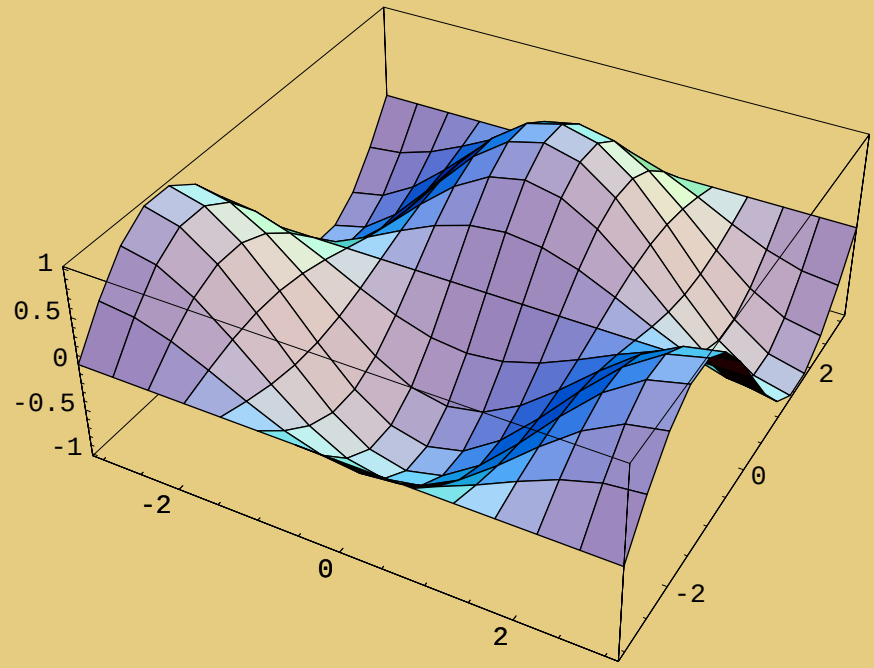
```
Plot[x^2 Cos[x], {x, 0, 20}]
```




```
- Graphics -
```

□ 3D Plot Example

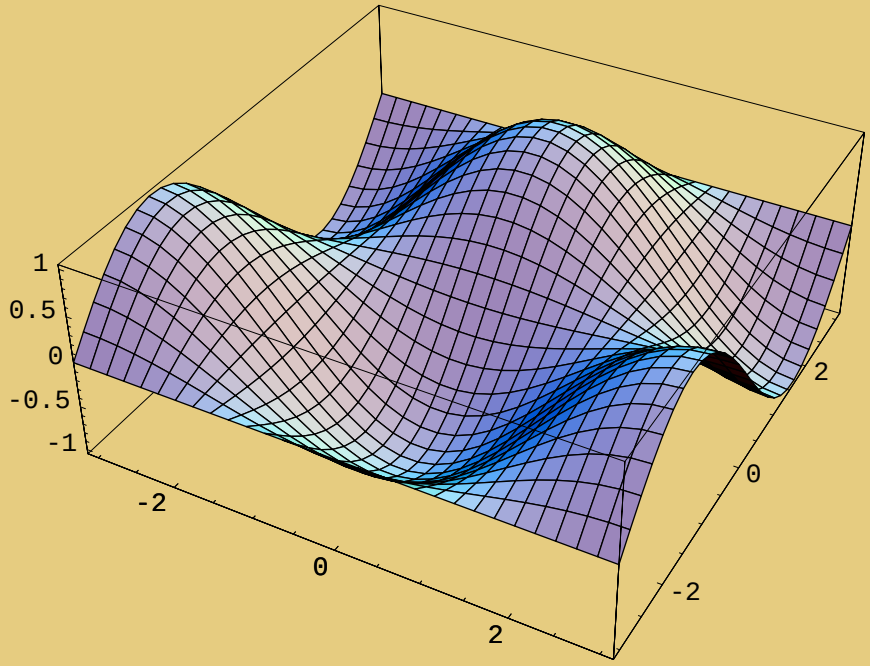
```
Plot3D[Cos[x] Sin[y], {x, -π, π}, {y, -π, π}]
```




```
- SurfaceGraphics -
```

We can increase the resolution.

```
Plot3D[Cos[x] Sin[y], {x, -π, π}, {y, -π, π}, PlotPoints -> 30]
```

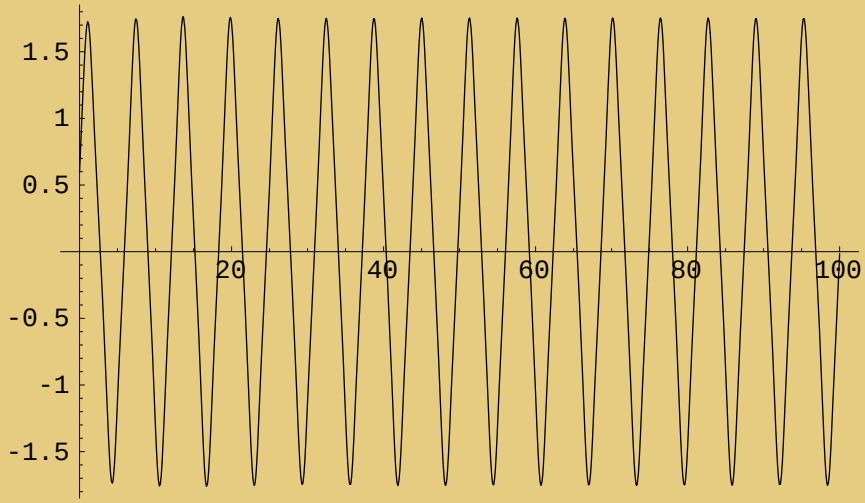



```
- SurfaceGraphics -
```

□ Plot the Solution to the Differential Equation

Remember that **duffsoln1** is an **InterpolatingFunction**. Note that you must wrap the replacement with **Evaluate** when working with **InterpolatingFunctions** in this way.

```
Plot[Evaluate[x[t] /. duffsoln1], {t, 0, 100}]
```

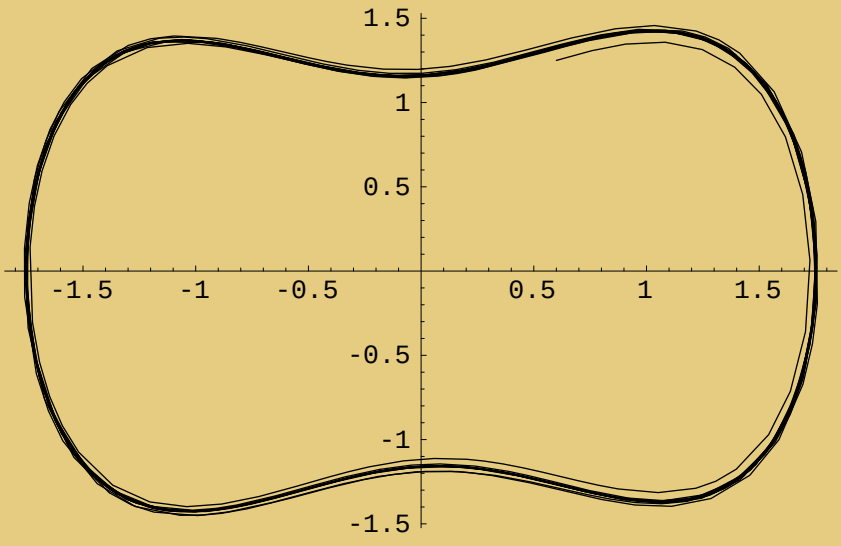



```
- Graphics -
```

□ Do a phase plot of the solution.

Now we are plotting $\dot{x}$ versus x ($\dot{x}$ is on the y axis and x is on the x axis). Notice that we can take the derivative of x by simply a "prime" on it.

```
ParametricPlot[Evaluate[{x[t], x'[t]} /. duffsoln1], {t, 0, 100},
 PlotRange -> All]
```



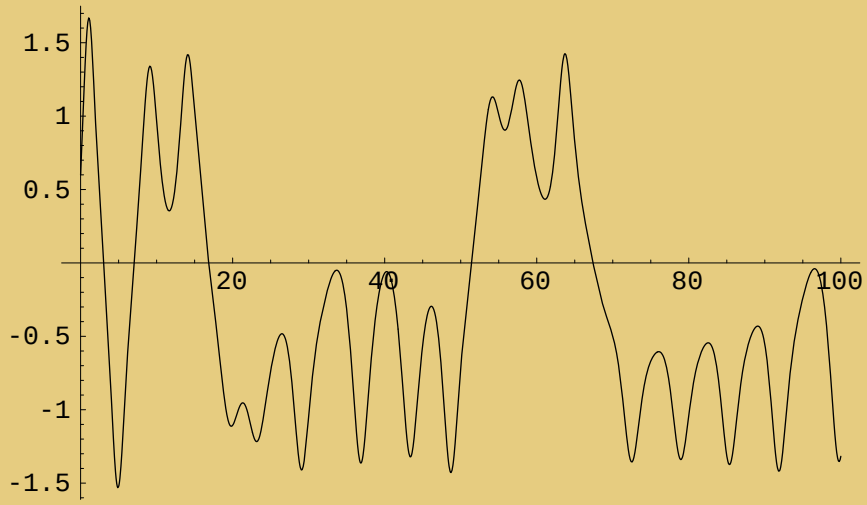

```
- Graphics -
```

That was a pretty boring solution. Here is a mathematically chaotic one! We integrate again with adifferent set of parameters. Now we see why the replacement rules are so useful.

```
duffsoln2 =
 NDSolve[duffing /. {γ -> 0.3, A -> 0.3}, x, {t, 0, 100}, MaxSteps -> 5000]
```

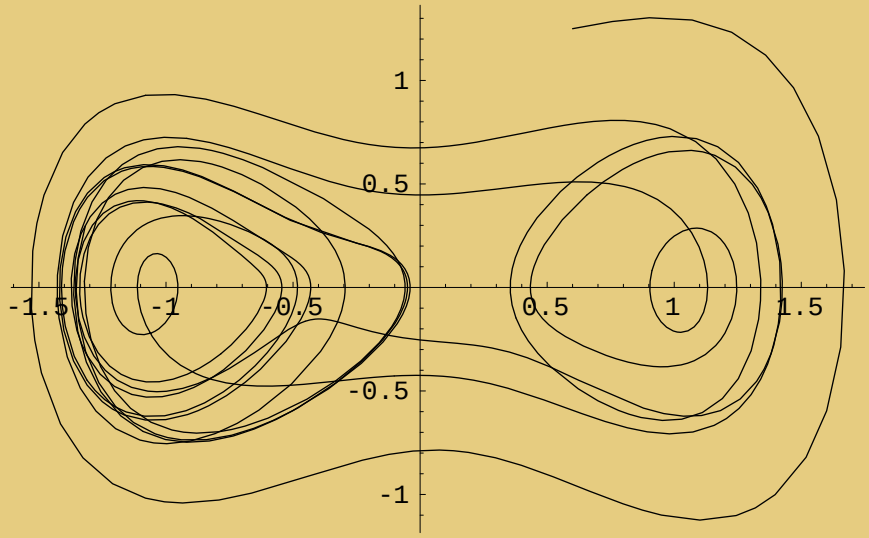
```
{{x → InterpolatingFunction[{{0., 100.}}, <>]}}
```

```
Plot[Evaluate[x[t] /. duffsoln2], {t, 0, 100}]
```




```
- Graphics -
```

```
ParametricPlot[Evaluate[{x[t], x'[t]} /. duffsoln2], {t, 0, 100},
 PlotRange -> All]
```




```
- Graphics -
```

Now on to a more complicated example.

A System of Second-Order Equations.

- **For a system of second-order ODEs, you can also put the equations and initial conditions in a list.**

Now, the list just gets longer. These happen to be the equations of motion for a damped, forced, elastic pendulum.

```
pendulum = {θ''[t] + 2 R'[t] θ'[t] / R[t] +
    c θ'[t] / m - g Cos[θ[t]] / R[t] == A Cos[ω t] / (m R[t]^2),
  R''[t] + c R'[t] / m - R[t] (θ'[t])^2 + k (R[t] - L0) / m - g Sin[θ[t]] == 0,
 θ[0] == 0, θ'[0] == 2.0, R[0] == 1.2, R'[0] == 0}
```

$$\left\{-\frac{g \text{Cos}[\theta[t]]}{R[t]}+\frac{c\,\theta'[t]}{m}+\frac{2\,R'[t]\,\theta'[t]}{R[t]}+\theta''[t]==\frac{A \text{Cos}[t\,\omega]}{m\,R[t]^2},\right.$$
$$\left.\frac{k\,(-L0+R[t])}{m}-g \text{Sin}[\theta[t]]+\frac{c\,R'[t]}{m}-R[t]\,\theta'[t]^2+R''[t]==0,\ \theta[0]==0,\right.$$
$$\left.\theta'[0]==2.,\ R[0]==1.2,\ R'[0]==0\right\}$$

□ Set up a vector of the parameters and their values.

```
params = {L0 -> 0.5, m -> 0.25, k -> 10, c -> 0.06, g -> 9.81, A -> 1, ω -> 3.6}
```

```
{L0 → 0.5, m → 0.25, k → 10, c → 0.06, g → 9.81, A → 1, ω → 3.6}
```

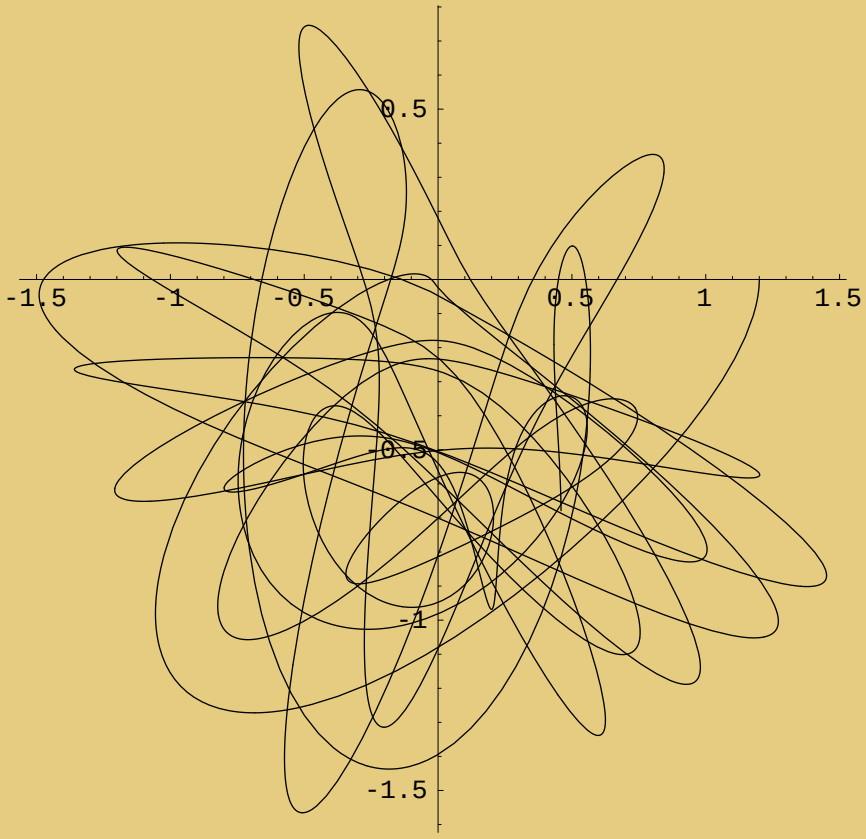
□ Numerically solve the equations of motion after substituting in the parameters.

```
pendsoln = NDSolve[pendulum /. params, {θ, R}, {t, 0, 20}, MaxSteps -> 20000]
```

```
{{θ → InterpolatingFunction[{{0., 20.}}, <>],
  R → InterpolatingFunction[{{0., 20.}}, <>]}}
```

□ Plot the trajectory.

```
ParametricPlot[Evaluate[{R[t] Cos[θ[t]], -R[t] Sin[θ[t]]} /. pendsoln],
 {t, 0, 20}, PlotPoints -> 1000, PlotRange -> All, AspectRatio -> 1]
```




```
- Graphics -
```

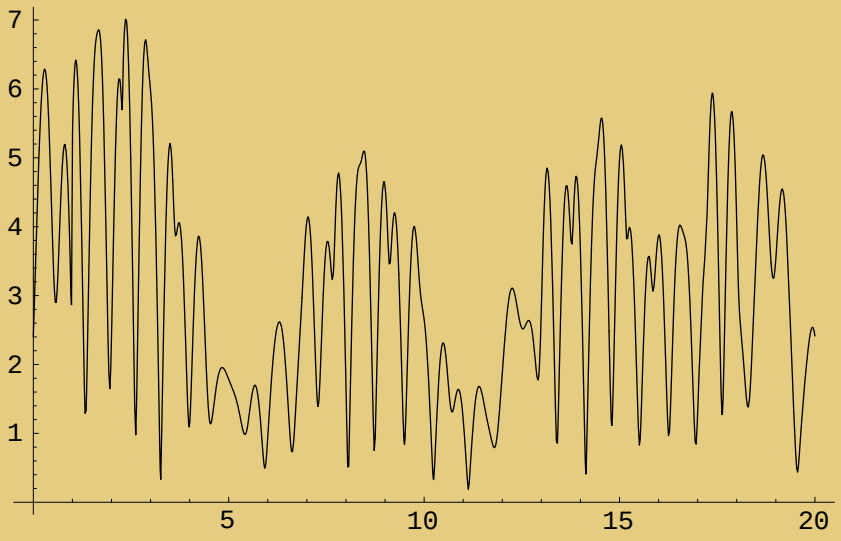
■ More on Interpolating Functions

We can combine them to form other quantities. For example, here is the velocity of the pendulum versus time.

```
velocity = Sqrt[(R'[t])^2 + (R[t] θ'[t])^2] /. pendsoln
```

```
{Sqrt[(InterpolatingFunction[{{0., 20.}}, <>][t]^2
     InterpolatingFunction[{{0., 20.}}, <>][t]^2 +
    InterpolatingFunction[{{0., 20.}}, <>][t]^2)]}
```

```
Plot[Evaluate[velocity], {t, 0, 20}]
```




```
- Graphics -
```

Mathematica can be quite wonderful!