Solving Boundary Value Problems for Ordinary Differential Equations in Matlab with `bvp4c`

Lawrence F. Shampine [*]   Jacek Kierzenka [†]

Mark W. Reichelt [‡]

October 26, 2000

1   Introduction

Ordinary differential equations (ODEs) describe phenomena that change continuously. They arise in models throughout mathematics, science, and engineering. By itself, a system of ODEs has many solutions. Commonly a solution of interest is determined by specifying the values of all its components at a single point $x = a$. This is an initial value problem (IVP). However, in many applications a solution is determined in a more complicated way. A boundary value problem (BVP) specifies values or equations for solution components at more than one x. Unlike IVPs, a boundary value problem may not have a solution, or may have a finite number, or may have infinitely many. Because of this, programs for solving BVPs require users to provide a guess for the solution desired. Often there are parameters that have to be determined so that the BVP has a solution. Again there might be more than one possibility, so programs require a guess for the parameters desired. Singularities in coefficients and problems posed on infinite intervals are not unusual. Simple examples are used in §2 to illustrate some of these possibilities.

This tutorial shows how to formulate, solve, and plot the solution of a BVP with the Matlab program `bvp4c`. It aims to make solving a typical BVP as easy as possible. BVPs are much harder to solve than IVPs and any solver might fail, even with good guesses for the solution and unknown parameters. `bvp4c` is an effective solver, but the underlying method and the computing environment are not appropriate for high accuracies nor for problems with extremely sharp changes in their solutions. Section 3 describes briefly the numerical method. Section 4 is a collection of examples that illustrate the solution of BVPs with `bvp4c`. The first three should be read in order because they introduce successively features of the solver as it is applied to typical problems. Although

---

[*]Math. Dept., SMU, Dallas, TX 75275 (lshampin@mail.smu.edu)

[†]The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760 (jkierzenka@mathworks.com)

[‡]11 Coolidge Road, Wayland, MA 01778 (reichelt@alum.mit.edu)

1

`bvp4c` accepts quite general BVPs, problems arise in the most diverse forms and they may require some preparation for their solution. The remaining examples illustrate this preparation for common tasks. Some exercises are included for practice. M-files for the solution of all the examples and exercises accompany this tutorial.

2   Boundary Value Problems

If the function $\mathbf{f}$ is smooth on $[a, b]$, the initial value problem $\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$, $\mathbf{y}(a)$ given, has a solution, and only one. Two-point boundary value problems are exemplified by the equation

$$y'' + y = 0 \tag{1}$$

with boundary conditions $y(a) = A$, $y(b) = B$. An important way to analyze such problems is to consider a family of solutions of IVPs. Let $y(x, s)$ be the solution of equation (1) with initial values $y(a) = A$, $y'(a) = s$. Each $y(x, s)$ extends to $x = b$ and we ask, for what values of s does $y(b, s) = B$? If there is a solution s to this algebraic equation, the corresponding $y(x, s)$ provides a solution of the differential equation that satisfies the two boundary conditions. Using linearity we can sort out the possibilities easily. Let $u(x)$ be the solution defined by $y(a) = A$, $y'(a) = 0$ and $v(x)$ be the solution defined by $y(a) = 0$, $y'(a) = 1$. Linearity implies that $y(x, s) = u(x) + sv(x)$, and the boundary condition $B = y(b, s) = u(b) + sv(b)$ amounts to a linear algebraic equation for the unknown initial slope s. The familiar facts of existence and uniqueness of solutions of linear algebraic equations then tell us that there is either exactly one solution to the BVP, or there are boundary values B for which there is no solution and others for which there are infinitely many solutions.

Eigenvalue problems, more specifically Sturm-Liouville problems, are exemplified by

$$y'' + \lambda y = 0$$

with $y(0) = 0$, $y(\pi) = 0$. Such a problem obviously has the trivial solution $y(x) \equiv 0$, but for some values of λ, there are non-trivial solutions. Such λ are called eigenvalues and the corresponding solutions are called eigenfunctions. If $y(x)$ is a solution of this BVP, it is obvious that $\alpha y(x)$ is, too. Accordingly, we need a normalizing condition to specify a solution of interest, for instance $y'(0) = 1$. For $\lambda > 0$, the solution of the IVP with $y(0) = 0$, $y'(0) = 1$ is $y(x) = \sin\left(x\sqrt{\lambda}\right)/\sqrt{\lambda}$. The boundary condition $y(\pi) = 0$ amounts to a non-linear algebraic equation for λ. Generally existence and uniqueness of solutions of nonlinear algebraic equations are difficult matters. For this example the algebraic equation is solved easily to find that the BVP has a non-trivial solution if, and only if, $\lambda = k^2$ for $k = 1, 2, \ldots$. This example shows that when solving a Sturm-Liouville problem, we have to specify not only a normalizing condition, but also which eigenvalue interests us.
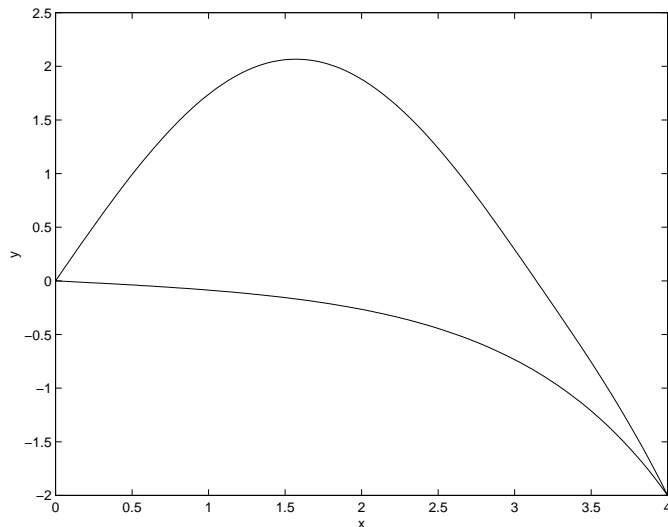
Figure 1: Two solutions for $y'' + |y| = 0$.

Nonlinearity introduces other complications illustrated by the problem [3]

$$y'' + |y| = 0$$

with $y(0) = 0$, $y(b) = B$. Proceeding as with the linear examples, it is found that for any $b > \pi$, there are exactly *two* solutions for any $B < 0$. One solution has the form $y(x, s) = s \sinh x$; it starts off with a negative slope s and decreases monotonely to B. The other starts off with a positive slope where it has the form $y(x, s) = s \sin x$. This solution crosses the axis at $x = \pi$, where its form changes and it decreases thereafter monotonely to B. Figure 1 shows an example of this with $b = 4$ and $B = -2$. Much as with eigenvalue problems, when solving nonlinear BVPs we have to specify which solution is the one that interests us.

Examples in §4 show that BVPs modelling physical situations do not necessarily have unique solutions. Other examples show that problems involving physical parameters might have solutions only for parameter values in certain ranges. The examples make it clear that in practice, solving BVPs may well involve an exploration of the existence and uniqueness of solutions of a model. This is quite different from solving IVPs.

3   Numerical Methods

The theoretical approach to BVPs of §2 is based on the solution of IVPs for ODEs and the solution of nonlinear algebraic equations. Because there are effective programs for both tasks, it is natural to combine them in a program for the solution of BVPs. The approach is called a shooting method. Because

3

it appears so straightforward to use quality numerical tools for the solution of BVPs by shooting, it is perhaps surprising that `bvp4c` is *not* a shooting code. The basic difficulty with shooting is that a perfectly nice BVP can require the integration of IVPs that are unstable. That is, the solution of a BVP can be insensitive to changes in boundary values, yet the solutions of the IVPs of shooting are sensitive to changes in initial values. The simple example

$$y'' - 100y = 0$$

with $y(0) = 1$, $y(1) = B$ makes the point. Shooting involves the solution $y(x,s) = \cosh 10x + 0.1s \sinh 10x$ of the IVP with initial values $y(0) = 1$, $y'(0) = s$. Obviously $\partial y / \partial s = 0.1 \sinh 10x$, which can be as large as $0.1 \sinh 10 \approx 1101$. A little calculation shows that the slope that results in satisfaction of the boundary condition at $x = 1$ is $s = 10(B - \cosh 10)/\sinh 10$ and then that for the solution of the BVP, $|\partial y/\partial B| = |\sinh 10x/\sinh 10| \leq 1$. Evidently the solutions of the IVPs are considerably more sensitive to changes in the initial slope s than the solution of the BVP is to changes in the boundary value B. If the IVPs are not too unstable, shooting can be quite effective. Unstable IVPs can cause a shooting code to fail because the integration "blows up" before reaching the end of the interval. More often, though, the IVP solver reaches the end, but is unable to compute an accurate result there and because of this, the nonlinear equation solver is unable to find accurate initial values. A variety of techniques are employed to improve shooting, but when the IVPs are very unstable, shooting is just not a natural approach to solving BVPs.

`bvp4c` implements a collocation method for the solution of BVPs of the form

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}, \mathbf{p}), \qquad a \leq x \leq b$$

subject to general nonlinear, two-point boundary conditions

$$\mathbf{g}(\mathbf{y}(a), \mathbf{y}(b), \mathbf{p}) = \mathbf{0}$$

Here $\mathbf{p}$ is a vector of unknown parameters. For simplicity it is suppressed in the expressions that follow. The approximate solution $\mathbf{S}(x)$ is a continuous function that is a cubic polynomial on each subinterval $[x_n, x_{n+1}]$ of a mesh $a = x_0 < x_1 < \ldots < x_N = b$. It satisfies the boundary conditions

$$\mathbf{g}(\mathbf{S}(a), \mathbf{S}(b)) = \mathbf{0}$$

and it satisfies the differential equations (collocates) at both ends and the midpoint of each subinterval

$$
\begin{aligned}
\mathbf{S}'(x_n) &= \mathbf{f}(x_n, \mathbf{S}(x_n)) \\
\mathbf{S}'((x_n + x_{n+1})/2) &= \mathbf{f}((x_n + x_{n+1})/2, \mathbf{S}((x_n + x_{n+1})/2)) \\
\mathbf{S}'(x_{n+1}) &= \mathbf{f}(x_{n+1}, \mathbf{S}(x_{n+1}))
\end{aligned}
$$

These conditions result in a system of nonlinear algebraic equations for the coefficients defining $\mathbf{S}(x)$. In contrast to shooting, the solution $\mathbf{y}(x)$ is approximated over the whole interval $[a, b]$ and the boundary conditions are taken into

account at all times. The nonlinear algebraic equations are solved iteratively by linearization, so this approach relies upon the linear equation solvers of MATLAB rather than its IVP codes. The basic method of `bvp4c`, which we call Simpson's method, is well-known and is found in a number of codes. It can be shown [8] that with modest assumptions, $\mathbf{S}(x)$ is a fourth order approximation to an isolated solution $\mathbf{y}(x)$, i.e., $\|\mathbf{y}(x) - \mathbf{S}(x)\| \leq Ch^4$. Here h is the maximum of the step sizes $h_n = x_{n+1} - x_n$ and C is a constant. Because it is not true of some popular collocation methods, we stress the important fact that this bound holds for all x in $[a, b]$. After $\mathbf{S}(x)$ is computed on a mesh with `bvp4c`, it can be evaluated inexpensively at any x, or set of x, in $[a, b]$ with the `bvpval` function.

Because BVPs can have more than one solution, BVP codes require users to supply a guess for the solution desired. The guess includes a guess for an initial mesh that reveals the behavior of the desired solution. The codes then adapt the mesh so as to obtain an accurate numerical solution with a modest number of mesh points. Coming up with a sufficiently good guess is often the hardest part of solving a BVP. `bvp4c` takes an unusual approach to the control of error that helps it deal with poor guesses. The continuity of $\mathbf{S}(x)$ on $[a, b]$ and collocation at the ends of each subinterval imply that $\mathbf{S}(x)$ also has a continuous derivative on $[a, b]$. For such an approximation, the residual $\mathbf{r}(x)$ in the ODEs is defined by

$$\mathbf{r}(x) = \mathbf{S}'(x) - \mathbf{f}(x, \mathbf{S}(x))$$

Put differently, this says that $\mathbf{S}(x)$ is the exact solution of the perturbed ODEs

$$\mathbf{S}'(x) = \mathbf{f}(x, \mathbf{S}(x)) + \mathbf{r}(x)$$

Similarly, the residual in the boundary conditions is $\mathbf{g}(\mathbf{S}(a), \mathbf{S}(b))$. `bvp4c` controls the sizes of these residuals. If the residuals are uniformly small, $\mathbf{S}(x)$ is a good solution in the sense that it is the exact solution of a problem close to the one supplied to the solver. Further, for a reasonably well-conditioned problem, small residuals imply that $\mathbf{S}(x)$ is close to $\mathbf{y}(x)$, even when h is not small enough that the fourth order convergence is evident. Shooting codes can also be described as controlling the sizes of these residuals: at each step an IVP code controls the local error, which is equivalent to controlling the size of the residual of an appropriate continuous extension of the formula used, and the nonlinear equation solver is used to find initial values for which the residual in the boundary conditions is small. Residual control has important virtues: residuals are well-defined no matter how bad the approximate solution, and residuals can be evaluated anywhere simply by evaluating $\mathbf{f}(x, \mathbf{S}(x))$ or $\mathbf{g}(\mathbf{S}(a), \mathbf{S}(b))$. `bvp4c` is based on algorithms that are plausible even when the initial mesh is very poor, yet furnish the correct results as $h \to 0$. They exploit some very interesting properties of the Simpson method shown in [8].

BVPs arise in the most diverse forms. Just about any BVP can be formulated for solution with `bvp4c`. The first step is to write the ODEs as a system of first order ODEs. This is a familiar task because it must also be done for the IVP solvers of MATLAB. The basic idea is to introduce new variables, one for each

variable in the original problem plus one for each of its derivatives up to one less than the highest derivative appearing. The process is illustrated in [11]. This is all that is necessary when solving an IVP, but BVPs can be much more complicated: As we have seen already, unlike IVPs, boundary value problems do not necessarily have a solution, and when they do, the solution is not necessarily unique. Indeed, BVPs commonly involve finding values of parameters for which the problem does have a solution. Also, singularities of various kinds are not at all unusual. The examples that follow illustrate the possibilities and show how to solve common problems.

4   Examples

In this section a variety of examples taken from the literature are used to illustrate both facts about boundary value problems and their numerical solution and details about how to solve boundary value problems with `bvp4c`. You should go through the first three examples in order because they show how to use the solver. Although `bvp4c` accepts BVPs of exceptionally general form, BVPs arise in such diverse forms that many problems require some preparation for their solution. The remaining examples illustrate this preparation and other aspects of the solver. Some exercises are suggested for practice. The prologues to `bvp4c`, `bvpval`, `bvpinit`, `bvpset`, and `bvpget` provide some information about capabilities not discussed here and details are found in [8]. You can learn more about BVPs and other approaches to their solution from the texts [1, 3, 7, 14]. The article [2] about reformulating BVPs into a standard form is highly recommended.

Example 1

A boundary value problem consists of a set of ordinary differential equations, some boundary conditions, and a guess that indicates which solution is desired. An example in [1] for the multiple shooting code MUSN is

$$
\begin{aligned}
u' &= 0.5u(w-u)/v \\
v' &= -0.5(w-u) \\
w' &= (0.9 - 1000(w-y) - 0.5w(w-u))/z \\
z' &= 0.5(w-u) \\
y' &= -100(y-w)
\end{aligned}
\tag{2}
$$

subject to boundary conditions $u(0) = v(0) = w(0) = 1$, $z(0) = -10$, $w(1) = y(1)$. MUSN requires a guess in the form of a function that can be evaluated at

any x in the interval. The guess used in [1] is

$$\begin{aligned}
u(x) &= 1 \\
v(x) &= 1 \\
w(x) &= -4.5x^2 + 8.91x + 1 \\
z(x) &= -10 \\
y(x) &= -4.5x^2 + 9x + 0.91
\end{aligned}$$

To solve this problem with **bvp4c**, you must provide functions that evaluate the differential equations and the residual in the boundary conditions. These functions must return column vectors. With components of **y** corresponding to the original variables as **y(1)**= u, **y(2)**= v, **y(3)**= w, **y(4)**= z, and **y(5)**= y, these functions can be coded in MATLAB as

```
function dydx = ex1ode(x,y)
dydx = [ 0.5*y(1)*(y(3) - y(1))/y(2)
         -0.5*(y(3) - y(1))
         (0.9 - 1000*(y(3) - y(5)) - 0.5*y(3)*(y(3) - y(1)))/y(4)
          0.5*(y(3) - y(1))
          100*(y(3) - y(5))];

function res = ex1bc(ya,yb)
res = [ ya(1) - 1
        ya(2) - 1
        ya(3) - 1
        ya(4) + 10
        yb(3) - yb(5)];
```

The guess is supplied to **bvp4c** in the form of a structure. Although the name **solinit** will be used throughout this tutorial, you can call it anything you like. However, it must contain two fields that must be called **x** and **y**. A guess for a mesh that reveals the behavior of the solution is provided as the vector **solinit.x**. A guess for the solution at these mesh points is provided as the array **solinit.y**, with each column **solinit.y(:,i)** approximating the solution at the point **solinit.x(i)**. It is not difficult to form a guess structure, but a helper function **bvpinit** makes it easy in the most common circumstances. It creates the structure when given the mesh and a guess for the solution in the form of a constant vector or the name of a function for evaluating the guess. For example, the structure is created for a mesh of five equally spaced points in $[0, 1]$ and a constant guess for the solution by

```
solinit = bvpinit(linspace(0,1,5),[1 1 1 -10 0.91]);
```

7

(As a convenience, `bvpinit` accepts both row and column vectors.) This constant guess for the solution is good enough for `bvp4c` to solve the BVP, but the example program `ex1bvp.m` uses the same guess as MUSN. It is evaluated in the

```
function v = ex1init(x)
v = [ 1
      1
      -4.5*x^2+8.91*x+1
      -10
      -4.5*x^2+9*x+0.91];
```

The guess structure is then formed with `bvpint` by

```
solinit = bvpinit(linspace(0,1,5),@ex1init);
```

The boundary value problem has now been defined by means of functions for evaluating the differential equations and the boundary conditions and a structure providing a guess for the solution. When default values are used, that is all you need to solve the problem with `bvp4c`:

```
sol = bvp4c(@ex1ode,@ex1bc,solinit);
```

The output of `bvp4c` is a structure called here `sol`. The mesh determined by the code is returned as `sol.x` and the numerical solution approximated at these mesh points is returned as `sol.y`. As with the guess, `sol.y(:,i)` approximates the solution at the point `sol.x(i)`. Figure 2 compares results computed with MUSN to the curves produced by `bvp4c` in `ex1bvp.m`. The fourth component has been shifted up by 10 to display all the solution components on the same scale.

Exercise: Bratu's equation arises in a model of spontaneous combustion and is mathematically interesting as an example of bifurcation simple enough to solve in a semi-analytical way, see e.g. [5] where it is studied as a nonlinear integral equation. The differential equation is

$$y'' + \lambda \exp(y) = 0$$

with boundary conditions $y(0) = 0 = y(1)$. Depending on the value of the parameter λ, there are two solutions, one solution, or none. There are two solutions when $\lambda = 1$ that you can obtain easily with appropriate guesses. A complete solution is found in `bratubvp.m`.
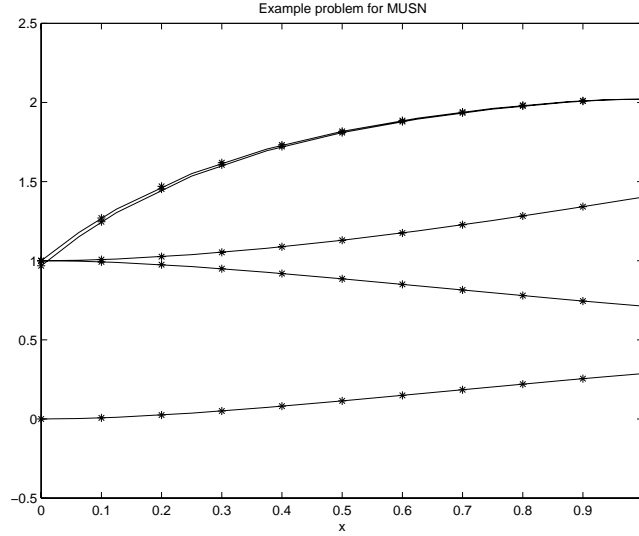
8

Figure 2: `bvp4c` and MUSN (*)solutions.

Example 2

This example shows how to change default values for `bvp4c`. The differential equation depending on a (known) parameter p,

$$y'' + 3py/(p + t^2)^2 = 0 \tag{3}$$

has an analytical solution $y(t) = t/\sqrt{p + t^2}$. A standard test problem for BVP codes [15] is to solve (3) on $[-0.1, +0.1]$ with boundary conditions

$$y(-0.1) = -0.1/\sqrt{p + 0.01}, \quad y(+0.1) = 0.1/\sqrt{p + 0.01}$$

that lead to this analytical solution. The value $p = 10^{-5}$ is used frequently in tests. This differential equation is linear and so are the boundary conditions. Such problems are special and there are a number of important codes like SUPORT [15] that exploit this, but `bvp4c` does not distinguish linear and nonlinear problems.

To solve the problem with `bvp4c` the differential equation must be written as a system of first order ODEs. When this is done in the usual way, the function `ex2ode` can be coded as

```
function dydt = ex2ode(t,y)
p = 1e-5;
dydt = [ y(2)
        -3*p*y(1)/(p+t^2)^2];
```

9

The residual in the boundary conditions is evaluated by the

```
function res = ex2bc(ya,yb)
p = 1e-5;
yatb = 0.1/sqrt(p + 0.01);
yata = - yatb;
res = [ ya(1) - yata
        yb(1) - yatb ];
```

In `ex2bvp.m` a constant guess based on linear interpolation of the boundary values is specified on an initial mesh of 10 equally spaced points:

```
solinit = bvpinit(linspace(-0.1,0.1,10),[0 10])
```

Having defined the BVP, it is now solved with default values by

```
sol = bvp4c(@ex2ode,@ex2bc,solinit)
```

The analytical solution shows that when p is small, there is a boundary layer at the origin where the solution changes sharply. This region of sharp change makes the BVP a relatively difficult one. The numerical solution and the exact solution evaluated at several points are shown in Figure 3. To resolve better the boundary layer, the code must be told to compute the solution more accurately. This is done just as with the MATLAB codes for initial value problems for ODEs. The relative error tolerance on the residuals is called `RelTol` and the absolute error tolerance is called `AbsTol`. The default values are `RelTol=` 10^{-3} and `AbsTol=` 10^{-6}. Values of optional parameters are set by means of a structure formed with the function `bvpset`. Although the name `options` will be used throughout this tutorial, you can call this structure anything you like. It is communicated to `bvp4c` via an optional argument. The solution shown in Figure 4 was obtained with `RelTol` reduced to 10^{-4}. This was done with the commands

```
options = bvpset('RelTol',1e-4);
sol = bvp4c(@ex2ode,@ex2bc,sol,options);
```

The input argument `sol` here is not a typographical error. The `solinit` formed earlier could be used again, but this computation is done in `ex2bvp.m` after the problem is solved with `RelTol=` 10^{-3}. The solution `sol` computed at this tolerance provides an excellent guess for the mesh and solution when `RelTol=` 10^{-4}. This is a very simple example of a technique called continuation, an important tool for solving difficult problems that is discussed more fully in examples that follow.
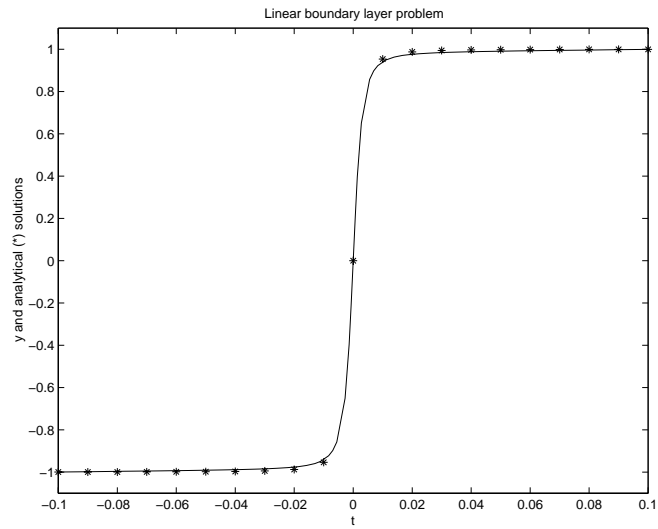
10

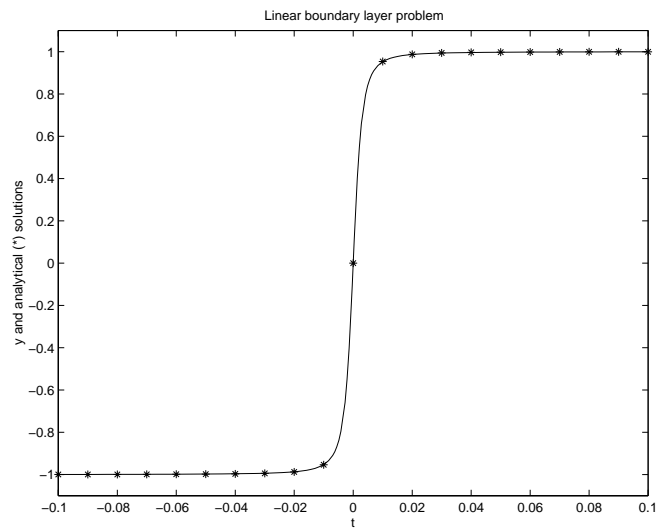Figure 3: Numerical solution obtained with `RelTol`$= 10^{-3}$.

Figure 4: Numerical solution obtained with `RelTol`$= 10^{-4}$.

11

Example 3

This example illustrates the formulation and solution of a boundary value problem involving an unknown parameter. It also shows how to evaluate the solution anywhere in the interval of integration. The task is to compute the fourth eigenvalue of Mathieu's equation,

$$y'' + (\lambda - 2q \cos 2x)y = 0 \tag{4}$$

on $[0, \pi]$ with boundary conditions $y'(0) = 0$, $y'(\pi) = 0$ when $q = 5$. The solution is normalized so that $y(0) = 1$. Even though all the initial values are known at $x = 0$, the problem requires finding a value for the parameter (eigenvalue) λ that allows the boundary condition $y'(\pi) = 0$ to be satisfied. `bvp4c` makes it easy to solve problems with unknown parameters, but there are additional arguments that have to be specified then, e.g., a guess must be provided for the parameters.

This problem is used to illustrate the code D02KAF in the NAG library [12]. Because D02KAF solves only Sturm-Liouville problems, it can exploit the large body of theory about such problems and their numerical solution [13]. As a consequence it is able to compute a particular eigenvalue. The `bvp4c` code is for general BVPs, so all it can do is compute the eigenvalue closest to a guess. This BVP can be solved with a constant guess for the eigenfunction, but we can make it much more likely that we compute the desired eigenvalue by supplying a guess for the eigenfunction that has the correct qualitative behavior. The function $\cos(4x)$ satisfies the boundary conditions and has the correct number of sign changes. It and its derivative are provided as a guess for the vector solution by

```
function v = ex3init(x)
v = [ cos(4*x)
     -4*sin(4*x)];
```

When a BVP involves unknown parameters, a vector of guesses for the parameters must be provided as the `parameters` field of `solinit`. The guess structure can be formed easily by providing the vector of guesses for unknown parameters as a third argument to `bvpinit`. With a guess of 15 for the eigenvalue, this is

```
solinit = bvpinit(linspace(0,pi,10),@ex3init,15)
```

When there are unknown parameters, the functions defining the differential equations and the boundary conditions must have an additional input argument, namely the vector of unknown parameters. In the functions

```
function dydx = ex3ode(x,y,lambda)
```

```
q = 5;
dydx = [  y(2)
        -(lambda - 2*q*cos(2*x))*y(1)];

function res = ex3bc(ya,yb,lambda)
res = [ ya(2)
        yb(2)
        ya(1) - 1];
```

`lambda` is the unknown parameter. It is not used in `ex3bc`, but it must be an argument.

In summary, the only complication introduced by unknown parameters is that a vector of guesses for the parameters must be provided and the functions defining the differential equations and boundary conditions must have the vector of unknown parameters as an additional argument. The solution shown in Figure 5 was obtained with

```
sol = bvp4c(@ex3ode,@ex3bc,solinit);
```

The computed values for the unknown parameters are returned in the field `sol.parameters`. When D02KAF is given an initial guess of $\lambda = 15$, it reports the computed eigenvalue to be 17.097; the same value is computed with `ex3bvp.m`.

The cost of solving a BVP with `bvp4c` depends strongly on the number of mesh points needed to represent the solution to the specified accuracy, so it tries to minimize this number. In previous examples the solution at the mesh points was plotted. When this is done in Figure 5, it is seen that the graph is not smooth at the ends of the interval. The values at mesh points are emphasized to show more clearly that `plot` draws a straight line between successive data points. The solution $\mathbf{S}(x)$ computed by `bvp4c` is continuous and has a continuous derivative on all of $[0, \pi]$. To get a smooth graph, we just need to evaluate it at more points. The function `bvpval` is used to evaluate $\mathbf{S}(x)$ at any x, or set of x, in $[0, \pi]$. Figure 6 is a plot of the solution evaluated at 100 equally spaced points in $[0, \pi]$ with the commands

```
xint = linspace(0,pi);
Sxint = bvpval(sol,xint);
```

Exercise: This problem is studied in section 5.4 of B.A. Finlayson [6]. It arises when modelling a tubular reactor with axial dispersion. An isothermal situation with n-th order irreversible reaction leads to the differential equation

$$y'' = Pe(y' - Ry^n)$$

13

Here Pe is the axial Peclet number and R is the reaction rate group. The boundary conditions are $y'(0) = Pe(y(0) - 1)$, $y'(1) = 0$. Using an orthogonal collocation method, Finlayson finds that $y(0) = 0.63678$ and $y(1) = 0.45759$ when $Pe = 1$, $R = 2$, and $n = 2$. These values are consistent with those obtained by others using a finite difference method. Solve this problem yourself. Use `bvpval` to evaluate the solution at enough points to get a smooth graph of $y(x)$. A complete solution is found in `trbvp.m`.
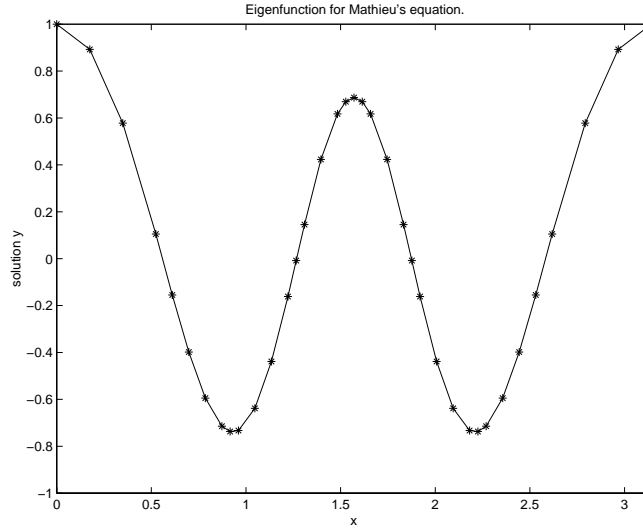

Figure 5: Numerical solution at mesh points only.

Example 4

Boundary conditions that involve the approximate solution only at one end or the other of the interval are called separated boundary conditions. This is generally the case; indeed, all the examples so far have separated boundary conditions. The most common example of non-separated boundary conditions is periodicity. Most BVP solvers accept only problems with separated boundary conditions, so some preparation is necessary in order to solve a problem with non-separated boundary conditions. This example involves the computation of a periodic solution of a set of ODEs. `bvp4c` accepts problems with general, non-separated boundary conditions, so the periodicity does not cause any complication for this solver. However, for this example the period is unknown, so some preparation is necessary for its solution.
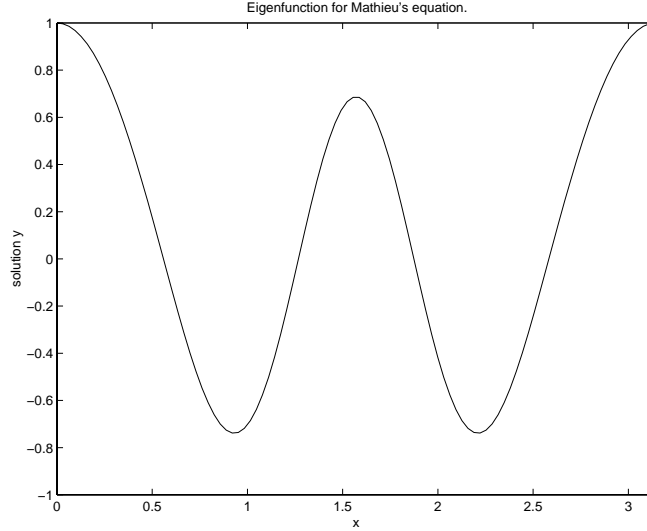
14

Figure 6: Solution evaluated on a finer mesh with `bvpval`.

In [16] the propagation of nerve impulses is described by

$$
\begin{aligned}
y_1' &= 3\left(y_1 + y_2 - \frac{1}{3}y_1^3 - 1.3\right) \\
y_2' &= -\left(y_1 - 0.7 + 0.8y_2\right)/3
\end{aligned}
$$

subject to periodic boundary conditions

$$
y_1\left(0\right) = y_1\left(T\right), \qquad y_2\left(0\right) = y_2\left(T\right)
$$

The difficulty here is that the period T is unknown. If we change the independent variable t to $\tau = t/T$, the differential equations become

$$
\begin{aligned}
\frac{dy_1}{d\tau} &= 3T\left(y_1 + y_2 - \frac{1}{3}y_1^3 - 1.3\right) \\
\frac{dy_2}{d\tau} &= -T\left(y_1 - 0.7 + 0.8y_2\right)/3
\end{aligned}
$$

The problem is now posed on the fixed interval $[0, 1]$ and the (non-separated) boundary conditions are

$$
y_1(0) = y_1\left(1\right), \qquad y_2\left(0\right) = y_2\left(1\right)
$$

An additional condition is necessary to determine the unknown parameter T. This so-called phase condition is chosen to eliminate the degenerate solutions $y'\left(t\right) \equiv 0$ and solutions with $T = 0$. In `ex4bc` we use $dy_2/d\tau\left(0\right) = 1$, i.e.,

$$
-T\left(y_1(0) - 0.7 + 0.8y_2(0)\right)/3 = 1
$$

15

The differential equations and the boundary conditions are coded in MATLAB as

```
function dydt = ex4ode(t,y,T);
dydt = [ T*3*(y(1) + y(2) - 1/3*(y(1)^3) - 1.3)
         T*(-1/3)*(y(1) - 0.7 + 0.8*y(2))        ];

function res = ex4bc(ya,yb,T)
res = [ ya(1) - yb(1)
        ya(2) - yb(2)
        T*(-1/3)*(ya(1) - 0.7 + 0.8*ya(2)) - 1];
```

Because a periodic solution is sought, we chose periodic functions as the initial guess evaluated in

```
function v = ex4init(x)
v = [ sin(2*pi*x)
      cos(2*pi*x)];
```

The length of the period was guessed to be 2π. The solution shown in Figure 7 was obtained with the commands

```
solinit = bvpinit(linspace(0,1,5),@ex4init,2*pi);
sol = bvp4c(@ex4ode,@ex4bc,solinit);
```

Before the solution was plotted, the independent variable was rescaled to its original value $t = T\tau$

```
T = sol.parameters
t = T*sol.x
```

The period was found to be $T = 10.71$. The plot shows that the initial guess was poor.

Example 5

This example illustrates the straightforward solution of a problem set on an infinite interval. Cebeci and Keller [4] use shooting methods to solve the Falkner-Skan problem that arises from a similarity solution of viscous, incompressible, laminar flow over a flat plate. The differential equation is
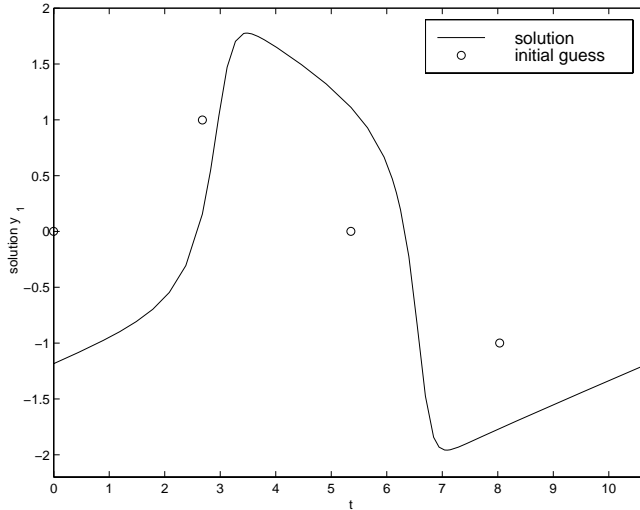
$$f''' + ff'' + \beta(1 - (f')^2) = 0 \tag{5}$$

16

Figure 7: Periodic solution of the nerve impulse model.

The boundary conditions are $f(0) = 0$, $f'(0) = 0$, and $f'(\eta) \to 1$ as $\eta \to \infty$. A straightforward approach used by Cebeci and Keller is to replace the boundary condition at infinity with one at a finite point.

Like Cebeci and Keller, we first write equation (5) as a system of three first order equations with variables f, $u = f'$, and $v = f''$. The nature of the flow depends on the physical parameter β. They report that for accelerating flows, $\beta > 0$, physically relevant solutions exist only for $-0.19884 \leq \beta \leq 2$, illustrating a point made in §2. A relatively difficult case, $\beta = 0.5$, is solved with the boundary condition $f'(6) = 1$ in `ex5bvp.m`. It is found that $f''(0) = 0.92768$, in agreement with the value 0.92768 reported by Cebeci and Keller. The plot of $f'(\eta)$ in Figure 8 shows that it approaches 1 very rapidly as $\eta \to 6$, so taking 6 as "infinity" appears to be reasonable for this problem.

It is prudent to consider a range of values for "infinity". For example, you might solve this problem on $[0, 4]$, $[0, 5]$, and $[0, 6]$, comparing the values of $f''(0)$ and the graphs of $f'(\eta)$ for each interval. Starting with a "large" value for "infinity" is tempting, but not a good tactic. This is made evident by the code failing with the crude guess and default error tolerances of `ex5bvp.m`. Boundary conditions at infinity pick out which solutions of the ODEs are candidates for solution of the BVP. If solutions of the ODEs come together very quickly as the singular point is approached, a solver will have difficulty distinguishing them numerically and so have difficulty solving the BVP. To solve a problem with a singularity, it may be necessary to sort out analytically the behavior of solutions of the ODEs near the singularity. This is illustrated by examples that follow.

Often it is hard to come up with a sufficiently good guess for the solution of a BVP. When physical parameters are present, one of the most effective ways to do this is to solve a sequence of problems starting with a set of parameter
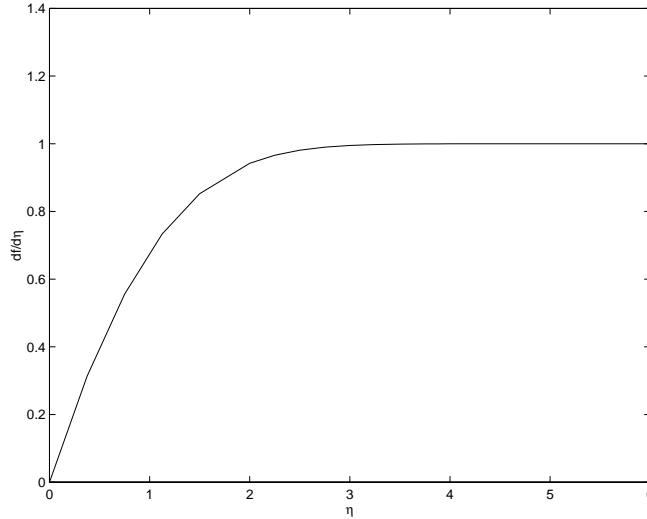
17

Figure 8: Falkner-Skan equation, positive wall shear, $\beta = 0.5$.

values for which the problem is easy to solve and using the result for one set of values as initial guess for the solution of a problem with parameter values that are only a little different. This is repeated until you reach the parameter values of interest. The tactic is called continuation. It is particularly natural when it is of physical interest to compute the solution for a range of parameter values. Cebeci and Keller use this in making up a table of values for solutions of the Falkner-Skan problem. Continuation is not needed to solve this problem with `bvp4c` for the single value of β considered in `ex5bvp.m`.

Exercise: Example 7.3 of [3] considers a similarity solution for the unsteady flow of a gas through a semi-infinite porous medium initially filled with gas at a uniform pressure. The BVP is

$$w''(z) + \frac{2z}{\sqrt{1 - \alpha w(z)}} w'(z) = 0$$

with $w(0) = 1$, $w(\infty) = 0$. A range of values of the parameter α is considered when this problem is solved numerically in Example 8.4 of [3]. Solve this problem for $\alpha = 0.8$ by replacing the boundary condition at infinity with one at a finite point. A complete solution is found in `gasbvp.m`.

Example 6

This example illustrates the straightforward solution of a problem with a coordinate singularity. The problem has three solutions. This BVP is Example 2 of a collection of test problems assembled by M. Kubiček et al. [9]. It arises in a study of heat and mass transfer in a porous spherical catalyst with a first order

18

reaction. There is a singular coefficient arising from the reduction of a partial differential equation to an ODE by symmetry.

The differential equation is

$$y'' + \frac{2}{x}y' = \phi^2 y \exp\left(\frac{\gamma\beta(1-y)}{1+\beta(1-y)}\right) \tag{6}$$

One boundary condition is $y(1) = 1$. The differential equation is singular at $x = 0$, but the singular coefficient arises from the coordinate system and we expect a smooth solution for which symmetry implies that $y'(0) = 0$. We must deal with the singularity in the coefficient at $x = 0$ because bvp4c always evaluates the ODEs at the end points. If we let $x \to 0$ in the equation, we find that

$$y''(0) + 2y''(0) = \phi^2 y(0) \exp\left(\frac{\gamma\beta(1-y(0))}{1+\beta(1-y(0))}\right)$$

because $y'(x)/x \to y''(0)$ then. Solving for $y''(0)$, we obtain the value that must be used in the function for evaluating the ODEs when $x = 0$. For some problems it is necessary to work out more terms in a Taylor series expansion and use them to compute the solution at a small distance from the origin. This example illustrates the fact that often providing the correct value at the singular point is enough. For the present example the parameters ϕ, γ, β are communicated to the solver as additional parameters f, g, b. The function is written in straightforward way:

```
function dydx = ex6ode(x,y,f,g,b)
yp = [y(2); 0];
temp = f^2 * y(1) * exp(g*b*(1-y(1))/(1+b*(1-y(1))));
if x == 0
  dydx(2) = (1/3)*temp;
else
  dydx(2) = -2*(y(2)/x) + temp;
end
```

and the solver is called with additional input arguments

```
sol = bvp4c(@ex6ode,@ex6bc,solinit,options,f,g,b);
```

Kubiček et alia consider a range of parameter values. The values $\phi = 0.6$, $\gamma = 40$, $\beta = 0.2$ used in ex6bvp.m lead to three solutions that are displayed in Figure 9. The example program also compares the initial values $y(0)$ computed for these solutions to values reported in [9].
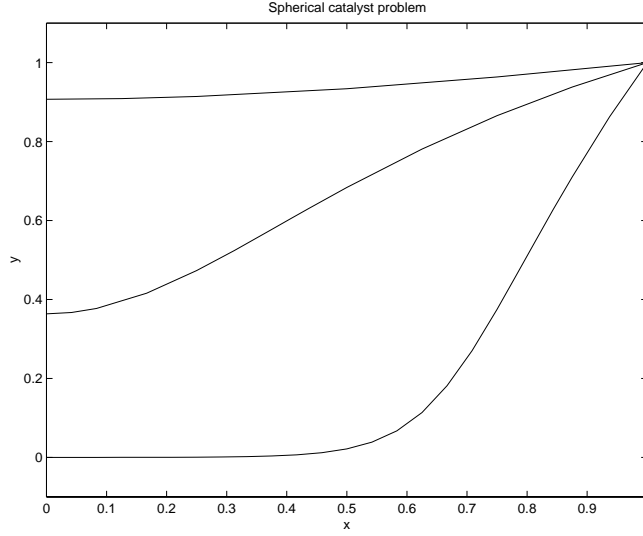
19

Figure 9: Problem with multiple solutions.

Example 7

This example illustrates the handling of a singular point. The idea is to sort out the behavior of solutions near the singular point by analytical means, usually some kind of convergent or asymptotic series expansion. The analytical approximation is used near the singular point and the solution is approximated elsewhere by numerical means. Generally this introduces unknown parameters, an important reason for making it easy to solve such problems with bvp4c. The straightforward approach of the preceding example approximates the solution at the singular point only. It relies on the solution being sufficiently smooth that the code will not need to evaluate the ODEs so close to the singular point that it gets into trouble. In this example, the solution is not smooth and it is necessary to deal with the singularity analytically. It is the first example of the documentation for the D02HBF code of [12]. The equation is

$$y'' = \frac{y^3 - y'}{2x} \tag{7}$$

and the boundary conditions are $y(0) = 0.1$, $y(16) = 1/6$. The singularity at the origin is handled by using series to represent the solution and its derivative at a "small" distance $d > 0$, namely

$$
\begin{aligned}
y(d) &= 0.1 + y'(0)\frac{\sqrt{d}}{10} + \frac{d}{100} + \dots \\
y'(d) &= \frac{y'(0)}{20\sqrt{d}} + \frac{1}{100} + \dots
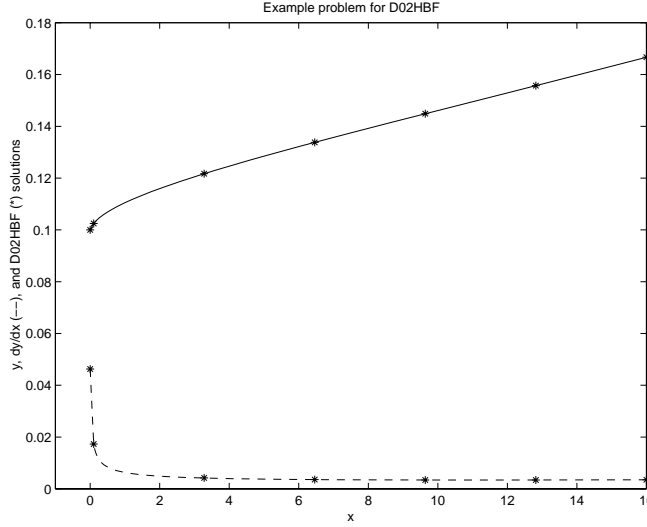\end{aligned}
$$

20

Figure 10: Problem with singular behavior at the origin.

The unknown value $y'(0)$ is treated as an unknown parameter p. The problem is solved numerically on $[d, 16]$. Two boundary conditions are that the computed solution and its first derivative agree with the values from the series at d. The remaining boundary condition is $y(16) = 1/6$. The choice of d must be balanced between a small value that provides an accurate representation of the solution with just a few terms from a series and a large value that avoids difficulties with the singularity. The value used in `ex7bvp.m` is $d = 0.1$.

D02HBF [12] requires given values or guesses for all components at both ends of the interval and all parameters. It is guessed that $y'(16) = 0$ and $y'(0) = 0.2$. In `ex7bvp.m` the same guess is used for $p = y'(0)$ and the solution is guessed to be a constant vector $[1, 1]$. The results from the documentation for D02HBF are compared in `ex7bvp.m` to curves produced by `bvp4c`. To obtain a solution on all of $[0, 16]$, the numerical solution on $[d, 16]$ is augmented by $y(0) = 0.1$, $y'(0) = p$. If necessary for a smooth graph, other values in $[0, d]$ could be obtained from the series. Figure 10 shows the sharp change in the solution at the origin that is seen analytically in the series.

Exercise: Section 6.2 of [7] discusses the numerical solution of a model of the steady concentration of a substrate in an enzyme-catalyzed reaction with Michaelis-Menten reaction rate. A spherical region is considered and the partial differential equation is reduced to an ODE by symmetry. The differential equation is

$$y'' + \frac{2}{x}y' = \frac{y}{\epsilon(y + k)}$$

where there are two physical parameters ϵ and k. The boundary conditions are $y(1) = 1$ and the symmetry condition $y'(0) = 0$. It is possible to solve this

21

BVP in the straightforward manner of Example 6, though there is a potential difficulty of computing a non-physical solution with negative values as shown in Keller's Figure 6.2.2. As an exercise you should solve the BVP by analyzing the behavior of the solution at $x = 0$. Because we expect a smooth solution, approximate it with a Taylor series expansion about $x = 0$. Show that

$$
\begin{aligned}
y(x) &= y(0) + 0x + \frac{y''(0)}{2}x^2 + \dots \\
y'(x) &= 0 + y''(0)x + \dots
\end{aligned}
$$

where

$$
y''(0) = \frac{y(0)}{3\epsilon(y(0) + k)}
$$

For parameters $\epsilon = 0.1$ and $k = 0.1$, solve the problem on $[d, 1]$ for $d = 0.001$ with the boundary condition $y(1) = 1$ and boundary conditions that require the numerical solution to agree with the expansions at $x = d$. You will have to introduce an unknown parameter $p = y(0)$. To plot the solution on all of $[0, 1]$, augment the numerical solution on $[d, 1]$ with the values at $x = 0$ provided by $y(0) = p$, $y'(0) = 0$. A complete solution is provided by `mmbvp.m`.

Example 8

This example uses continuation to solve a difficult problem. Example 1.4 of [1] describes flow in a long vertical channel with fluid injection through one side. The ODEs are

$$
\begin{aligned}
f''' - R\left[(f')^2 - f\,f''\right] + R\,A &= 0 \\
h'' + R\,f\,h' + 1 &= 0 \\
\theta'' + P\,f\,\theta' &= 0
\end{aligned}
$$

In the text [1] the problem is reformulated to deal with the unknown parameter A. This is not necessary for `bvp4c`, though it is necessary to write the ODEs as a system of seven first order differential equations. Here R is the Reynolds number and $P = 0.7\,R$. Because of the presence of the (scalar) unknown A, this system is subject to eight boundary conditions

$$
\begin{aligned}
f(0) &= f'(0) = 0, \quad f(1) = 1, \quad f'(1) = 1 \\
h(0) &= h(1) = 0 \\
\theta(0) &= 0, \quad \theta(1) = 1
\end{aligned}
$$

The differential equations and boundary conditions functions are

```
function dydx = ex8ode(x,y,A,R);
P = 0.7*R;
```

```
    dydx = [ y(2)
             y(3)
             R *( y(2)^2 - y(1)*y(3) - A )
             y(5)
             -R*y(1)*y(5) - 1
             y(7)
             -P*y(1)*y(7) ];

  function res = ex8bc(ya,yb,A,R)
  res = [ ya(1)
          ya(2)
          yb(1) - 1
          yb(2)
          ya(4)
          yb(4)
          ya(6)
          yb(6) - 1 ];
```

Note that the known parameter R follows the unknown parameter A on the argument lists. For $R = 100$, the BVP can be solved without difficulty with the guess structure

```
  solinit = bvpinit(linspace(0,1,10),ones(7,1),1);
```

However, when $R = 10000$, `bvp4c` fails with this guess. For a large Reynolds number the solution changes very rapidly near $x = 0$, i.e., there is a boundary layer there. Generally `bvp4c` is able to cope with poor guesses for the mesh, but when very sharp changes in the solution are present, you may need to help it with a guess that reveals the regions of sharp change. In [1] it is suggested that a sequence of problems be solved with the mesh and solution for one value of R used as initial guess for a larger value. This is continuation in the (known) parameter R. As in previous examples, you might actually want solutions for a range of R, but here the tactic is needed to get guesses good enough that `bvp4c` can compute solutions for large Reynolds numbers. Continuation is easy with `bvp4c` because the structure for guesses is exactly the same as that for solutions. Accordingly, in `ex8bvp.m` we solve the problem for $R = 100$ using `solinit` as stated above. The solution for one value of R is then used as guess for the BVP with R increased by a factor of 10. `ex8bvp.m` is a comparatively expensive computation because BVPs are solved for the three Reynolds numbers $R = 100, 1000, 10000$; a fine mesh is needed for the larger Reynolds numbers; and there are seven ODEs and one unknown parameter. Still, solving this BVP with `bvp4c` is routine except for the use of continuation to get a sufficiently good guess. The three solutions computed in `ex8bvp.m` are shown in Figure 11. It might be remarked that the BVP with $R = 10000$ was solved on a mesh of 91 points.
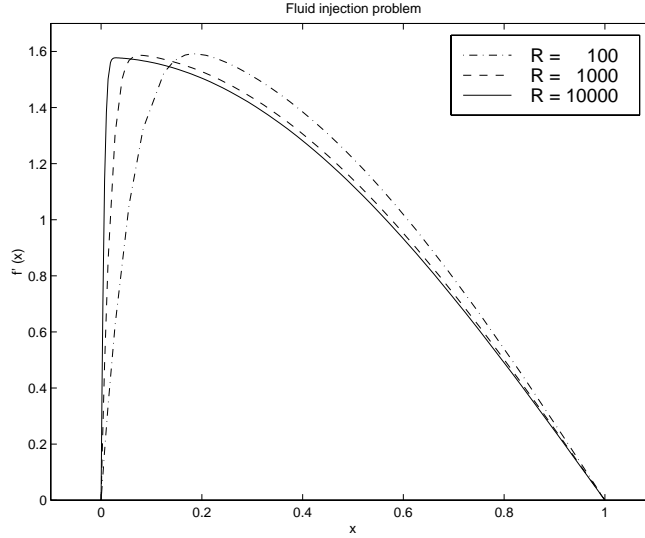
Figure 11: Solution obtained by continuation.

Example 9

This example introduces multipoint BVPs. Chapter 8 of [10] is devoted to the study of a physiological flow problem. After considerable preparation Lin and Segel arrive at equations that can be written for $0 \leq x \leq \lambda$ as

$$
\begin{aligned}
v' &= (C - 1)/n \\
C' &= (vC - \min(x, 1))/\eta.
\end{aligned} \tag{8}
$$

Here n and η are dimensionless (known) parameters and $\lambda > 1$. The boundary conditions are $v(0) = 0$, $C(\lambda) = 1$. The quantity of most interest is the dimensionless emergent osmolarity $Os = 1/v(\lambda)$. Using perturbation methods, Lin and Segel approximate this quantity for small n by $Os \approx 1/(1 - K_2)$ where $K_2 = \lambda \sinh(\kappa/\lambda)/(\kappa \cosh(\kappa))$. The parameter κ here is such that $\eta = \lambda^2/(n\kappa^2)$.

The term $\min(x, 1)$ in the equation for $C'(x)$ is not smooth at $x = 1$. Indeed, Lin and Segel describe this BVP as two problems, one set on $[0, 1]$ and the other on $[1, \lambda]$, connected by the requirement that the functions $v(x)$ and $C(x)$ be continuous at $x = 1$. Numerical methods do not have their usual order of convergence when the ODEs are not smooth. Despite this, `bvp4c` is sufficiently robust that it can solve the problem formulated in this way without difficulty. That is certainly the easier way to solve this particular problem, but it is better practice to recognize that this is a multipoint BVP. In particular, it is a three-point BVP because it involves boundary conditions at three points rather than the two that we have seen in all the other examples. `bvp4c` accepts only two-point BVPs. There are standard ways of reformulating a multipoint BVP into a two-point BVP discussed in [2] and [1]. The usual way is first to introduce

24

unknowns $y_1(x) = v(x)$, $y_2(x) = C(x)$ for the interval $0 \leq x \leq 1$, so that the differential equations there are

$$\frac{dy_1}{dx} = (y_2 - 1)/n$$

$$\frac{dy_2}{dx} = (y_1 y_2 - x)/\eta$$

One of the boundary conditions becomes $y_1(0) = 0$. Next, unknowns $y_3(x) = v(x)$, $y_4(x) = C(x)$ are introduced for the interval $1 \leq x \leq \lambda$, resulting in the equations

$$\frac{dy_3}{dx} = (y_4 - 1)/n$$

$$\frac{dy_4}{dx} = (y_3 y_4 - 1)/\eta$$

The other boundary condition becomes $y_4(\lambda) = 1$. With these new variables the continuity conditions on v and C become boundary conditions, $y_1(1) = y_3(1)$ and $y_2(1) = y_4(1)$. This is all easy enough, but the trick is to solve the four differential equations simultaneously. This is accomplished by defining a new independent variable $\tau = (x - 1)/(\lambda - 1)$ for the second interval. Like x in the first interval, this independent variable ranges from 0 to 1 in the second interval. In this new independent variable, the differential equations on the second interval become

$$\frac{dy_3}{d\tau} = (\lambda - 1)(y_4 - 1)/n$$

$$\frac{dy_4}{d\tau} = (\lambda - 1)(y_3 y_4 - 1)/\eta$$

The boundary condition $y_4(x = \lambda) = 1$ becomes $y_4(\tau = 1) = 1$. The continuity condition $y_1(x = 1) = y_3(x = 1)$ becomes $y_1(x = 1) = y_3(\tau = 0)$. Similarly, the other continuity condition becomes $y_2(x = 1) = y_4(\tau = 0)$. Because the differential equations for the four unknowns are connected only through the boundary conditions and both sets are to be solved for an independent variable ranging from 0 to 1, we can combine them as

$$\frac{dy_1}{dt} = (y_2 - 1)/n$$

$$\frac{dy_2}{dt} = (y_1 y_2 - x)/\eta$$

$$\frac{dy_3}{dt} = (\lambda - 1)(y_4 - 1)/n$$

$$\frac{dy_4}{dt} = (\lambda - 1)(y_3 y_4 - 1)/\eta$$

to be solved for $0 \leq t \leq 1$. In the common independent variable t, the boundary conditions are $y_1(0) = 0$, $y_4(1) = 1$, $y_1(1) = y_3(0)$, and $y_2(1) = y_4(0)$.

25

Notice that the boundary conditions arising from continuity are not separated because they involve values of the solution at both ends of the interval. Periodic solutions of ODEs and the two-point BVPs resulting from reformulation of multipoint BVPs are the most common sources of non-separated boundary conditions. They cause no complication for `bvp4c`, but most solvers require additional preparation of the problem to separate the boundary conditions.

`ex9bvp.m` solves the three-point BVP for $n = 5 \times 10^{-2}$, $\lambda = 2$, and a range $\kappa = 2, 3, 4, 5$. The solution for one value of κ is used as guess for the next, an example of continuation in a physical parameter. For each κ the computed Os is compared to the approximation of Lin and Segel. Figure 12 shows the solutions $v(x)$ and $C(x)$ for $\kappa = 5$.
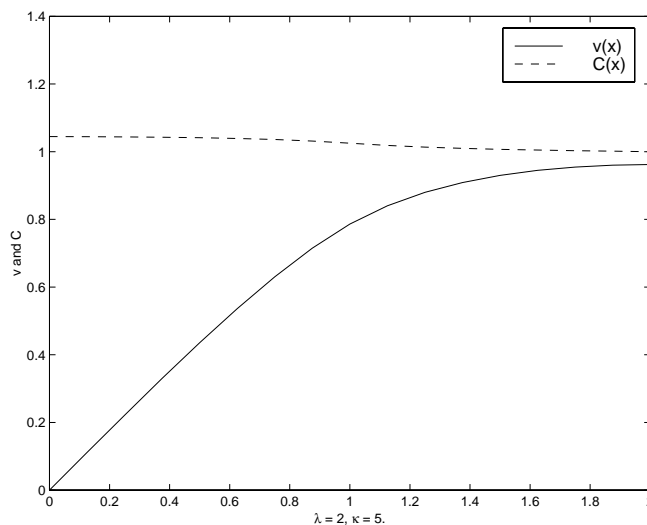

Figure 12: A three-point BVP.

References

[1] U. Ascher, R. Mattheij, and R. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, SIAM, Philadelphia, PA, 1995.

[2] U. Ascher and R. Russell, *Reformulation of boundary value problems into 'standard' form*, SIAM Review, 23 (1981), pp. 238–254.

[3] P. Bailey, L. Shampine, and P. Waltman, *Nonlinear Two Point Boundary Value Problems*, Academic, New York, 1968.

[4] T. CEBECI AND H. KELLER, *Shooting and parallel shooting methods for solving the Falkner-Skan boundary-layer equations*, J. Comp. Physics, 7 (1971), pp. 289–300.

[5] H. DAVIS, *Introduction to Nonlinear Differential and Integral Equations*, Dover, New York, 1962.

[6] B. FINLAYSON, *The Method of Weighted Residuals and Variational Principles*, Academic, New York, 1972.

[7] H. KELLER, *Numerical Methods for Two-Point Boundary-Value Problems*, Dover, New York, 1992.

[8] J. KIERZENKA, *Studies in the Numerical Solution of Ordinary Differential Equations*, PhD thesis, Southern Methodist University, Dallas, TX, 1998.

[9] M. KUBIČEK, V. HLAVÁČEK, AND M. HOLODNICK, *Test examples for comparison of codes for nonlinear boundary value problems in ordinary differential equations*, in Codes for Boundary-Value Problems in Ordinary Differential Equations, Lecture Notes in Computer Science #76, B. Childs et al., ed., Springer, New York, 1979, pp. 325–346.

[10] C. LIN AND L. SEGEL, *Mathematics Applied to Deterministic Problems in the Natural Sciences*, SIAM, Philadelphia, PA, 1988.

[11] THE MATHWORKS, INC., *Using MATLAB*, 24 Prime Park Way, Natick, MA, 1996.

[12] NUMERICAL ALGORITHMS GROUP INC., *NAG FORTRAN 77 Library Manual, Mark 17*, Oxford, UK, 1996.

[13] J. PRYCE, *Numerical Solution of Sturm-Liouville Problems*, Clarendon Press, Oxford, UK, 1993.

[14] S. ROBERTS AND J. SHIPMAN, *Two-Point Boundary Value Problems: Shooting Methods*, Elsevier, New York, 1972.

[15] M. SCOTT AND H. WATTS, *Computational solution of linear two point boundary value problems via orthonormalization*, SIAM J. Numer. Anal., 14 (1977), pp. 40–70.

[16] R. SEYDEL, *From Equilibrium to Chaos*, Elsevier, New York, 1988.