Classroom Tips and Techniques: Numeric Solution of a Two-Point BVP

Robert J. Lopez
Emeritus Professor of Mathematics and Maple Fellow
Maplesoft

Introduction

A recent query from a colleague in engineering posed a nonlinear two-point boundary value problem that could only be solved numerically. In this month's article, we'll explore the solution of a similar problem, highlighting the way Maple can be used to obtain the required solution.

Initializations

```
> restart
> with(plots):
  with(Optimization):
  with(LinearAlgebra):
> interface(typesetting=extended):
  Typesetting[Suppress](y(x)):
```

Statement of the BVP

Consider the nonlinear ODE

```
> q := y''+y^2*y'+cos(y) = x
```

$$q := y'' + y^2 y' + \cos(y) = x$$

and the related nonlinear boundary conditions

```
> b1 := (1+y(0))*y'(0)-3*y(0) = 0;
  b2 := (1+y(1))*y'(1)-3*y(1)-y(1)^2+1/2 = 0
```

$$b1 := (1 + y(0))\, y'(0) - 3\, y(0) = 0$$

$$b2 := (1 + y(1))\, y'(1) - 3\, y(1) - y(1)^2 + \frac{1}{2} = 0$$

that pose a two-point boundary value problem on the interval $0 \le x \le 1$.

Using *dsolve* to Obtain a Solution

A naive use of Maple's **dsolve** command did not provide a solution to the problem originally posed, running at great length and then crashing. However, inclusion of the *approxsoln* parameter very quickly led to the desired solution. This parameter provides an initial guess for an iterative Rayleigh-Ritz solution process that converges very quickly to a solution. In the original engineering problem there were two possible solutions, only one of which was physically meaningful.

If we apply that same strategy to the problem posed in the previous section, assuming that perhaps a solution might be close to $y(x) = x$, we obtain the graph shown in Figure 1.

```
> Q := dsolve( {q, b1, b2}, y(x), numeric,
      approxsoln = [y(x) = x]) :
  odeplot(Q)
```

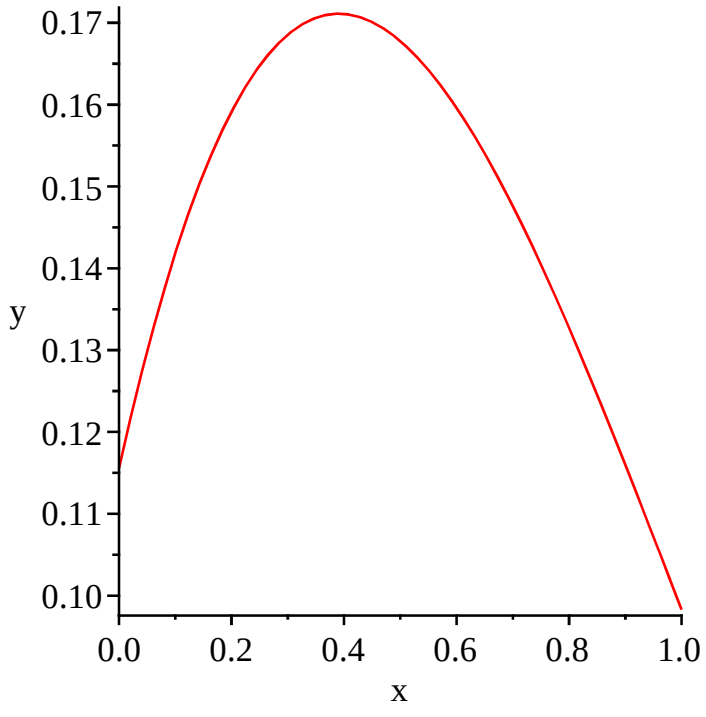


Figure 1 Solution of nonlinear two-point BVP starting with initial guess $y(x) = x$

In principle, it should be possible to solve this BVP via a shooting method, using the two boundary conditions as the governing equations that determine $y(0) = a$ and $y'(0) = b$. In the sequel, we apply this idea to the solution process, and also come to grips with another disturbing facet of the problem we selected, namely, finding all possible solutions.

A Solution from First Principles

We begin by defining a procedure $h(a, b)$ that, given values for $y(0) = a$, $y'(0) = b$, returns the list $[y(1), y'(1)]$. This procedure implements a shooting method, starting from the initial values of a and b, numerically integrating a solution for the resulting initial value problem, and returning the endpoint values.

```
> h := proc(a,b)
  local F;
  if type(a, numeric) or type(b, numeric) then
  F := dsolve({q, y(0)=a,D(y)(0)=b}, y(x), numeric);
  [op([2,2],F(1)),op([3,2],F(1))];
  else 'h'(a,b);
  end if;
  end proc:
```

In terms of a, b, and $h(a, b)$, the boundary conditions can be expressed via the expressions

$$> f := (1 + a)\, b - 3\, a$$
$$g := \left(1 + h(a, b)_1\right) h(a, b)_2 - 3\, h(a, b)_1 + 0.5 - h(a, b)_1^2$$

$$f := (1 + a)\, b - 3\, a$$

$$g := \left(1 + h(a, b)_1\right) h(a, b)_2 - 3\, h(a, b)_1 + 0.5 - h(a, b)_1^2$$

It is now possible to graph the functions f and g, resulting in Figure 2.

$$> implicitplot([f = 0, g = 0], a = -5..1, b = -11..6.5, color = [black, red])$$

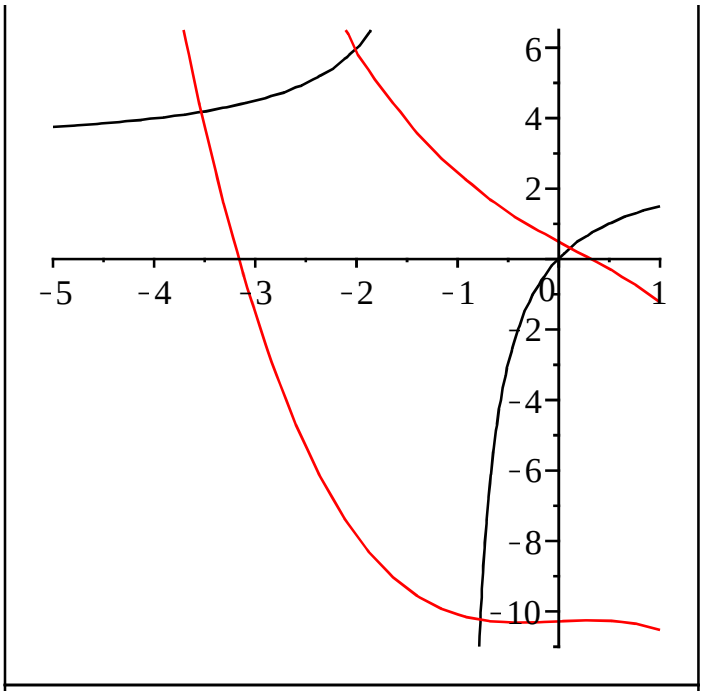


Figure 2 Graph of the equations for $y(0) = a$ and $y'(0) = b$, as determined by the boundary conditions of the BVP

Figure 2 suggests that there are actually four solutions of the given BVP.

Unfortunately, the fsolve command in Maple 11 does not permit systems of procedure calls, as would be required by our definitions of f and g through the procedure $h(a, b)$. Fortunately, this shortcoming is corrected in the next version of Maple. Meanwhile, we need to solve a pair of nonlinear algebraic equations whose terms are calculated via the solution of an initial value problem.

We propose three methods for this. First, we implement Broyden's method as a generalization of the Secant algorithm for solving a single equation. Second, we apply a minimization technique to $f^2 + g^2$, a sum of squares. Finally, because of the special nature of the first boundary condition, we can eliminate b from the second equation, and apply the Secant method to the single resulting equation.

In anticipation of all three iterative techniques, we define the following initial points, each taken from Figure 2.

```
> P||(1..4) := ⟨-3.6, 4.2⟩, ⟨-2, 6⟩, ⟨-0.8, -10⟩, ⟨0.1, 0.4⟩
```

$$P1, P2, P3, P4 := \begin{bmatrix} -3.6 \\ 4.2 \end{bmatrix}, \begin{bmatrix} -2 \\ 6 \end{bmatrix}, \begin{bmatrix} -0.8 \\ -10 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.4 \end{bmatrix}$$

Broyden's Method

Newton's method for solving the system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ is captured by the fixed-point scheme

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J^{-1}(\mathbf{x}_k)\mathbf{F}(\mathbf{x}_k)$$

where $J(\mathbf{x})$ is the Jacobian matrix for $\mathbf{F}$. Since this requires computation of derivatives, it would be tedious to implement a method for repeatedly computing J. As an alternative, we consider Broyden's method wherein J is only computed once. Thereafter, an approximation to J is computed, an approximation that converges to $J(\widehat{\mathbf{x}})$, where $\widehat{\mathbf{x}}$ is a solution to the system $\mathbf{F}(\mathbf{x}) = \mathbf{0}$.

After obtaining $J(\mathbf{x}_0) = J_0$ either analytically or approximately by numeric differentiation, Broyden's method thereafter replaces J_k, $k \geq 1$, with

$$A_k = A_{k-1} + \frac{\mathbf{F}(\mathbf{x}_k) - \mathbf{F}(\mathbf{x}_{k-1}) - A_{k-1}[\mathbf{x}_k - \mathbf{x}_{k-1}]}{\|\mathbf{x}_k - \mathbf{x}_{k-1}\|^2}[\mathbf{x}_k - \mathbf{x}_{k-1}]^{\mathrm{T}}$$

as an approximation to $J(\mathbf{x}_k)$.

Our implementation of Broyden's method takes the vector $\langle f, g \rangle$ as impute, along with an initial point, a stepsize d used for numerically computing J_0, and the integer N used to define the computational tolerance 10^{-N}. Our code is simplistic, in that it does not type-checking, and does not make provision for generalization to other systems. It is provided as a sketch to show how such a computation might be implemented.

```
> Broyden := proc(F,X0,d,N)
  local X, F0, F1, f, g, fa, ga, fb, gb, A0, A1, x0, x1, tol, k,
  L, Z;
  x0 := X0;
  X := <a,b>:
  f := F[1];
  g := F[2];
  Z := Equate(X,x0);
  F0 := eval(F,Equate(X,x0)):
  fa := (eval(f,Equate(X,x0+<d,0>)) - F0[1])/d:
  ga := (eval(g,Equate(X,x0+<d,0>)) - F0[2])/d:
  fb := (eval(f,Equate(X,x0+<0,d>)) - F0[1])/d:
  gb := (eval(g,Equate(X,x0+<0,d>)) - F0[2])/d:
  A0 := Matrix([[fa,fb],[ga,gb]]):
  x1 := x0 - A0^(-1).F0:
  Digits := 30:
  tol := 10^(-N):
  for k from 1 to 10 do
  F1 := eval(F, Equate(X,x1));
  L := x1 - x0;
  A1 := A0 + (F1 - F0 - A0.L).L^%T/(L.L);
  Z := A1^(-1).F1;
  if sqrt(Z.Z)<tol then break;
```

```
else x0 := x1; x1 := x1-Z; A0 := A1; F0 := F1;
end if;
end do:
Digits := 10:
x1;
end proc:
```

Using our Broyden procedure, we compute solutions for $y(0) = a$ and $y'(0) = b$.

> **for** k **to** 4 **do**
$S||k := Broyden(\langle f, g\rangle, P||k, 0.0001, 20)$
end do

$$S1 := \begin{bmatrix} -3.535939018506848260191122195536 \\ 4.182993746342681840407423506 40 \end{bmatrix}$$

$$S2 := \begin{bmatrix} -2.002236227145694393748397818 28 \\ 5.993306287225129465590019156 86 \end{bmatrix}$$

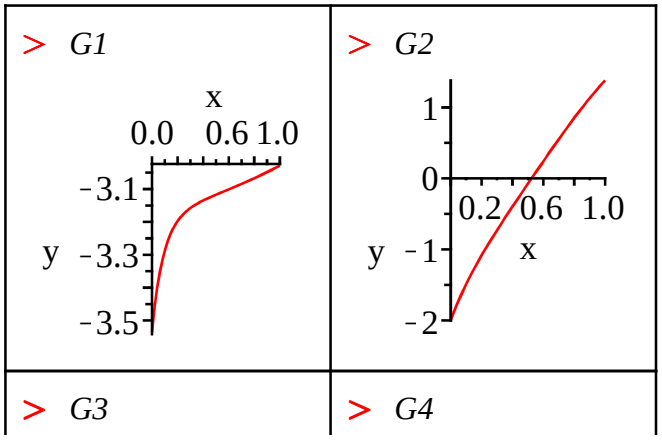
$$S3 := \begin{bmatrix} -0.773424076146977449613242004934 \\ -10.240594803647667594207749862 1 \end{bmatrix}$$

$$S4 := \begin{bmatrix} 0.115558402402174006320135310253 \\ 0.310763834918918823625744340795 \end{bmatrix}$$

With pairs (a, b) determined, we can solve the corresponding four initial value problems, and graph their solutions. The following commands generate the solutions and graphs,

> **for** k **to** 4 **do**
$temp := dsolve(\{q, y(0) = S||k[1], D(y)(0) = S||k[2]\}, y(x), numeric);$
$G||k := odeplot(temp, [x, y(x)], x = 0..1);$
end do:

and the graphs themselves appear in Figure 3.



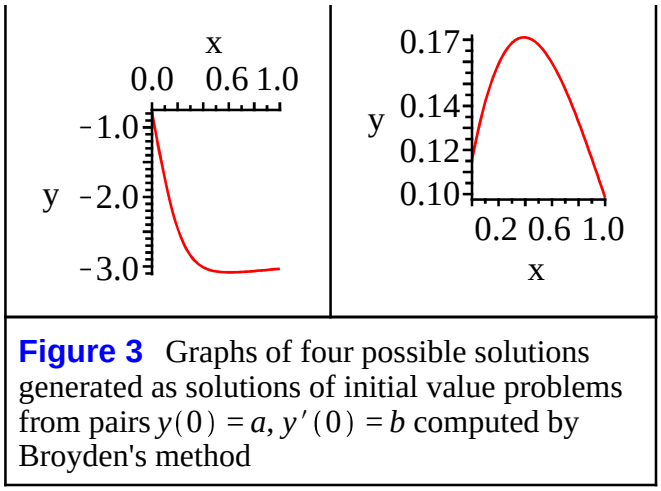


Figure 3 Graphs of four possible solutions generated as solutions of initial value problems from pairs $y(0) = a, y'(0) = b$ computed by Broyden's method

The four solutions found in Figure 3 can be obtained directly with Maple's dsolve command, provided an appropriate approximate solution is provided. If we list the four approximate solutions

```
> L := [x - 4, 3 x - 2, -10 x, x (1 - x)]
```

$$L := [x - 4, 3x - 2, -10x, x(1 - x)]$$

then the following loop computes the corresponding **dsolve** solutions.

```
> for k to 4 do
  Q||k := dsolve( {q, b1, b2}, y(x), numeric, approxsoln = [y(x) = L_k] )
  end do:
```

Figure 4 displays these solutions.

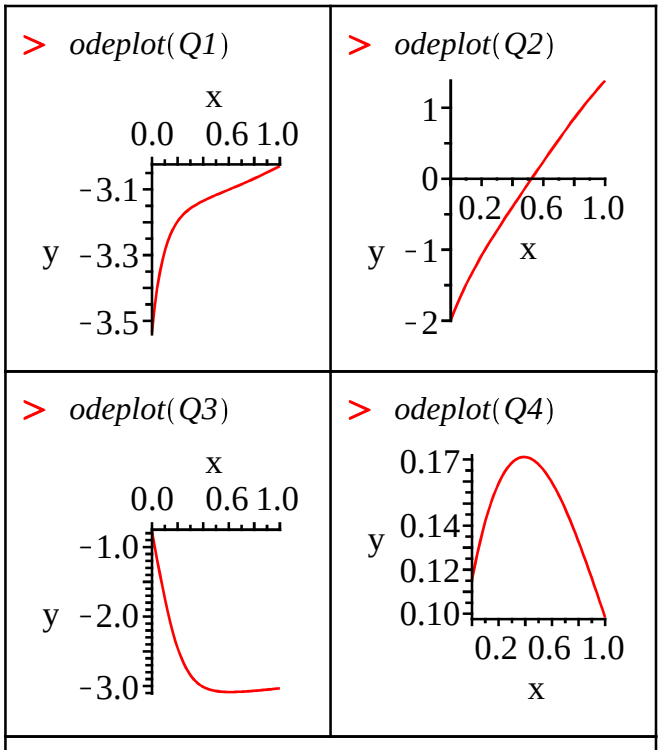


Figure 4 Solutions obtained by **dsolve** using the approxsoln parameter

The graphs in Figures 3 and 4 are essentially the same.

Minimizing a Sum of Squares

A pair (a, b) that minimizes the sum of squares

> $SS := f^2 + g^2$

$$SS := ((1+a)\,b - 3\,a)^2 + \left(\left(1 + h(a,b)_1\right) h(a,b)_2 - 3\,h(a,b)_1 + 0.5 - h(a,b)_1^2\right)^2$$

is a solution to the equations $f = g = 0$. We can compute such minimizing pairs with the **NLPSolve** command in Maple's *Optimization* package. This command provides the nonlinear simplex (Nelder-Mead) method, which does not require computation of derivatives. The computed pairs (a, b) and the values of *SS* at the minima are given by the following loop.

```
> for k to 4 do
      SS||k := NLPSolve(SS, initialpoint = [a = P||k_1, b = P||k_2], method
      = nonlinearsimplex, evaluationlimit = 1000, optimalitytolerance = 1.0 10^-15)
  end do
```

$$SS1 := [4.24383076000000013\ 10^{-16}, [a = -3.53593901734596372, b = 4.18299373881071544]]$$

$$SS2 := [2.78130335809999996\ 10^{-16}, [a = -2.00223622924222688, b = 5.99330629759563838]]$$

$$SS3 := [2.52635688370000019\ 10^{-16}, [a = -0.773424077481638550, b = -10.2405948453594214]]$$

$$SS4 := [7.66794320149520000\ 10^{-16}, [a = 0.115558396306133294, b = 0.310763844991456040]]$$

The first number returned in each solution is the value of *SS* at the computed extremum.

To facilitate a comparison with the values computed by Broyden's method, we express the nonlinear simplex solutions as the vectors

```
> for k to 4 do
  V||k := eval(⟨a, b⟩, SS||k_2)
  end do
```

$$V1 := \begin{bmatrix} -3.53593901734596372 \\ 4.18299373881071544 \end{bmatrix}$$

$$V2 := \begin{bmatrix} -2.00223622924222688 \\ 5.99330629759563838 \end{bmatrix}$$

$$V3 := \begin{bmatrix} -0.773424077481638550 \\ -10.2405948453594214 \end{bmatrix}$$

$$V4 := \begin{bmatrix} 0.115558396306133294 \\ 0.310763844991456040 \end{bmatrix}$$

The Euclidean norms of the differences between the two solutions are given by

```
> for k to 4 do
  Norm(S||k - V||k, 2)
  end do
```

$$7.62090333232966255\ 10^{-9}$$

$$1.05803072908730576\ 10^{-8}$$

$$4.17331013238361704\ 10^{-8}$$

$$1.17736025877342888\ 10^{-8}$$

The agreement is substantial.

Secant Method

If the first boundary condition is solve for $b = b(a)$, we obtain

```
> B := solve( f = 0, b )
```

$$B := \frac{3\,a}{1+a}$$

with which b can be eliminated from the second boundary condition. Thus, we have the single equation

```
> Ga := g |b = B :
  Ga = 0
```

$$\left(1 + h\left(a, \frac{3\,a}{1+a}\right)_1\right) h\left(a, \frac{3\,a}{1+a}\right)_2 - 3\,h\left(a, \frac{3\,a}{1+a}\right)_1 + 0.5 - h\left(a, \frac{3\,a}{1+a}\right)_1^2 = 0$$

to solve. Figure 5 provides a graph of the left-hand side of this equation.

```
> plot( Ga, a = -5..2, -5..5 )
```

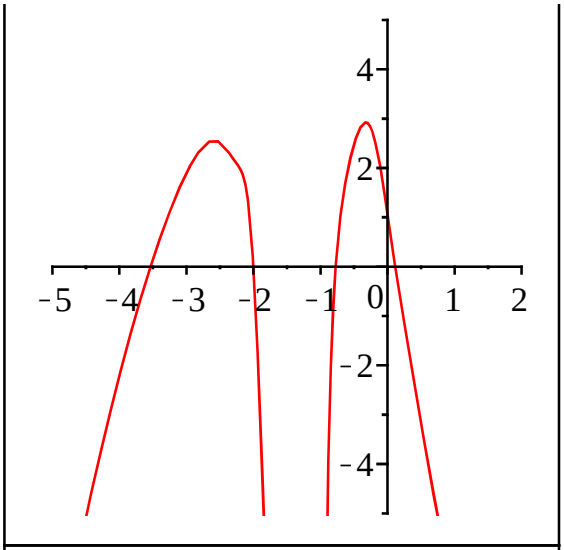


Figure 5 Graph of $g(a, b(a))$

There are four intercepts for the graph in Figure 5, consistent with our earlier findings on the number of solutions of the nonlinear BVP. The equation $g(a, b(a)) = 0$ can be solved with the Secant method, which we have coded as the following procedure. Our procedure includes a calculation of $b(a)$, and returns both a and $b(a)$.

```
> Secant := proc(X0,N)
  local y0,k,x0,x1,x2,y1,X,Y,tol;
  Digits := 30:
  tol := 10^(-N):
  x0 := X0;
  y0 := eval(Ga,a=x0):
  x1 := 1.05*x0:
  unassign('X','Y');
  for k from 1 to 10 do
  y1 := eval(Ga,a=x1);
  x2 := x1 - y1 * (x1 - x0)/(y1 - y0);
  if abs(x2-x1)<tol then X:=evalf[N](x2); Y:=evalf[N](eval(B,a=
  x2));break:
  else x0:=x1; y0:=y1; x1:=x2; y1:=eval(B,a=x1);
  end if;
  end do:
  Digits := 10:
  <X,Y>;
  end proc:
```

An application of the Secant method leads to the following four solutions.

```
> for k to 4 do
```

$W||k := Secant(P||k[1], 20)$
end do

$$W1 := \begin{bmatrix} -3.5359390185068482602 \\ 4.1829937463426818404 \end{bmatrix}$$

$$W2 := \begin{bmatrix} -2.0022362271456943937 \\ 5.9933062872251294658 \end{bmatrix}$$

$$W3 := \begin{bmatrix} -0.77342407614697744961 \\ -10.240594803647667594 \end{bmatrix}$$

$$W4 := \begin{bmatrix} 0.11555840240217400632 \\ 0.31076383491891882363 \end{bmatrix}$$

As we did earlier, we compare these solutions with those computed by Broyden's method.

> **for** k **to** 4 **do**
$Norm(S||k - W||k, 2)$
end do

0.
0.
0.
0.

The solutions appear to be identical.