Fitting Parameters in a Differential Equation With a Non-Analytical Solution

Mathew Smith 29-Jan-2005

```
In[64]:= Remove["Global`*"]
         Off[General::"spell"]
         Off[General::"spell1"]
         << Graphics`
```

Often differential equations do not have analytical solutions, however such equations can be solved using numerical techniques. As was shown in the "Differential Equations in *Mathematica*" notes the NDSolve command in *Mathematica* can be used to solve differential equations numerically. For example consider the equation:

if: $t < 3$ $\frac{df}{dt} = 10 - f$,
if: $t \geq 3$ $\frac{df}{dt} = -f$
where $f = 5$ at $t = 0$.

This can be solved in *Mathematica* using the following command:

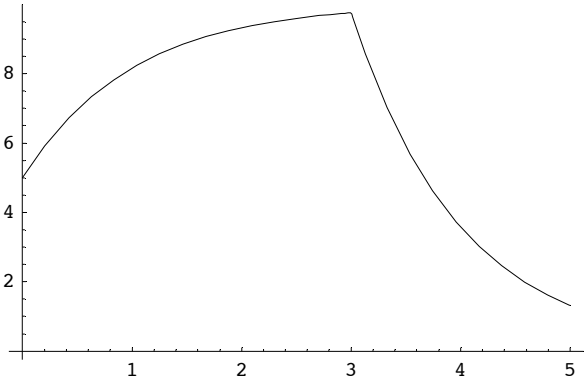
```
In[37]:= sol = NDSolve[{f'[t] == If[t < 3, 10 - f[t], -f[t]], f[0] == 5}, f, {t, 0, 5}][[1]]
Out[37]= {f → InterpolatingFunction[{{0., 5.}}, <>]}
```

The result is an interpolating function which has been fitted to the numerical data. The function is only valid in the specified range of the independant variable i.e for $0 \leq t \leq 5$. We can define a function fn(t) that is equal to the polynomial using the following command:

```
In[38]:= fn[t_] = f[t] /. sol
Out[38]= InterpolatingFunction[{{0., 5.}}, <>][t]
```

And then plot the result using:

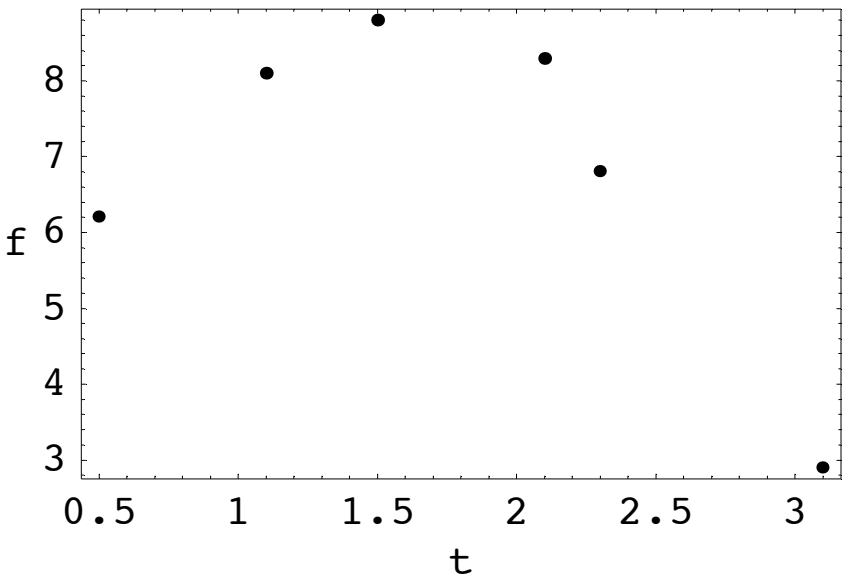
```
In[39]:= p1 = Plot[fn[t], {t, 0, 5}];
```



Imagine we have conducted an experiment where we have measured f as a function of t with the following results:

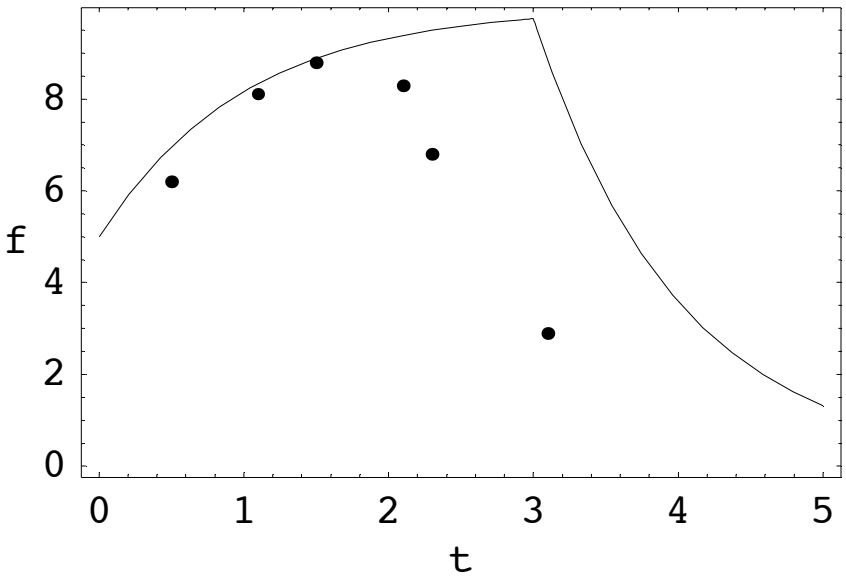
```
In[68]:= texp = {0.5, 1.1, 1.5, 2.1, 2.3, 3.1};
         fexp = {6.2, 8.1, 8.8, 8.3, 6.8, 2.9};
```

```
In[81]:= p2 = ListPlot[Transpose[{texp, fexp}],
            PlotStyle → AbsolutePointSize[6], Frame → True, Axes → False,
            FrameLabel → {"t", "f"}, TextStyle → {FontSize → 20, FormatType → Bold},
            RotateLabel → False, ImageSize → {400, 300}];
```



We can compare our numerical solution for f as follows :

```
In[43]:= Show[p2, p1];
```



The solution clearly isn't such a great fit. The goodness of the fit can be characterised by calculating the sum of the square differences between the solution and the data :

$R = \sum (\text{fexp} - f[t])^2$

To do this first we evaluate our numerical function at the experimental values of t:

```
In[44]:= fn1 = fn[t] /. t → texp
Out[44]= {6.96735, 8.33564, 8.88435, 9.38772, 9.49871, 8.82313}
```

Then we calculate find the difference between the experimental values and values from the model and square them:

```
In[45]:= df = (fexp - fn1)^2
Out[45]= {0.588821, 0.0555285, 0.007115, 1.18313, 7.28301, 35.0834}
```

Finaly we sum the numbers using:

```
In[46]:= R = Apply[Plus, df]
Out[46]= 44.2011
```

These comands can be neatly bundled together to give:

```
In[47]:= R = Apply[Plus, (fexp - (f[t] /. sol /. t → texp))^2]
Out[47]= 44.2011
```

Imagine that we decide to modify our initial differential equation such that we have:

if: $t < p$ $\frac{df}{dt} = 10 - f$,
if: $t \geq p$ $\frac{df}{dt} = -f$

Where p has to be deduced experimentally. Also we realise that in our haste to do the experiment we forgot to record the initial value for f at t = 0, lets call this q, so we don't know that constant either. How would we go about finding values for these? One way to do this would be to guess at initial values for p and q, then use NDSolve to find the solution for f(t) and then calculate the sum of the square differences. One could then proceed to calculate the least squares difference iteratively over a large range of values of p and q until the minimum in the sum of the square difference is found. Fortunatley for us we can write a short piece of code in *Mathematica* that will do this for us. We define the function sse(p,q) as follows:

```
In[71]:= sse[p_?NumberQ, q_?NumberQ] := Block[{sol, f}, sol =
           NDSolve[{f'[t] == If[t < p, 10 - f[t], -f[t]], f[0] == q}, f, {t, 0, 5}][[1]];
         Apply[Plus, (fexp - (f[t] /. sol /. t → texp))^2]]
```

The exact syntax is perhaps a little confusing but what the function is doing is actually quite simple. The _?NumberQ following the letters p and q signifies that the function needs numerical inputs for both p and q. The fuction is then defined as a block, this allows us to combine *Mathematica* commands which will all be evaluated together. The declaration {sol,f} tells *Mathematica* that the block contains two seperate functions; sol and f. The function then evaluates the NDSolve command to find f(t) with the supplied values of q and p before calculating the least squares difference between f(t) and the experimental values of f. The semi-colon after the NDSolve comand supresses its output so that the function just returns the least squares difference. For example if we evalute the function for p =3 and q=5 we get:

```
In[72]:= sse[3, 5]

Out[72]= 44.2011
```

Which is the same as we calculated above.

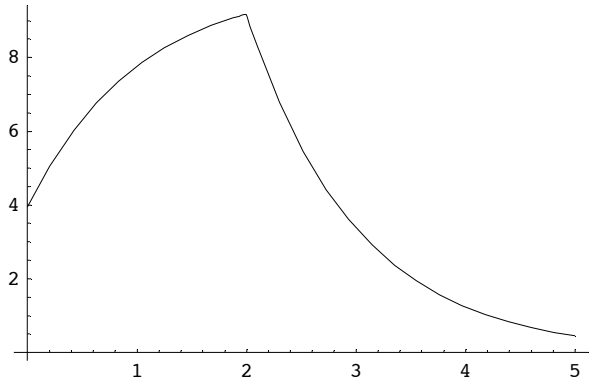
The FindMinimum in *Mathematica* command uses numerical methods to find the minimum of a function. We can use it to find the minimum value of sse(p,q) in the range $2 \leq p \leq 3$ and $4 \leq q \leq 5$ as follows:

```
In[73]:= bf = FindMinimum[sse[p, q], {p, 2, 3}, {q, 4, 5}]

Out[73]= {0.0743735, {p → 1.9962, q → 3.93581}}
```

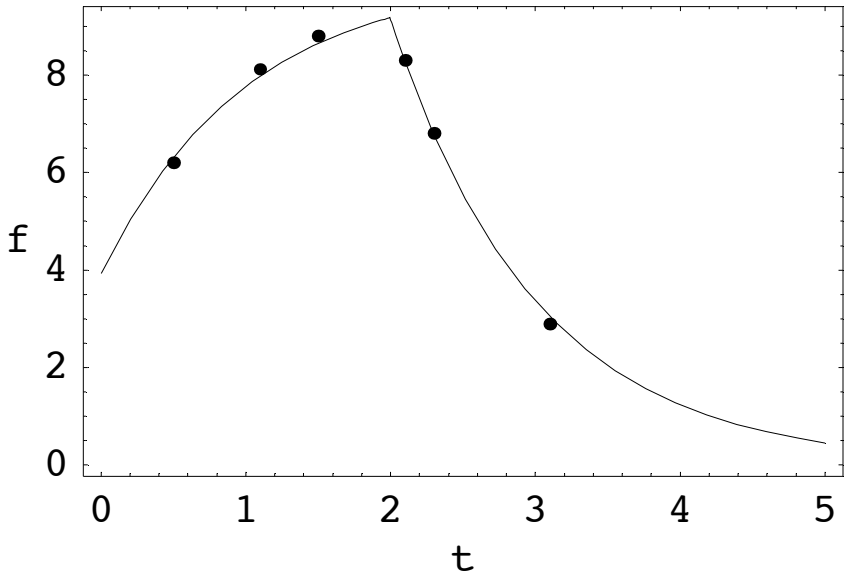
The results tell us that the minimum occurs at p = 1.9962 and q = 3.93581. We can now find f(t) at this value and plot it against our data:

```
In[74]:= pf = bf[[2, 1, 2]]

Out[74]= 1.9962

In[75]:= qf = bf[[2, 2, 2]]

Out[75]= 3.93581
```

```
In[77]:= solbf =
          NDSolve[{f'[t] == If[t < pf, 10 - f[t], -f[t]], f[0] == qf}, f, {t, 0, 5}][[1]]

Out[77]= {f → InterpolatingFunction[{{0., 5.}}, <>]}

In[79]:= p3 = Plot[f[x] /. solbf, {x, 0, 5}];
```




```
In[83]:= Show[p2, p3];
```



The result seems to fit the data very well.