Boundary value problems with Mathematica

■ Example, easy, to show the method

Mathematica's NDSolve is not able to solve the following (generalized) boundary problem automatically.

```
In[1]:= NDSolve[{y'''[t] - t y'[t] + 2 y[t] == -10 + t, y[-1] == 3, y'[2] == 4, y[4] == 3}, y, {t, -1, 4}]
```

We define

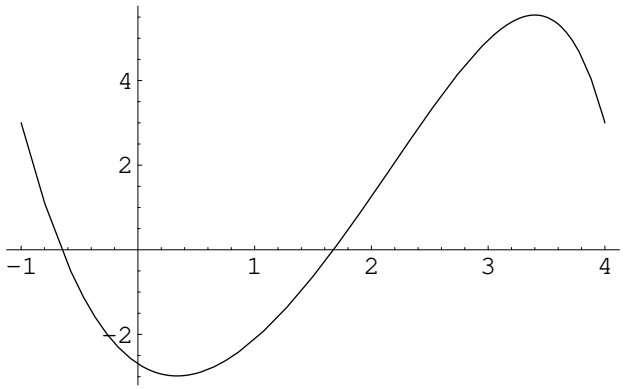
```
In[2]:= f1[a_, b_] := NDSolve[{y'''[t] - t y'[t] + 2 y[t] == -10 + t,
           y[-1] == 3, y'[-1] == a, y''[-1] == b}, y, {t, -1, 4}]

        f2[a_, b_] := With[{f = First[y /. f1[a, b]]}, {f'[2], f[4]}]
```

and solve with FindRoot.

```
In[4]:= sol = FindRoot[f2[a, b] == {4, 3}, {a, {0, 1}}, {b, {0, 1}}]
        Plot[Evaluate[y[t] /. f1[a /. sol, b /. sol]], {t, -1, 4}]
```

```
Out[4]= {a → -10.5693, b → 11.8265}
```




```
Out[5]= - Graphics -
```

■ Nonlinear Example

Consider the problem

$$y''(t) = 5 \sinh(5\, y(t))$$
$$y(0) = 0$$
$$y(1) = 1$$

Stoer and Bulirsch use this example in their book to illustrate that simple shooting is *not* sufficient to solve nasty problems and thus motivate the method of multiple shooting. The code below, however, demonstrates that Mathematica does it with simple shooting and ease.

We proceed as before,

```
In[6]:= f3[a_] := NDSolve[{y''[t] == 5 Sinh[5 y[t]], y[0] == 0, y'[0] == a}, y, {t, 0, 1}]
```

but it turns out that the solution sometimes doesn't reach the right boundary.

```
In[7]:= f3[1]

        NDSolve::ndsz : At t == 0.43129882432337696`, step size is effectively zero; singularity suspected.

Out[7]= {{y → InterpolatingFunction[{{0., 0.431299}}, <>]}}
```

We turn off the message

```
In[8]:= Off[NDSolve::ndsz]
```

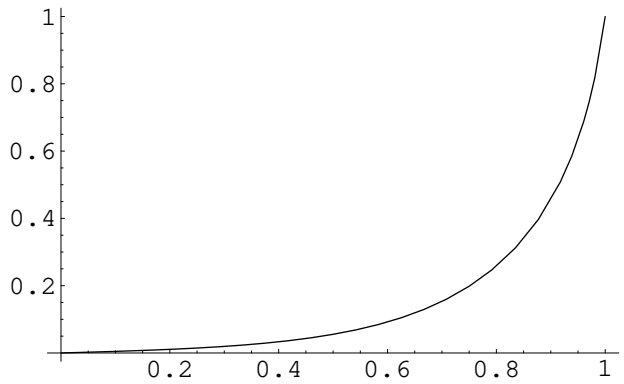
and cheat a bit in the definition of the second function: When it doesn't reach the right boundary, we use instead the last value calculated.

```
In[9]:= f4[a_] := With[{f = f3[a][[1, 1, 2]]}, f[f[[1, 1, 2]]]]
```

Now we proceed again as before.

```
In[10]:= sol = FindRoot[f4[a] == 1, {a, {0, 1}}]
         Plot[Evaluate[y[t] /. f3[a /. sol]], {t, 0, 1}]

Out[10]= {a → 0.0457418}
```




```
Out[11]= - Graphics -
```

A large class of boundary value problems can be solve like that.

The above problem actually has an analytic solution. Previously, *Mathematica* was not able to find an exact expression, but since Version 4.1 we get at least the general form of the solution.

```
In[12]:= sol = DSolve[{y''[t] == 5 * Sinh[5 * y[t]]}, y, t]

Solve::ifun : Inverse functions are being used by Solve, so some solutions may not be found.

Out[12]= {{y → Function[{t},
      -2/5 i JacobiAmplitude[-5/2 Sqrt[-2 t^2 - t^2 C[1] - 4 t C[2] - 2 t C[1] C[2] - 2 C[2]^2 - C[1] C[2]^2], 4/(2 + C[1])]]},
   {y → Function[{t}, -2/5 i JacobiAmplitude[
      5/2 Sqrt[-2 t^2 - t^2 C[1] - 4 t C[2] - 2 t C[1] C[2] - 2 C[2]^2 - C[1] C[2]^2], 4/(2 + C[1])]]}}
```

We can now try to solve for the two constants of integration. One of the constants appears only in one of the arguments of `JacobiAmplitude`, so it is readily solved for.

```
In[13]:= const1 = Solve[0 == y[0] /. sol[[2]], C[2]]

Solve::ifun : Inverse functions are being used by Solve, so some solutions may not be found.

Out[13]= {{C[2] → 0}, {C[2] → 0}}
```

The other constant, however, appears in both arguments, *Mathematica* cannot handle this yet, and we have to dig out some book on elliptic functions.

```
In[14]:= constw = Solve[1 == y[1] /. sol[[2]] /. const1[[1]], C[1]]

Solve::dinv : The expression JacobiAmplitude[5/2 Sqrt[-2 - C[1]], 4/(2 + C[1])]
   involves unknowns in more than one argument, so inverse functions cannot be used.

Out[14]= Solve[1 == -2/5 i JacobiAmplitude[5/2 Sqrt[-2 - C[1]], 4/(2 + C[1])], C[1]]
```

Note that we could try to do shooting with the exact form of the solution. However, singularities in this case are not so easily detected, as the evaluation of the exact expression at the right boundary would not give us any clue.

■ Surface height in a capillary

The following differential equation describes the surface height of a fluid in a cylindrical capillary. It is a boundary value problem with a parameter, the curvature κ of the capillary surface at center point. The given surface angle θ at the wall of the capillary determines the stiffness of the problem.

$$y''(x) = \left(y'(x)^2 + 1\right)^{3/2} (2\kappa + B\, y(x)) - \frac{(y'(x)^2 + 1)\, y'(x)}{x}$$
$$y(0) = 0$$
$$y'(0) = 0$$
$$y'(1) = \cot(\theta)$$

We set up the same procedure as in the previous example to solve the problem by simple shooting. Note that the right side of the differential equation is singular at 0, so that we have to have to substitute explicitly the limit as x→0 of the right side for small values of x. Again, we have to trap the solution of the initial value problem in case they become singular before reaching the right end point.

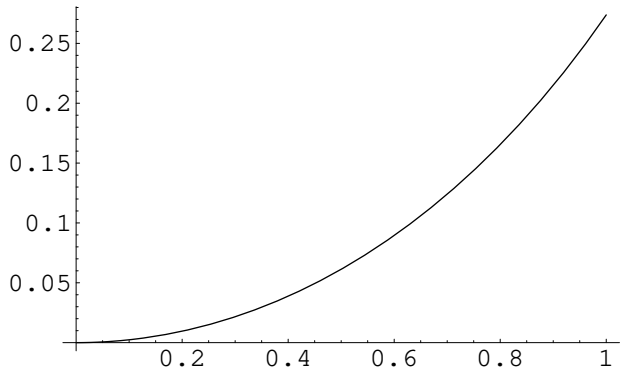
When θ=1, we readily obtain a solution.

```
In[15]:= f5[κ_] := NDSolve[
           {y''[x] == If[x < 10^-13, κ, (1 + y'[x]^2)^(3/2) (y[x] + 2κ) - (1 + y'[x]^2) y'[x] / x],
            y[0] == 0, y'[0] == 0}, y, {x, 0, 1}]

         Off[NDSolve::ndsz]
         f6[κ_] := With [{f = f5[κ][[1, 1, 2]]}, f'[f[[1, 1, 2]]]]

         θ = 1;
         sol = FindRoot [f6[κ] == Cot[θ], {κ, {0, 1}}, WorkingPrecision → 32, AccuracyGoal → 6]
         Plot [Evaluate [y[t] /. f5 [κ /. sol]], {t, 0, 1}]
```

```
Out[19]= {κ → 0.47526}
```




```
Out[20]= - Graphics -
```

When θ is decreased, the problem becomes stiffer, and we have to increase the maximal number of iterations of the root finder to converge within the predetermined accuracy.

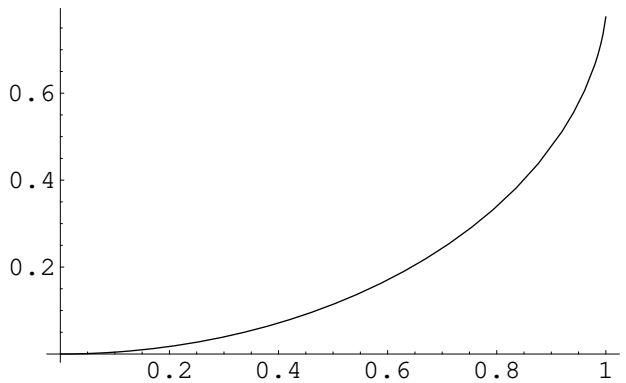
```
In[21]:= θ = 0.1;
         sol = FindRoot [f6[κ] == Cot[θ], {κ, {0, 1}}]

         FindRoot::frsec : Secant method failed to converge to the prescribed accuracy after 15 iterations.

Out[22]= {κ → 0.854386}

In[23]:= sol = FindRoot [f6[κ] == Cot[θ], {κ, {0, 1}}, MaxIterations → 20]
         Plot [Evaluate [y[t] /. f5 [κ /. sol]], {t, 0, 1}]

Out[23]= {κ → 0.854386}
```




```
Out[24]= - Graphics -
```

This value of θ is even more drastic.

```
In[25]:= θ = 0.01;
         sol = FindRoot [f6[κ] == Cot[θ], {κ, {0, 1}}, MaxIterations → 100]

        FindRoot::frmp : Machine precision is insufficient to achieve the accuracy 1.0000000000000006`*^-6.

Out[26]= {κ → 0.857986}
```

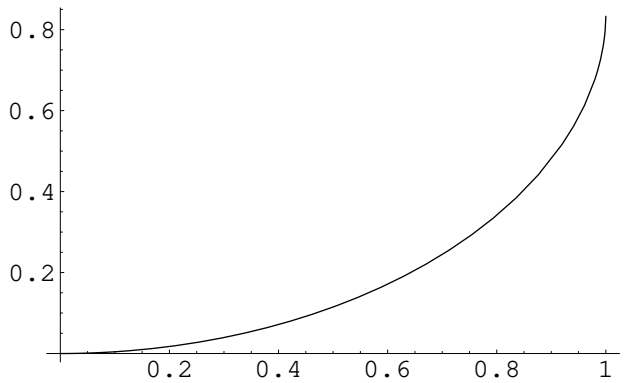
Mathematica notices that the result is not accurate to 6 digits. Increasing the precision target for the ODE solver helps. This is a very tricky computation, so narrowing the initial search values for κ helps avoiding some warning messages (however the root solver will converge even for an initial search range $\{\kappa,\{0,1\}\}$).

```
In[27]:= f5[κ_] := NDSolve[
           {y''[x] == If[x < 10^-13, κ, (1 + y'[x]^2)^(3 / 2) (y[x] + 2 κ) - (1 + y'[x]^2) y'[x] / x],
            y[0] == 0, y'[0] == 0}, y, {x, 0, 1},
           MaxSteps → 10000, PrecisionGoal → Infinity, AccuracyGoal → 8]

         Off[NDSolve::ndsz]
         f6[κ_] := With [{f = f5[κ][[1, 1, 2]]}, f'[f[[1, 1, 2]]]]

         θ = 1 / 100;
         sol = FindRoot [f6[κ] == Cot[θ], {κ, {0.857, 0.858}}, MaxIterations → 100]
         Plot [Evaluate [y[t] /. f5 [κ /. sol]], {t, 0, 1}]

Out[31]= {κ → 0.858001}
```




```
Out[32]= - Graphics -
```

■ Acknowledgment

I found the idea of how to implement simple shooting in *Mathematica* in a MathGroup posting by Jurgen Tischer. The first two examples are due to him, I only did some editing when converting his post into a *Mathematica* notebook. The original post is archived at

http://library.wolfram.com/mathgroup/archive/1998/Dec/msg00048.html