Chapter 3 - Boundary Value Problems

3.1 Introduction

Recall that a system of ODEs has many solutions.

In an IVP, the solution of interest is determined by specifying the values of all the solution components at one point as well as a direction of integration.

In a BVP, there are relationships between solution components at more than one point in the range of the independent variable, to which we now refer as x instead of t to emphasize the lack of directionality in the integration.

As we have noted, IVPs often have unique solutions, but BVPs often have none or more than one.

Because of this, the user must supply an initial guess at the solution of interest.

Often there are sets of parameters that must be determined for the BVP to have a solution.

Again, there may be one set or more than one set!

We mainly consider the so-called *two-point BVP*

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}, \mathbf{p}),\ a \leq x \leq b,\ \mathbf{0} = \mathbf{g}(\mathbf{y}(a), \mathbf{y}(b), \mathbf{p}), \tag{1}$$

where $\mathbf{p}$ is a vector of unknown parameters.

This is called a two-point BVP because the BCs involve the solution at only the 2 end points $x = a$ and $x = b$.

If conditions on the function are given at more than 2 points, then we have a *multi-point BVP*.

The MATLAB BVP solvers are called `bvp4c` and `bvp5c`, and they accept multi-point BVPs directly.

However, many others solvers do not, so we discuss how to convert multi-point BVPs to two-point BVPs.

Note 1. *There is a code called* `bvp6c` *out of the Oxford University Computing Laboratory that is a sixth-order extension of* `bvp4c`*.*

It seems to have all the functionality of `bvp4c` *and* `bvp5c`*, but it is more closely related theoretically to the numerical method in* `bvp4c`*.*

`bvp4c` and `bvp5c` can actually handle a class of singular BVPs of the form

$$\begin{aligned}\mathbf{y}'(x) &= \frac{\mathbf{S}}{x}\mathbf{y} + \mathbf{f}(x,\mathbf{y},\mathbf{p}),\ a \le x \le b,\\ \mathbf{0} &= \mathbf{g}(\mathbf{y}(a),\mathbf{y}(b),\mathbf{p}),\end{aligned}$$

where $\mathbf{S}$ is a constant matrix specified as the value of the `SingularTerm` option of `bvpset`.

Note 2. *The right-hand side function for this problem evaluates only* $\mathbf{f}(x,\mathbf{y},\mathbf{p})$*.*

The boundary conditions must be consistent with the necessary condition $\mathbf{S}\mathbf{y}(a) = \mathbf{0}$, and the initial guess should satisfy this condition as well.

Such problems are often the result of PDEs that have been converted to ODEs by means of a switch to cylindrical or spherical co-ordinates to take advantage of symmetry.

Parameters may arise naturally as the physical model is derived, or they may be added artificially as part of the solution process, e.g., if there are singular coefficients or the problem is posed on an infinite interval.

`bvp4c` and `bvp5c` accept problems with unknown parameters directly; but many other solvers do not, so we discuss how to convert problems with parameters so they can be handled by these other solvers.

Note 3. *For brevity, we generally suppress the explicit dependence of* $\mathbf{p}$ *in (1).*

We have talked about conversion of ODEs to standard (first-order) form, but we also noted that some solvers handle higher-order equations like $\ddot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ directly.

It is even more advantageous to handle high order directly for BVPs, but it complicates the interface, so it is not done in MATLAB.

Many codes also assume the BCs to be *separated*; i.e., each component of $\mathbf{g}$ involves unknowns at only one end point at a time.

We also discuss how to convert problems with non-separated BCs to ones with separated BCs.

It is not always clear what BCs make sense and where they can be applied (cf. the pendulum problem of Section 1.2).

There is considerable art to solving a BVP with a singularity at an end point or one posed on an infinite interval.

BVPs are generally much harder to solve than IVPs!

e.g., any solver can fail even when a good initial guess at the solution and parameters are provided.

Sometimes a solver may find a *pseudo-solution*, i.e., a numerical solution to a problem with no analytical solution.

Although `bvp4c` and `bvp5c` can be effective, no solver is best for all problems.

In particular, the moderate orders of `bvp4c` and `bvp5c` and the PSE of MATLAB make it inappropriate for problems with sharp changes or that require stringent tolerances.

3.2 BVPs

The facts of life for BVPs are much different than they are for IVPs.

For $f(x, y, y')$ sufficiently smooth, the IVP

$$y'' = f(x, y, y'), \quad y(a) = y_a,\ y'(a) = s,$$

has a unique solution $y(x)$ for $x \geq a$.

The exemplary two-point BVP is the linear ODE

$$y'' + y = 0 \tag{2}$$

with separated BCs

$$y(a) = y_a, \qquad y(b) = y_b.$$

A useful way to analyze such problems is to let $y(x;s)$ be the solution of the ODE (2) with initial values $y(a;s) = y_a$ and $y'(a;s) = s$.

With this problem, for each s, $y(x;s)$ satisfies the BC at $x = a$ and exists everywhere between a and b.

So what magical value of s will $y(b;s) = y_b$?

If there is such a magical s that satisfies this *algebraic equation*, we will have solved the BVP.

For a linear problem, this is (relatively) easy to do:

Let $y_1(x)$ be the solution of (2) with ICs $y_1(a) = y_a$ and $y_1'(a) = 0$.

Let $y_2(x)$ be the solution of (2) with ICs $y_2(a) = 0$ and $y_2'(a) = 1$.

Linearity implies that the general solution of (2) is

$$y(x;s) = y_1(x) + sy_2(x).$$

Thus the BC

$$y_b := y(b; s) = y_1(b) + s y_2(b)$$

is a linear algebraic equation that can be solved for the magical initial slope s.

It is easy to deduce now that the BVP will have exactly one solution when $y_2(b) \neq 0$; in this case the magical slope is

$$s = \frac{y_b - y_1(b)}{y_2(b)}.$$

Moreover, if $y_2(b) = 0$, then there are infinitely many solutions when $y_b = y_1(b)$ and no solutions otherwise.

Even in a case with such clear theory, we can appreciate there being numerical difficulties if $y_2(b) \approx 0$.

The extremes of this are computing a “solution” when none exists or concluding there is no solution when there is.

Sturm–Liouville Eigenproblems

Sturm–Liouville (SL) eigenproblems are exemplified by the ODE

$$y'' + \lambda y = 0, \quad 0 \leq x \leq \pi, \tag{3}$$

with non-separated periodic BCs

$$y(0) = y(\pi), \quad y'(0) = y'(\pi),$$

or separated *Dirichlet* BCs

$$y(0) = 0, \quad y(\pi) = 0.$$

For all λ, we have the trivial solution $y(x) \equiv 0$.

We are more interested in the special values of λ (called *eigenvalues*) for which there exist non-trivial solutions $y(x)$ (called *eigenfunctions*).

Note 4. *This is a* nonlinear *BVP because the unknown λ multiplies the unknown $y(x)$.*

We see that if $y(x)$ solves the BVP, then so does $\alpha y(x)$ for any constant α.

This means that to completely specify a solution, we need a normalizing condition; e.g., $y'(0) = 1$.

This is a valid choice because $y'(0) = 0$ leads to the trivial solution; thus if $y'(0) \neq 0$, we can scale $y(x)$ such that $y'(0) = 1$.

We can think of the normalizing condition as another BC needed to determine the unknown parameter λ.

For $\lambda > 0$, the solution of (3) subject to $y(0) = 0$ and $y'(0) = 1$ is

$$y(x) = \frac{\sin(\sqrt{\lambda}x)}{\sqrt{\lambda}}.$$

If we solve the BVP with Dirichlet BCs, the condition $y(\pi) = 0$ amounts to a nonlinear algebraic equation for λ.

Existence and uniqueness of solutions for nonlinear algebraic equations are generally hard to establish.

However, this problem is easy enough that we find the BVP has a non-trivial solution if and only if

$$\lambda = k^2, \qquad k = 1, 2, \ldots .$$

Note 5. *When solving SL problems, we need to know which eigenvalue is of interest!*

Fortunately, so much is known about these problems theoretically that good codes let you specify exactly what you want, e.g., the fifth eigenvalue and its corresponding eigenfunction.

If you wish to use standard BVP software, then you must provide guesses for both the eigenvalue and eigenfunction of interest to you.

This can be tricky!

The use of special-purpose software for these problems is recommended.

Nonlinear BVPs

As usual, nonlinearity introduces further complications.

Consider the ODE

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

with separated BCs

$$y(0) = 0, \qquad y(b) = y_b.$$

If a linear BVP has more than one solution, it must have infinitely many; but a nonlinear BVP may have only a finite number.[1]

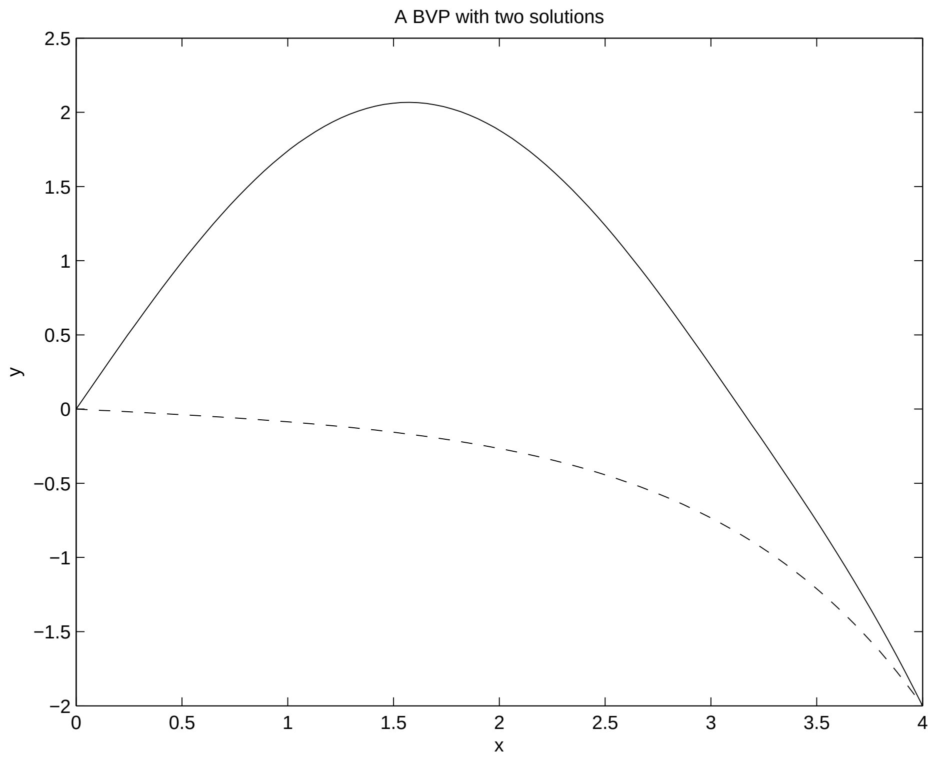
We find that for any $b > \pi$, there are exactly 2 solutions for each $y_b < 0$.

One solution has the form $y(x; s) = s \sinh x$.

[1]This is completely analogous to the number of solutions of a set of linear equations versus a set of nonlinear equations.

The second solution has the form $y(x; s) = s \sin x$.

Here is what the solutions look like for $b = 4$ and $y_b = -2$; see also the MATLAB demonstration program `twobvp.m`.



To produce the figure, we needed 2 initial guesses: $y(x) \equiv -1$ is used to compute the solution that is always negative, and $y(x) \equiv 1$ is used to compute the other one.

In general, *models of physical situations described by BVPs may not have unique solutions!*

Also, problems involving parameters may only have solutions for parameters in certain ranges.

Thus in practice the solution of BVPs may involve a theoretical exploration of existence and uniqueness.

This is in contrast to the solution of IVPs where the local existence and uniqueness of solutions are guaranteed by mild theoretical conditions that are almost always satisfied in practice.

3.3 BCs

It may not be clear what kinds of BCs to use and where they should be applied.

It is common for BVPs to be singular in the sense that either the ODEs are singular at an endpoint or the interval is infinite (or both!).

The standard theory breaks down in such cases, so we have to garner all the insight we can from the underlying physics and the mathematical analysis in order to help solve the problem.

Generally the number of BCs required equals the sum of the orders of the ODEs plus the number of unknown parameters.

This is analogous to the situation with IVPs.

BVP solvers fail if you do not provide the correct number of BCs, i.e., too few or too many.

Here are some typical scenarios when it may be difficult to determine BCs or how to apply them.

- All obvious physical constraints have been imposed, yet there are not enough BCs. In this case you should look for other conditions like a conservation of integrals or normalizing condition.

- Some BCs may only describe the behaviour of the solution, rather than giving a specific condition; e.g., the solution must be bounded at a singular point, or it must decay in a certain way at infinity. It is usually possible to convert these kinds of BCs to standard form, possibly by introducing some unknown parameters.

- There may be too many BCs. You may know more than is necessary to completely specify the BVP. BCs that are consequences of the ODEs and some other BCs should be dropped. Usually these BCs involve higher derivatives of the solution.

3.3.1 BCs at singular points

We begin by discussing singularities at finite points.

This is a common problem when reducing a PDE to an ODE using cylindrical or spherical symmetry.

Consider *Bratu's equation*, which is used to model spontaneous combustion:

$$u_{xx} + u_{yy} + e^y = 0.$$

In the case of cylindrical or spherical symmetry, we get the following ODE for $u = u(r)$

$$u'' + k\frac{u'}{r} + e^u = 0, \quad 0 \le r \le 1,$$

with $k = 1$ for cylindrical symmetry and $k = 2$ for spherical symmetry.

The obvious problem is the term u'/r for $r = 0$.

We expect this problem to be an artifact related to the co-ordinate system and not something fundamental.

Indeed because of symmetry, $u'(0) = 0$, so u'/r is indeterminate, not undefined, near $r = 0$.

In fact for methods that do not need to evaluate the right-hand side at $r = 0$, there is no difficulty solving this problem.

Unfortunately, `bvp4c` and `bvp5c` do evaluate the right-hand side at both ends of each mesh subinterval, so it is less straightforward in this case.

The idea is to use analytical means to approximate the solution near the singular point.[2]

This is where you need to bring in knowledge from outside of numerical analysis.

Often some kind of series or asymptotic approximation is used.

[2]We have seen this as a method for dealing with singularities in IVPs.

For this example, we expect a smooth solution with even symmetry, so we can expand in a Taylor series of the form

$$u(r) = u(0) + \frac{u''(0)}{2} r^2 + \frac{u^{(4)}(0)}{4!} r^4 + \ldots .$$

Substituting this into the ODE

$$(u''(0) + \ldots) + \frac{k}{r}(u''(0)r + \ldots) + e^{u(0)+\cdots} = 0.$$

Equating the leading coefficient to 0, we find

$$u''(0) = -\frac{e^{u(0)}}{k+1},$$

leading to the approximation

$$u(r) \approx u(0) - \frac{e^{u(0)}}{2(k+1)} r^2.$$

This is good enough to approximate $u(r)$ on an interval $[0, \delta]$ for sufficiently small δ.

The derivative of this approximation also provides an approximation to $u'(r)$.

The numerical solution can then be found on $[\delta, b]$ where the equations are not singular.

Note however that $u(0)$ is unknown; this is a typical example of how unknown parameters are introduced while solving BVPs.

A BC tells us how the solution of interest behaves as it approaches an end point; this may not be as simple as approaching a given value.

The BC $u'(0) = 0$ for Bratu's problem tells us that $u \to u_0$ (a constant) as $r \to 0$, but other types of BCs are possible.

It can be more difficult to determine the behaviour of solutions at a singular point.

Consider the BVP

$$yy'' = -1, \qquad y(0) = 0,\ y(1) = 0.$$

For $y(x) \to 0$ as $x \to 0, 1$, we need that $y''(x)$ become unbounded at these points.

First we note that the solution is symmetric about $x = 0.5$, so we can restrict our analytical treatment to the origin.

We must expect (at least) two solutions because if $y(x)$ is a solution, then so is $-y(x)$.

The solution $y(x)$ vanishes at $x = 0$ but has unbounded derivatives there, so we try

$$y(x) \sim ax^b$$

for constants a and b.

Substituting this into the ODE, we find that

$$(ax^b)(ab(b-1)x^{b-2}) = a^2b(b-1)x^{2b-2} = -1.$$

Now if $2b - 2 < 0$, this expression is unbounded as $x \to 0$.

If $2b - 2 > 0$, the limit exists, but it is 0, not -1.

If $2b - 2 = 0$, this expression is identically 0.

This tells us that the form that we have assumed for the solution is wrong!

With some insight, we assume

$$y(x) \sim ax(-\log(x))^b.$$

Substituting this into the ODE, we find that

$$-a^2b(-\log(x))^{2b-1}[1-(b-1)(-\log(x))^{-1}] = -1.$$

For this to have a limit as $x \to 0$ we must have $2b - 1 = 0$ or $b = 1/2$.

From this limit we find $-1 = -a^2(1/2)$ or $a = \pm\sqrt{2}$.

It seems that there are exactly 2 solutions; one will be positive for $x > 0$, and that is $y(x) \sim x\sqrt{-2\log(x)}$.

3.3.2 BCs at infinity

Consider the ODE

$$y''' + 2y'' - y' - 2y = 0.$$

Its general solution is

$$y(x) = Ae^{x} + Be^{-x} + Ce^{-2x}.$$

There are 3 components, 2 that decay and 1 that grows as $x \to \infty$.

Suppose we wish to solve this problem on $[0, \infty)$ with BCs

$$y(0) = 1,\ y'(0) = 1,\ \text{and } y(\infty) = 0.$$

The third BC implies $A = 0$.

The other 2 BCs lead to $B = 3$ and $C = -2$.

However consider the BCs

$$y(0) = 1,\ y(\infty) = 0,\ \text{and } y'(\infty) = 0.$$

The second BC implies $A = 0$.

But the last BC places no constraint on the solution, and the first one only gives us $C = 1 - B$.

Thus this BVP has infinitely many solutions!

If a BVP is not well-posed with BCs at infinity, it is natural to expect difficulties when trying to impose them at a finite point $b \gg 1$.

i.e., suppose we try to impose

$$y(0) = 1,\ y(b) = 0,\ \text{and } y'(b) = 0.$$

For large values of b (even as "large" as $b = 20$), the linear system that determines A, B, and C will be ill-conditioned.

The essence of the matter is that the two solution parts that decay exponentially cannot be distinguished from each other numerically by their values at $x = b$.

We can gain even more insight by generalizing this to a system of linear ODEs with constant coefficients

$$\mathbf{y}' = \mathbf{J}\mathbf{y} + \mathbf{q}(x).$$

For simplicity suppose $\mathbf{J}$ is non-singular.

Then the function $\mathbf{p}(x) := -\mathbf{J}^{-1}\mathbf{q}(x)$ is a particular solution of the ODEs.

Further assume that $\mathbf{J}$ has a complete set of eigenvectors $\{\mathbf{v}_j\}$ and corresponding eigenvalues $\{\lambda_j\}$.

This implies there are constants α_j such that

$$\mathbf{y}(a) - \mathbf{p}(a) = \sum_{j=1}^{m} \alpha_j \mathbf{v}_j,$$

and the general solution to the ODEs is

$$\mathbf{y}(x) = \mathbf{p}(x) + \sum_{j=1}^{m} \alpha_j e^{\lambda_j (x-a)} \mathbf{v}_j.$$

So $\mathbf{y}(x)$ is bounded on $[a, \infty)$ only when the BCs at $x = a$ imply that $\alpha_k = 0$ for all k such that $\mathrm{Re}(\lambda_k) > 0$.

Accordingly, the BVP is well-conditioned for $b \gg a$ only when the BCs at $x = a$ exclude the terms that grow exponentially fast as x increases.

Similarly the expansion

$$\mathbf{y}(b) - \mathbf{p}(b) = \sum_{j=1}^{m} \beta_j \mathbf{v}_j,$$

tells us that the BCs at $x = b$ must imply that $\beta_k = 0$ for any k such that $\mathrm{Re}(\lambda_k) < 0$ for the BVP to be well-conditioned.

Roughly speaking, components that decay rapidly from left to right must be determined by the BCs at the left end point, and vice versa for components that increase from left to right (and hence decay from right to left).

The general truth is that the BCs restrict the kinds of behaviours that a solution can have.

Travelling wave solutions of Fisher's equation

A wave travelling with speed c has the form $u(x,t) = U(z)$, where $z = x - ct$.

A travelling wave solution $U(z)$ of Fisher's equation[3] satisfies the ODE

$$U'' + cU' + U(1-U) = 0.$$

It is easy to see there are 2 steady states $U(z) \equiv 1, 0$.

Typical BCs that preclude steady-state solutions are

$$U(-\infty) = 1, \quad U(\infty) = 0.$$

For a given c one might be tempted to conclude we have a properly defined BVP.

[3]This equation was originally derived for the simulation of propagation of a gene in a population.

However, it turns out there are an infinite number of solutions: if $U(z)$ is a solution, then so is $U(z+\gamma)$ for any constant γ.

So if you replace $\pm\infty$ by some large numbers $\pm Z$ and use your favourite BVP solver, you are not likely to succeed because you have not specified enough information to specify which of the infinite number of solutions you want.

This problem has two critical points $(0,0)$ and $(1,0)$ in the phase plane (U,U').

A linear stability analysis shows that if $c \geq 2$, $(0,0)$ is a stable node and $(1,0)$ is a saddle point.

Analysis shows that if $c \geq 2$ then near the point $(0,0)$ a solution to the BVP exists with

$$U'(z) \sim \beta U(z),$$

where $\beta = (-c+\sqrt{c^2-4})/2$, and near the point $(1,0)$

a solution exists with

$$(U(z)-1)' \sim \alpha(U(z)-1),$$

where $\alpha = (-c+\sqrt{c^2+4})/2$.

Linear stability analysis amounts to approximating the nonlinear ODEs by linear, constant-coefficient ones.

But we know solutions to such problems are a particular solution plus a linear combination of exponentials.

So let's look for a solution of this form approaching $(1,0)$ as $z \to -\infty$: let

$$U(z) \sim 1 + pe^{\alpha z},$$

and hence

$$U'(z) \sim \alpha p e^{\alpha z}.$$

To approach $(1,0)$ as $z \to -\infty$ we require Re$(\alpha) > 0$.

Substituting this into the ODE, we find that

$$\alpha^2 p e^{\alpha z} + c\alpha p e^{\alpha z} \sim pe^{\alpha z} + p^2 e^{2\alpha z};$$

hence

$$\alpha^2 + c\alpha \sim 1 + pe^{\alpha z} \sim 1,$$

as $z \to -\infty$.

Thus $\alpha = (-c \pm \sqrt{c^2+4})/2$,[4] and only $\alpha = (-c + \sqrt{c^2+4})/2$ has $\mathrm{Re}(\alpha) > 0$.

To formulate the correct BC without introducing an extra unknown for p, we can require

$$\frac{U'(z)}{U(z)-1} \to \alpha,$$

as $z \to -\infty$; i.e., we choose a large number Z and impose the BC

$$\frac{U'(-Z)}{U(-Z)-1} - \alpha = 0.$$

For fun, we proceed a little differently in working out the BC for $z \to \infty$.

[4]The fact that the two values have opposite signs reflects the fact that $(1,0)$ is a saddle point.

Because $U(z) \to 0$ as $z \to \infty$, $U(z)(1-U(z)) \approx U(z)$, and so we can look at

$$U'' + cU' + U = 0,$$

having solution
$$U(z) \sim qe^{\beta z},$$
where β satisfies $\beta^2 + c\beta + 1 = 0$.

More analysis indicates we must use $\beta = (-c + \sqrt{c^2-4})/2$; essentially we work with the more slowly decaying part of the solution — the faster decaying one is then guaranteed to have vanished by the endpoint we choose.

Recall that if $U(z)$ is a solution, then so is $U(z+\gamma)$; this allows us to choose $q = 1$ and impose the BC

$$\frac{U(Z)}{e^{\beta Z}} - 1 = 0.$$

Other kinds of asymptotic behaviour

Not all BCs posed on infinite intervals have solutions that decay exponentially.

Consider

$$y' = \lambda x y, \quad y(0) = 1,\ y \sim e^{-x^2} \text{ as } x \to \infty.$$

Note that λ is an unknown.

It is easy to solve this BVP analytically to obtain

$$\lambda = -2, \qquad y(x) = e^{-x^2}.$$

Numerically we would impose a BC for some large b.

Because this $y(x)$ decays faster than exponentially, we may choose a b that is too large.

This can cause the BVP code to fail because it cannot distinguish between the various solutions at $x = b$.

Algebraic decay

At the other extreme are the BVPs with solutions that decay algebraically.

Consider

$$y' = -\lambda y^2, \qquad y(0) = 1,\ y(\infty) = 0,$$

where again λ is unknown.

The analytical solution is easily found to be

$$y(x) = \frac{1}{\lambda x + 1}.$$

This solution satisfies the BC at ∞ for any $\lambda > 0$; i.e., this BVP has infinitely many solutions.

We may need to provide more information about how the solution to a singular problem has to behave in order to have a well-posed problem.

e.g., if we require $y(x) \sim 1/x$ as $x \to \infty$, then there is a unique solution with $\lambda = 1$.

With algebraic decay, b must generally be taken to be quite large in order for it to represent the solution at ∞ to a reasonable accuracy.

e.g., suppose we impose $y(b) = 1/b$; then it is easily found that

$$\lambda_b = 1 - \frac{1}{b}, \quad y_b(x) = \frac{1}{x + 1 - x/b}.$$

Hence we would need quite a large b even for modest accuracy!

e.g., $b = 10^4$ for relative accuracy 10^{-4}.

3.4 Numerical Methods for BVPs

The theoretical approach to solving BVPs from Section 3.2 is based on the solution to IVPs and nonlinear algebraic equations.

Because we have good software for each of these problems, it is a natural idea to combine them to solve BVPs in what is known as a *shooting method*.

However, the most popular BVP solvers are *not* shooting codes.

The basic difficulty is that a well-posed BVP can lead to ill-posed shooting IVPs!

Consider

$$y'' - 100y = 0, \qquad y(0) = 1,\ y(10) = y_b.$$

Shooting (from left to right) involves solving an IVP with ICs

$$y(0) = 1, \quad y'(0) = s.$$

The exact solution to the IVP is

$$y(x;s) = \cosh(10x) + 0.1s\sinh(10x).$$

We note that

$$\max_{x\in[0,10]} \frac{\partial y}{\partial s} = 0.1\sinh(10x) \approx 1.3\times 10^{42}.$$

This says that the solution is *extremely sensitive* to the choice of s.

The value of s that solves the BVP is

$$s = \frac{10(y_b - \cosh(100))}{\sinh(100)}.$$

Using this value in $y(x;s)$, we then find that

$$\left|\frac{\partial y}{\partial y_b}\right| = \left|\frac{\sinh(10x)}{\sinh 100}\right| \le 1;$$

i.e., the solution is *not* sensitive to the boundary value.

Shooting can be effective if the resulting IVPs are not too unstable; if they are, codes may blow up before reaching $x = b$.

More often, however, the solver will reach $x = b$, but $y(b; s)$ will be inaccurate because we are solving an unstable IVP.

This difficulty feeds back into the nonlinear solver, which then cannot find an accurate value of s that solves the BVP.

Through proper preparation of the problem, shooting can solve more problems than perhaps is appreciated.

The approach we have been describing is more precisely called *simple shooting* because we only take one shot and hope the integration makes it all the way to $x = b$.

One way to stabilize simple shooting (and potentially improve its efficiency) is to split $[a, b]$ into several parts and perform simple shooting only over each part.

The idea is that if $\mathbf{f}(t, \mathbf{y})$ is Lipschitz, then there is a bound for the stability of the IVP involving $e^{L(b-a)}$; so reducing the effective length $(b-a)$ leads to exponential improvement in stability!

This method is thus known as *multiple shooting*.

Moreover the IVPs on each subinterval can be integrated *simultaneously*; hence multiple shooting is also called *parallel shooting*.

The solutions on the various subintervals must then be pieced together to form a continuous approximation over the entire interval.

Suppose we have a BVP with m first-order ODEs and $N-1$ breakpoints; i.e., $[a, b]$ has been divided into N subintervals.

Then we have $(N+1)m$ (possible unknowns) – m (determined from BCs) $= Nm$ unknowns in m (BCs) $+\ m(N-1)$ (continuity conditions) $= Nm$ nonlinear algebraic equations.[5]

[5]This accounting is slightly more complicated for non-separated BCs.

The nonlinear algebraic equations are of course solved by a variant of Newton's method.

At each iteration, a highly structured linear system must be solved.

It is critical to exploit this structure in practice!

If the break points are chosen carefully and the linear systems handled properly, multiple shooting can be quite effective.

In this view of multiple shooting, each IVP is solved by an IVP solver using variable step sizes.

Finite-difference methods can be viewed as multiple shooting methods where only one step is taken between break points; i.e., the solution to the BVP is approximated by points $\mathbf{y}_0$, $\mathbf{y}_1$, . . . , $\mathbf{y}_N$ on a mesh $a = x_0 < x_1, \ldots, < x_N = b$.

The BCs translate into the set of m (nonlinear) equations $\mathbf{g}(\mathbf{y}_0, \mathbf{y}_N) = \mathbf{0}$.

A simple but important numerical method is the trapezoidal rule

$$\mathbf{y}_{i+1} - \mathbf{y}_i = \frac{\Delta x_i}{2}[\mathbf{f}(x_i, \mathbf{y}_i) + \mathbf{f}(x_{i+1}, \mathbf{y}_{i+1})],$$

for $i = 0, 1, \ldots, N-1$.

These Nm equations plus the m BCs constitute a system of $(N+1)m$ nonlinear algebraic equations for the $(N+1)m$ unknowns $\mathbf{y}_i \approx \mathbf{y}(x_i)$, $i = 0, 1, \ldots, N$.

Notice that the $\mathbf{y}_i$ are defined implicitly even if the one-step method is formally explicit; e.g., forward Euler would lead to

$$\mathbf{y}_{i+1} - \mathbf{y}_i = \Delta x_i \mathbf{f}(x_i, \mathbf{y}_i),$$

but $\mathbf{y}_0$ and $\mathbf{y}_N$ are still coupled nonlinearly through the BCs $\mathbf{g}(\mathbf{y}_0, \mathbf{y}_N) = \mathbf{0}$.

This happens because of the fundamental difference between IVPs and BVPs.

In an IVP, all the information that specifies a solution is given at a single point, and the solution evolves from that point in a specific direction, discarding some or all of the past solution values.

A BVP has information given at (at least) 2 points, so there is no “direction” of integration, and there is no discarding of “past” solution values; i.e., all points of the solution must be solved for simultaneously.

So explicit one-step formulas do not have the same appeal as they did for IVPs.

Conversely, we need not be scared of implicit methods anymore.

In particular, the trapezoidal rule is symmetric and A-stable, making it a solid method for BVPs.

As with multiple shooting, there are $(N+1)m$ unknowns, but now N is (much) larger, so it is even more important to take advantage of the structure of the equations.

To understand this structure better, assume for now that the ODEs and BCs are linear;[6] i.e., let

$$\mathbf{y}' = \mathbf{J}(x)\mathbf{y} + \mathbf{q}(x),$$

$$\mathbf{B}_a\mathbf{y}(a) + \mathbf{B}_b\mathbf{y}(b) = \boldsymbol{\beta}.$$

The trapezoidal rule on $[x_i, x_{i+1}]$ is

$$\left[-\mathbf{I} - \frac{\Delta x_i}{2}\mathbf{J}(x_i)\right]\mathbf{y}_i + \left[\mathbf{I} - \frac{\Delta x_i}{2}\mathbf{J}(x_{i+1})\right]\mathbf{y}_{i+1} = \frac{\Delta x_i}{2}[\mathbf{q}(x_i) + \mathbf{q}(x_{i+1})].$$

These equations can be written in matrix form

$$\begin{pmatrix} \mathbf{S}_0 & \mathbf{R}_0 & & & \\ & \mathbf{S}_1 & \mathbf{R}_1 & & \\ & & \ddots & \ddots & \\ & & & \mathbf{S}_{N-1} & \mathbf{R}_{N-1} \\ \mathbf{B}_a & & & & \mathbf{B}_b \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{N-1} \\ \boldsymbol{\beta} \end{pmatrix},$$

where, for $i = 0, 1, \ldots, N-1$,

$$\mathbf{S}_i = -\frac{2}{\Delta x_i}\mathbf{I} - \mathbf{J}(x_i),\ \mathbf{R}_i = \frac{2}{\Delta x_i}\mathbf{I} - \mathbf{J}(x_{i+1}),\ \mathbf{v}_i = \mathbf{q}(x_i) + \mathbf{q}(x_{i+1}).$$

[6]In any event, these linear problems are the building blocks used in the iteration to solve nonlinear problems.

This is handled easily in MATLAB by treating this as a general sparse matrix.

Other solvers restrict their scope to separated BCs

$$\mathbf{B}_a\mathbf{y}(a) = \boldsymbol{\beta}_a, \quad \mathbf{B}_b\mathbf{y}(b) = \boldsymbol{\beta}_b,$$

so that they can obtain a banded system

$$\begin{pmatrix} \mathbf{B}_a & & & & \\ \mathbf{S}_0 & \mathbf{R}_0 & & & \\ & \mathbf{S}_1 & \mathbf{R}_1 & & \\ & & \ddots & \ddots & \\ & & & \mathbf{S}_{N-1} & \mathbf{R}_{N-1} \\ & & & & \mathbf{B}_b \end{pmatrix} \begin{pmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \\ \mathbf{y}_N \end{pmatrix} = \begin{pmatrix} \boldsymbol{\beta}_a \\ \mathbf{v}_0 \\ \vdots \\ \mathbf{v}_{N-1} \\ \boldsymbol{\beta}_b \end{pmatrix},$$

which can be easily stored and solved.

Higher-order RK methods for BVPs

Smooth problems are more efficiently solved (especially for low tolerances) by using methods of higher order than the second-order trapezoidal rule.

A general RK method with s stages forms s intermediate values $\mathbf{y}_{i,j} \approx \mathbf{y}(x_{i,j})$ at the points $x_{i,j} := x_i + c_j \Delta x_i$.

For IVPs our focus was on explicit methods that calculated these stages successively.

In general, the stages are determined simultaneously by solving a system of ms nonlinear algebraic equations

$$\mathbf{y}_{i,j} = \mathbf{y}_i + \Delta x_i \sum_{k=1}^{s} a_{j,k} \mathbf{f}(x_{i,k}, \mathbf{y}_{i,k}),$$

and then forming

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \Delta x_i \sum_{j=1}^{s} b_j \mathbf{f}(x_{i,j}, \mathbf{y}_{i,j}).$$

Implicit RK (IRK) formulas that reduce to Gaussian quadrature rules when solving the quadrature problem $\mathbf{y}' = \mathbf{f}(x)$ are popular because they have symmetry, excellent stability, and maximal order $2s$ for a method with s stages.

The first such rule is the midpoint rule

$$\mathbf{y}_{i+1} - \mathbf{y}_i = \Delta x_i \mathbf{f}(x_{i+1/2}, \mathbf{y}_{i+1/2}),$$

which has 1 stage and (maximal) order of 2.

Gaussian formulas do not evaluate the function at the endpoints of the subintervals; we pointed out that this can be useful when solving a problem with a singularity at an endpoint.

Finite-difference formulas only provide solutions at mesh points; hence we consider continuous extensions for these methods as we did in the IVP context.

A natural continuous extension for an s-stage IRK method is the polynomial $\mathbf{S}(x)$ that interpolates the mesh values $(\mathbf{S}(x_i) = \mathbf{y}_i)$ and the derivatives at the abscissae $(\mathbf{S}'(x_{i,j}) = \mathbf{f}(x_{i,j}, \mathbf{y}_{i,j}))$, $i = 1, 2, \ldots, s$.

For a certain class of IRK methods including those based on Gaussian quadrature formulas, it is not hard to show that $\mathbf{S}(x_{i+1}) = y_{i+1}$; i.e., the natural continuous extension is itself continuous!

This fact also implies that $\mathbf{S}(x)$ satisfies the ODE at the abscissae: $\mathbf{S}'(x_{i,j}) = \mathbf{f}(x_{i,j}, \mathbf{S}(x_{i,j}))$; i.e., $\mathbf{S}(x)$ *collocates* the ODE at the $x_{i,j}$ on each subinterval.

Generally the natural continuous extension is only $C[a,b]$, but the formulas of Lobatto type collocate the ODE at both ends of each subinterval.

Hence $\mathbf{S}(x) \in C^1[a,b]$ for these formulas.

The MATLAB function `bvp4c` uses a Simpson formula as its basic discretization; it is a three-point Lobatto method of order 4, so its natural continuous extension is $C^1[a,b]$ and also order 4.

The MATLAB function `bvp5c` uses a four-point Lobatto IIIA method of order 6, and its continuous extension is $C^1[a,b]$ and of order 5.

A classic approach to solving BVPs is to choose a form for $\mathbf{S}(x)$ that allows it to satisfy the BCs, then any remaining parameters are determined by collocating the ODE at sufficiently many points.

Some important classes of finite-difference methods can be viewed as collocation with a continuous piecewise-polynomial $\mathbf{S}(x)$, i.e., a spline.

That is why `bvp4c` and `bvp5c` are described as collocation codes.

`bvp4c` solves BVPs by computing a cubic spline on each subinterval $[x_i, x_{i+1}]$ of a mesh on $[a,b]$.

The spline is determined by requiring that $\mathbf{S}(x)$ be continuous on $[a, b]$, satisfy the BCs, and collocate the ODEs at the midpoint and endpoints of each subinterval

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

As mentioned this in fact makes $\mathbf{S}(x) \in C^1[a, b]$.

As usual these conditions result in a nonlinear system of algebraic equations to determine the spline coefficients.

When you do this, it turns out $\mathbf{S}(x)$ is the natural continuous extension of the Simpson formula.

So the method can be viewed as a collocation method or a finite-difference method with continuous extension.

`bvp5c` sacrifices one order of convergence by not evaluating the implicit formulas exactly, so we get a solution of order 5 instead of 6.

Enright and Muir implement a family of IRK formulas with continuous extensions in a Fortran code called `MIRKDC`.

One member of this family corresponds to the Simpson formula, but the continuous extension is higher degree (and higher accuracy) than the natural continuous extension used in `bvp4c`.

`MIRKDC` has been re-written to take advantage of enhancements in the Fortran programming language to make it more user-friendly. The software package is now called `BVP_SOLVER`.

Collocation is not restricted to first-order ODEs, and there are advantages to treating higher-order ODEs directly; this is done in the Fortran codes `COLSYS` and `COLNEW`.

Strictly speaking, these methods can only be viewed as equivalent to finite-difference methods when applied to first-order systems.

On initial guesses ...

Because a BVP can have any number of solutions, it is necessary to supply codes with an initial guess to pick out the solution of interest.

A good guess is often necessary to obtain convergence.

This guess not only involves giving a set of function values but also a mesh that captures the behaviour of the solution.

Codes will adapt the mesh after obtaining convergence on a given mesh so as to obtain a sufficiently accurate solution with as few mesh points as possible.

Mesh selection for BVPs is much more difficult than it is for IVPs: For an IVP, the most difficult step size to select is the first one; the steps that follow are adjusted one at a time and only slow variation is permitted.

The most difficult part of solving a BVP is providing suitable initial guesses *for the mesh and the solution* that converge to the desired solution.

On error control ...

A natural approach to error control is to estimate the truncation error and adjust the mesh accordingly.

When the truncation error on $[x_i, x_{i+1}]$ can be expressed in terms of the derivative of the solution, this derivative can be approximated by differentiating an interpolant using $\mathbf{y}_i$, $\mathbf{y}_{i+1}$, and perhaps solutions values from neighbouring intervals.

An important point for BVPs is that values on either side of $[x_i, x_{i+1}]$ can be used.

Another way of estimating error is to compare the result to that of a higher-order formula; this is done in the code `TWPBVP` of Cash and Wright.

Armed with estimates of truncation errors, meshes can be adjusted in much the same manner as they were with IVPs.

An obvious difference is that when solving BVPs, the entire mesh is (generally) changed, whereas only one step at a time is changed when solving IVPs.

Because BVPs are global in nature, a mesh refinement in one region will affect the errors in another, and not necessarily always for the better!

However, for methods where the truncation error on $[x_i, x_{i+1}]$ depends on a solution derivative, the effects of a local mesh change are local (to leading order), so improving the solution in one region does improve the solution overall.

A serious difficulty for BVP solvers is that the theory for estimating truncation errors and adjusting the mesh depend on the mesh being sufficiently fine.

So it is important that a solver adapt a mesh sensibly even when error estimates are crude or the mesh is poorly adapted to the solution.

To improve reliability, the codes `COLSYS` and `COLNEW` supplement the control of truncation error with an estimate using *extrapolation*.

The idea is as follows: Suppose we can compute a function $F(x;h)$ that involves a parameter h, and we want the limiting value $F(x;0)$.[7]

If we know that

$$e(x;h) := F(x;h) - F(x;0) \sim \phi(x)h^p \text{ as } h \to 0,$$

then we can use $F(x;h)$ and $F(x;h/2)$ to compute the error of the more accurate result.

Our assumption about how $e(x;h)$ depends on h tells us that

$$e\left(x;\frac{h}{2}\right) \sim \phi(x)\left(\frac{h}{2}\right)^p,$$

and hence

$$\begin{aligned} F(x;h) - F\left(x;\frac{h}{2}\right) &= e(x;h) - e\left(x;\frac{h}{2}\right) \\ &\sim \phi(x)\, h^p\, [1 - 2^{-p}]. \end{aligned}$$

[7] It is assumed that you cannot simply substitute $h = 0$ into $F(x;h)$.

Solving this equation for $\phi(x)h^p$ provides a computable estimate of the error of the more accurate result:

$$e\left(x;\frac{h}{2}\right) \sim \frac{1}{2^p-1}\left[F(x;h) - F\left(x;\frac{h}{2}\right)\right].$$

The codes `MIRKDC` and `bvp4c` take an approach to error control that is intended to deal more robustly with poor guesses to the mesh and the solution.

They produce approximate solutions $\mathbf{S}(x) \in C^1[a,b]$ and control the *residual*, i.e., the amount by which $\mathbf{S}(x)$ fails to satisfy the ODEs and the BCs:

$$\mathbf{r}(x) := \mathbf{S}'(x) - \mathbf{f}(x, \mathbf{S}(x)),\ \boldsymbol{\delta} := \mathbf{g}(\mathbf{S}(a), \mathbf{S}(b)) - \mathbf{0}.$$

In the framework of *backward error analysis*, $\mathbf{S}(x)$ is the exact solution to the BVP

$$\mathbf{Y}' = \mathbf{f}(x, \mathbf{Y}) + \mathbf{r}(x),\ \mathbf{g}(\mathbf{Y}(a)\mathbf{Y}(b)) - \boldsymbol{\delta} = \mathbf{0},$$

which is close to the original BVP if $\mathbf{r}(x)$, $\boldsymbol{\delta}$ are “small”.

If the BVP is well-conditioned, then small perturbations to the problem result in small perturbations to the solution, so a solution that is accurate by backward error analysis is also accurate in the usual sense.

This approach is attractive because the residual is well-defined no matter how crude the mesh.

For robustness, `bvp4c` measures the size of the residual on $[x_i, x_{i+1}]$ by

$$\int_{x_1}^{x_{i+1}} \|\mathbf{r}(x)\|^2 \, dx,$$

which is approximated using a five-point Lobatto quadrature formula, hence requiring two additional $\mathbf{f}(x, \mathbf{S}(x))$ evaluations per subinterval.

`bvp5c` takes advantage of a remarkable relation between the true error and the (scaled) residual in its error control:

$$\|\mathbf{e}(x)\|_{\infty,i} = \frac{8\sqrt{5}}{125}\Delta x_i \|\mathbf{r}(x)\|_{\infty,i} + \mathcal{O}(\Delta x^6),$$

where $\Delta x = \max_i \Delta x_i$.

In practice `bvp5c` requires

$$\Delta x_i \|\mathbf{r}(x)\|_{\infty,i} \leq \tau$$

for each subinterval i.

It turns out that

$$\|\mathbf{r}(x)\|_{\infty,i} \sim \Delta x_i^4 \, \|\mathbf{y}^{(5)}(x_{i+1/2})\|_\infty + \mathcal{O}(\Delta x^5);$$

so it is reasonable to expect that as long as $\mathbf{y}^{(5)}(x_{i+1/2})$ is not exceptionally small, control of $\|\mathbf{r}(x)\|_{\infty,i}$ will effectively control $\|\mathbf{e}(x)\|_{\infty,i}$.

Shooting as residual control

As a final remark, simple (and multiple) shooting can be viewed as controlling residuals.

At each step, the IVP solver controls the local error, which is equivalent to controlling the size of $\mathbf{r}(x)$ of an appropriate continuous extension of the numerical method being used.

The nonlinear algebraic equation solver then finds suitable ICs to make $\|\boldsymbol{\delta}\|$ small.

In the end, the result is a numerical solution that satisfies the ODEs and the BCs with a small residual.

3.5 Solving BVPs in MATLAB

We now examine a variety of nontrivial examples to illustrate how to solve BVPs in MATLAB using `bvp4c`.

We are not aware of any significant differences in the performances of `bvp4c` and `bvp5c`, and so we discuss the examples in terms of `bvp4c` for simplicity and uniformity of the presentation.

BVPs arise in such diverse forms that they often require some (and sometimes extensive) preparation for solution by standard software.

Example 3.5.1

Recall, Bratu's equation arises in a model of spontaneous combustion. The BVP is

$$y'' + \lambda e^y = 0, \quad y(0) = y(1) = 0.$$

Bratu showed that for $0 \leq \lambda < \lambda^* = 3.51383\ldots$, there are 2 solutions, and both are concave down.

These solutions grow closer as $\lambda \to \lambda^*$, and they coalesce to give a unique solution when $\lambda = \lambda^*$; there is no solution when $\lambda > \lambda^*$.

We begin by solving the BVP when $\lambda = 1$.

See `ch3ex1.m`

Note that the initial guess is $y(x) = x(1-x)$.

If you take $y(x) = 5x(1-x)$, the solver still converges to the same solution, the initial guess $y(x) = 20x(1-x)$ converges to the other solution, and $y(x) = 100x(1-x)$ does not converge at all.

Example 3.5.2

We consider a nonlinear eigenproblem of lubrication theory from Keller (1992):

$$\epsilon y' = \sin^2(x) - \lambda \frac{\sin^4(x)}{y}, \quad y\left(-\frac{\pi}{2}\right) = 1,\ y\left(\frac{\pi}{2}\right) = 1,$$

where ϵ is known, but λ is unknown.

`bvp4c` makes it easy to solve BVPs involving unknown parameters, but most solvers do not, so we go through this example twice.

Either way, however, an initial guess for unknown parameters must be provided.

They are supplied as the third argument of `bvpinit`.

Correspondingly, the vector of unknown parameters are passed as the third arguments to the functions for evaluating the ODEs and the residual in the BCs.

This must be done even if the unknown parameters do not explicitly appear in either the ODEs or the BCs.

The parameters are then stored in the `parameters` field of the solution structure.

See `ch3ex2.m`

Note 6. *The BCs do not involve λ.*

If the solver does not treat unknown parameters directly, we simply define $y_1(x) = y(x)$ and $y_2(x) = \lambda$ and add a trivial ODE indicating that $y_2(x)$ is constant:

$$\begin{aligned} y_1' &= \frac{1}{\epsilon}\left[\sin^2(x) - y_2 \frac{\sin^4(x)}{y_1}\right], \\ y_2' &= 0. \end{aligned}$$

Many solvers require analytical partial derivatives for the ODEs and BCs with respect to the solution.

If you introduce new variables to account for unknown parameters, the corresponding partial derivatives must be provided.

Example 3.5.3

The propagation of nerve impulses is described in Seydel (1988) by the ODEs

$$y_1' = 3\left(y_1 + y_2 - \frac{y_1^3}{3} - 1.3\right),$$
$$y_2' = -\frac{1}{3}(y_1 - 0.7 + 0.8y_2),$$

subject to periodic BCs

$$y_1(0) = y_1(T), \quad y_2(0) = y_2(T).$$

If we know T, these BCs are enough to completely specify the solution.

If we do not, then we need another BC.

Problems such as these are often converted IVPs; i.e., they are originally IVPs that need to be solved up to the unknown time T.

In such cases it is reasonable to assume we have (say) $y_1(0) = y_0$; for concreteness we assume $y_1(0) = 0$.

So we can impose the BCs

$$y_1(0) = 0, \quad y_1(T) = 0, \quad y_2(0) = y_2(T).$$

However there is still the complication that the interval of integration $[0, T]$ is unknown because T is unknown.

The only widely used solver that can handle this kind of problem directly is the multiple shooting code `D02SAF` from the NAG library.[8]

For all other codes (including `bvp4c` and `bvp5c`)), the problem must be transformed to one on a fixed interval.

This is easily accomplished by scaling the independent variable from t to $x = t/T$; hence $d/dt = T d/dx$, and the problem is posed on the known interval $[0, 1]$.

[8]It can do this because it treats the ends of each shooting subinterval as unknown.

Then the BVP becomes

$$\frac{dy_1}{dx} = 3T\left(y_1 + y_2 - \frac{y_1^3}{3} - 1.3\right),$$
$$\frac{dy_2}{dx} = -\frac{T}{3}(y_1 - 0.7 + 0.8y_2),$$

subject to

$$y_1(0) = 0, \quad y_1(1) = 0, \quad y_2(0) = y_2(1).$$

See `ch3ex3.m`

These BCs are non-separated, so if your solver can only handle separated BCs, more preparation is required.

A standard trick is to introduce a new unknown (the unknown periodic value) $y_3(t) = y_2(T)$.

This new unknown function is constant, so we also append a trivial ODE to the system.

We convert the non-separated BC $y_2(0) = y_2(1)$ into two separated BCs $y_2(0) = y_3(0)$ and $y_2(1) = y_3(1)$.

Example 3.5.6

Fluid flow in a long vertical channel with fluid injection in one side is described in Ascher, Mattheij, and Russell (1995) by the ODEs

$$\begin{aligned} f''' - R[(f')^2 - ff''] + RA = 0, \\ h'' + Rfh' + 1 = 0, \\ \theta'' + P_e f\theta' = 0, \end{aligned}$$

where R is the Reynolds number, $P_e = 0.7R$ is the Peclet number, and A is unknown.

This system has total order 7, but because of the unknown A, we require 8 BCs:

$$\begin{gathered} f(0) = f'(0) = 0,\ f(1) = 1,\ f'(1) = 0, \\ h(0) = h(1) = 0,\ \theta(0) = 0,\ \theta(1) = 1. \end{gathered}$$

It is quite common to study how a solution behaves as a function of parameters in the problem.

If a BVP is solved for a given set of parameters, we expect the solution, the mesh, and any unknown parameters to be good initial guesses for the solution to the BVP with slightly changed parameters.

`bvp4c` and `bvp5c` take advantage of this by accepting a solution structure as a guess structure.

This is not only an efficient way to obtain solutions to BVPs over a wide range of parameter values, it is a powerful way of solving difficult BVPs, in particular those for which an initial guess that converges to a solution is particularly difficult to obtain.

In this context, building up a solution to a BVP by solving a sequence of problems with different parameter values is known as *continuation*.

This BVP is hard for large R because there is a boundary layer at $x = 0$ that requires many mesh points to resolve.

It is easy to find a solution for $R = 100$ using even a constant initial guess; to do the same thing when $R = 10,000$ is very difficult!

It turns out we can use the solution for $R = 100$ as an initial guess to obtain the solution for $R = 1,000$, which in turn can be used as an initial guess to obtain the solution for $R = 10,000$.

See `ch3ex6.m`

Tough problems are often solved in this manner, requiring a large number of BVP solves.

It may then be worthwhile to reduce the run-time of each one.

This program includes both vectorization and analytical partial derivatives.

The MATLAB Symbolic Toolbox has a function called `jacobian` that can help you to generate the matrices of partial derivatives.

We might use the code

```
syms y y1 y2 y3 y4 y5 y6 y7 R A P F
y = [ y1; y2; y3; y4; y5; y6; y7 ];
P = 0.7*R;
F = [ y2; y3; R*(y2^2 - y1*y3 - A); y5;
      -R*y1*y5 - 1; y7; -P*y1*y7 ];
dFdy = jacobian(F,y),
dFdA = jacobian(F,A)
```

and edit the output to define the subfunction `Jac`.

The Jacobian of the BCs is handled similarly; the (i,j) element of `dBCdya` is $\frac{\partial g(ya,yb)}{\partial ya}$.

This is an 8×7 matrix because we have 8 BCs and `y` has 7 components.

With the default options, the authors found `ch3ex6.m` runs in 13.35s.

It is straightforward to vectorize the code by changing scalar quantities like `y(1)` into arrays like `y(1,:)` and changing scalar operations like `*` and `^` to array operations `.*` and `.^`.

Vectorization reduces the run time to 8.02s.

Just using analytical partial derivatives but no vectorization, the code runs in 6.64s.

Using analytical partial derivatives and vectorization results in a run time of 4.01s.

If we continue to $R = 1,000,000$, `bvp4c` reports

```
Warning: Unable to meet the tolerance
without using more than 142 mesh points.
```

The default limit on the number of mesh points is `floor(1000/m)`, where `m` is the number of ODEs.

This is quite arbitrary, so if you receive this warning you can use the option `Nmax` to increase it.

In this case if we set `Nmax=500`, we can solve the BVP.

But this is a rather difficult problem; `bvp4c` required 437 mesh points to obtain convergence with the default tolerances; even with vectorization and analytical partial derivatives, the computation took 22.69s, which is more than 5 times longer than before.

Example 3.5.7

Continuation is an extremely important tool in the practical solution of BVPs.

Suppose you are having trouble solving the BVP

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}), \quad \mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0}$$

because you cannot find an initial guess that is good enough for the solver to converge.

Suppose you can simplify your BVP to a form

$$\mathbf{y}' = \mathbf{F}(x, \mathbf{y}), \quad \mathbf{G}(\mathbf{y}(a), \mathbf{y}(b)) = \mathbf{0}$$

that can be solved easily (perhaps even analytically).

e.g., $\mathbf{F}(x, \mathbf{y})$ and $\mathbf{G}(\mathbf{y}(a), \mathbf{y}(b))$ may be linear; such problems are easy to solve because no iteration to solve nonlinear equations is required.

The idea is to "continue" from the solution of the easy problem to the solution of the problem you really want to solve.

A simple way to do this is to *embed* the problem in a family of problems by introducing an artificial parameter μ and solve the family of BVPs

$$\begin{aligned} \mathbf{y}' &= \mu \mathbf{f}(x, \mathbf{y}) + (1-\mu)\mathbf{F}(x, \mathbf{y}), \\ \mathbf{0} &= \mu \mathbf{g}(\mathbf{y}(a), \mathbf{y}(b)) + (1-\mu)\mathbf{G}(\mathbf{y}(a), \mathbf{y}(b)), \end{aligned}$$

for a sequence of μ values going from 0 to 1.

This simple approach has some utility, but it is not very powerful:

1. Finding a pair $\mathbf{F}, \mathbf{G}$ that captures the behaviour of the original BVP is difficult for hard problems.

2. It is possible that the BVP have no solution for some values of μ.

3. The exact choice of the sequence of μ values can make the difference between success and failure. Theory and software to determine a suitable sequence of μ values exist, but it seems there are still many open questions.

As an example, we consider the problem

$$\begin{aligned}
y_1' &= y_2,\\
y_2' &= y_3,\\
y_3' &= -\left(\frac{3-n}{2}\right) y_1 y_3 - n y_2^2 + 1 - y_4^2 + s y_2,\\
y_4' &= y_5,\\
y_5' &= -\left(\frac{3-n}{2}\right) y_1 y_3 - (n-1) y_2 y_4 + s(y_4 - 1),
\end{aligned}$$

where $n = -0.1$, $s = 0.2$, and the ODEs are to be solved on $[0, b]$ with $b = 11.3$ and subject to

$$y_1(0) = 0,\ y_2(0) = 0,\ y_4(0) = 0,\ y_2(b) = 0,\ y_4(b) = 1.$$

We split the ODEs into the linear and nonlinear parts and introduce a parameter δ to control the influence of the nonlinear parts:

$$\mathbf{y}' = \begin{pmatrix} y_2 \\ y_3 \\ 1 + sy_2 \\ y_5 \\ s(y_4 - 1) \end{pmatrix} + \delta \begin{pmatrix} 0 \\ 0 \\ -\left(\frac{3-n}{2}\right) y_1 y_3 - n y_2^2 - y_4^2 \\ 0 \\ -\left(\frac{3-n}{2}\right) y_1 y_3 - (n-1) y_2 y_4 \end{pmatrix}$$

i.e., $\delta = 0$ is a linear problem, $\delta = 1$ is the problem we want to solve, and $0 < \delta < 1$ is an intermediate nonlinear problem with nonlinearity scaled by δ.

For $\delta = 0$, the BVP is easy to solve.

We then use the solution for this δ as the initial guess for the BVP with a larger value of δ.

We continue in this manner until we reach $\delta = 1$.

It turns out the problem is easy enough that the sequence $\delta = 0, 0.1, 0.5, 1$ does the trick.

See `ch3ex7.m`

Example 3.5.8

`bvp4c` and `bvp5c` can handle multi-point BCs directly; `COLSYS` and `COLNEW` handle separated multi-point BCs.

However, many solvers do not, so we illustrate how to prepare multi-point BVPs for two-point BVP codes.

A physiological fluid flow problem from Lin and Segel (1988) can be formulated (after much preparation!) as the BVP

$$v' = \frac{C-1}{n},$$
$$C' = \frac{vC - \min(x,1)}{\eta},$$

where n and η are known (dimensionless) parameters, the ODEs are to be solved on $x \in [0, \lambda]$ for $\lambda > 1$, and subject to the BCs

$$v(0) = 0, \quad C(\lambda) = 1.$$

The quantity of most interest is the (dimensionless) *emergent osmolarity* defined by

$$O_s = \frac{1}{v(\lambda)}.$$

Lin and Segel use perturbation methods to conclude that for small n

$$O_s \sim \frac{1}{1-K},$$

where

$$K = \frac{\lambda \sinh(\kappa/\lambda)}{\kappa \cosh(\kappa)},$$

and κ is a parameter such that

$$\eta = \frac{\lambda^2}{n\kappa^2}.$$

It is important to note that the $\min(x, 1)$ term in the ODE for C' is not smooth at $x = 1$.

Numerical methods have less than their classical order of convergence when the ODEs (and hence their solutions) are not smooth.

Indeed Lin and Segel describe this BVP as two problems, one on $[0,1]$ and the other on $[1,\lambda]$, with the solutions connected by the requirement that $v(x)$ and $C(x)$ be continuous at $x=1$.

If you don't know perturbation theory, you can still solve the problem by recognizing it as a three-point BVP: we have BCs at $x=0,1,\lambda$.

First we introduce unknowns $y_1(x)=v(x)$ and $y_2(x)=C(x)$ for the interval $x\in[0,1]$; this leads to the ODEs

$$\begin{aligned}\frac{dy_1}{dx}&=\frac{y_2-1}{n},\\ \frac{dy_2}{dx}&=\frac{y_1y_2-x}{\eta},\end{aligned}$$

and the BC $y_1(0)=0$.

Then we introduce unknowns $y_3(x) = v(x)$ and $y_4(x) = C(x)$ for the interval $x \in [1, \lambda]$; this leads to the ODEs

$$\begin{aligned} \frac{dy_3}{dx} &= \frac{y_4 - 1}{n}, \\ \frac{dy_4}{dx} &= \frac{y_3 y_4 - 1}{\eta}, \end{aligned}$$

and the BC $y_4(\lambda) = 1$.

With these new variables, the continuity conditions at the internal boundary $x = 1$ translate to $y_1(1) = y_3(1)$ and $y_2(1) = y_4(1)$.

The only problem left is how can we solve all 4 ODEs simultaneously, i.e., on the same interval.

It is easy to scale and shift the interval $[1, \lambda]$ to map it onto $[0, 1]$ by defining a new independent variable

$$\xi = \frac{x - 1}{\lambda - 1}.$$

Noting that $\frac{d}{dx} = (\lambda - 1)\frac{d}{d\xi}$, the ODEs on $[1, \lambda]$ become

$$\frac{dy_3}{d\xi} = \frac{(\lambda - 1)(y_4 - 1)}{n},$$
$$\frac{dy_4}{d\xi} = \frac{(\lambda - 1)(y_3 y_4 - 1)}{\eta},$$

the original BC $y_4(x = \lambda) = 1$ becomes $y_4(\xi = 1) = 1$, the continuity condition $y_1(x = 1) = y_3(x = 1)$ becomes $y_1(x = 1) = y_3(\xi = 0)$, and the continuity condition $y_2(x = 1) = y_4(x = 1)$ becomes $y_2(x = 1) = y_4(\xi = 0)$.

See `ch3ex8.m`

The problem is solved for $n = 0.05$, $\lambda = 2$, and $\kappa = 2, 3, 4, 5$, with continuation in κ.

This problem can also be solved directly with `bvp4c` and `bvp5c` as a multi-point BVP with much less mathematical preparation but a little more accounting.

The essential difference is to define $[0, 1]$ and $[1, \lambda]$ as subdomains and specify the BCs and left and right BCs on these subdomains.

The ODE should also be defined in such a way that the appropriate function is evaluated based on the domain.

See `threebvp.m`