Tuitorial 2: Differential equations in *Mathematica*: Analytic solutions

Brian Washburn, Version 1.0, 01/08/06

```
Off[General::spell];
```

Differential equations solved analytically in *Mathematica*

Typically one uses the function DSolve

```
?DSolve
```

```
DSolve[eqn, y, x] solves a differential equation for the function y, with independent variable
   x. DSolve[{eqn1, eqn2, ... }, {y1, y2, ... }, x] solves a list of differential
   equations. DSolve[eqn, y, {x1, x2, ... }] solves a partial differential equation. More…
```

Lets solve for a simple harmonic oscillator like a spring x''[t]=$-\omega^2$ x[t]. We want to find the position x as a function of time t, where ω is a constant.

```
DSolve[x''[t] == -ω^2 x[t], x, t]
```

```
{{x → Function[{t}, C[1] Cos[t ω] + C[2] Sin[t ω]]}}
```

This is a very general solution in terms of sine and cosine. However, we can make the solution more specific by imposing boundary conditions. Let us impose that velocity at t=0 is zero and the position at t=0 is a. Notice that you need two boundary conditions since one will get two constants when solving a 2nd order DE.

```
DSolve[{x''[t] == -ω^2 x[t], x[0] == a, x'[0] == 0}, x, t]
```

```
{{x → Function[{t}, a Cos[t ω]]}}
```

We cannot work with this function so well. Let's use some syntax to clean things up in order to define the solution to a function which we can plot, find its derivative, etc. Define the function xsol[t] which will be this analytic solution to the DE. The symbol := is a delayed assignment, we need to use it because we do not want to evaluate the function at first, but when you want to compute a value of xsol[t].

```
closeForm = DSolve[{x''[t] == -ω^2 x[t], x[0] == a, x'[0] == 0}, x, t]
```

```
{{x → Function[{t}, a Cos[t ω]]}}
```

```
xsol[t_] := (x /. closeForm[[1, 1]])[t]
```

```
xsol[t]
```

```
a Cos[t ω]
```

We get a cosine function with amplitude a and frequency ω. The function xsol[t] is an analtyic function t with the correct boundary conditions. Let us check the boundary conditions. We should get x(0)=a and dx/dt(0)=0

```
xsol[0]
```

```
a
```

Find dx/dt by taking the derivative analytically

```
dxsol[t_] = ∂t xsol[t]
```

```
-a ω Sin[t ω]
```

or by

```
D[xsol[t], t]
```

```
-a ω Sin[t ω]
```

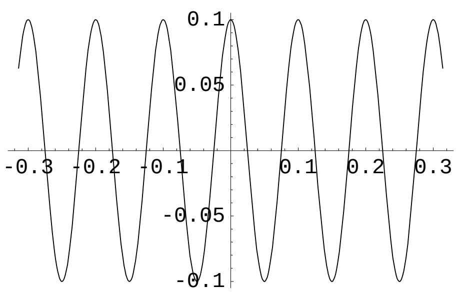
```
dxsol[0]
```

```
0
```

Our analytic solution xsol[t] has the correct boundary conditions. Let's plot the function. To do this we need some values so set a=0.1 m and $\omega=2*\pi*10$ Hz

```
a = 0.1;
ω = 2 π 10;
```

```
Plot[xsol[t], {t, -0.1 π, 0.1 π}];
```



In general, *Mathematica* kind of stinks for solving differential equations analytically. It might be best just to solve them by hand. However, the power of *Mathematica* comes in solving differential equations numerically, ones you cannot solve by hand!!!

■ Differential equations solved numerically in *Mathematica*

Let us solve the same differential equation numerically. To solve the DE numerically we cannot have any undefined constants. So define a and ω, and solve the DE x''[t]=$-\omega^2$ x[t] with the boundary conditions, x[0]=a and x'[0]=0. We will used the command NDSolve[].

```
a = 0.1;
ω = 2 π 10;
```

```
? NDSolve
```

```
NDSolve[eqns, y, {x, xmin, xmax}] finds a numerical solution to the ordinary
   differential equations eqns for the function y with the independent variable x in
   the range xmin to xmax. NDSolve[eqns, y, {x, xmin, xmax}, {t, tmin, tmax}] finds a
   numerical solution to the partial differential equations eqns. NDSolve[eqns, {y1,
   y2, ... }, {x, xmin, xmax}] finds numerical solutions for the functions yi. More…
```

```
solx = NDSolve[{x''[t] == -ω^2 x[t], x[0] == a, x'[0] == 0.0}, x, {t, -0.1 π, 0.1 π}]
```

```
{{x → InterpolatingFunction[{{-0.314159, 0.314159}}, <>]}}
```

Notice that we need a range of time values {t,-0.1 π,0.1 π}. This is because the output is not an analytic function! The output is really a list of x values for a given range of times (-0.1 π seconds to 0.1 π seconds). The Interpolation function is this "list". So, when you evaluate x[t] it looks up the value in the interpolation function for that value of t. Again, we have the function to something we can plot.

```
xsol[t_] := x[t] /. solx[[1]];
```

Let us check the numerical solution by testing the boundary conditions!

```
xsol[0]
```

```
0.1
```

Which is a! To find the velocity, we can take the derivative of xsol[t]. The result is a new interpolation function.

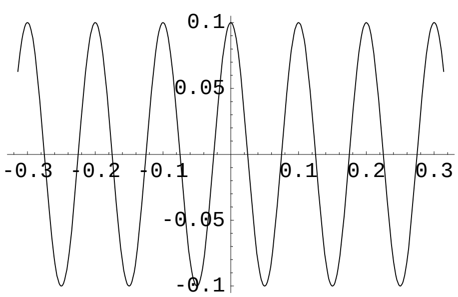
```
vxsol[t_] = ∂t xsol[t]
```

```
InterpolatingFunction[{{-0.314159, 0.314159}}, <>][t]
```

```
vxsol[0]
```

```
0.
```

```
Plot[xsol[t], {t, -0.1 π, 0.1 π}];
```



This plot is the same as the analytic result!

REMEMBER, WHEN DOING NUMERICAL SOLUTIONS ALWAYS FIND SOME WAY TO CHECK YOUR RESULTS