Math 563 Lecture Notes
Numerical methods for boundary value problems

Jeffrey Wong

April 12, 2020

Related reading: Ascher and Petzold, Chapter 6 (a good discussion of stability) and Chapter 7 (which includes details on multiple shooting and setting up Newton's method for these problems).

1 Boundary value problems (background)

An ODE **boundary value problem** consists of an ODE in some interval $[a, b]$ and a set of 'boundary conditions' involving the data at **both** endpoints.

After converting to a first order system, any BVP can be written as a system of m-equations for a solution $\mathbf{y}(x) : \mathbb{R} \to \mathbb{R}^m$ satisfying

$$\frac{d\mathbf{y}}{dx} = F(x, \mathbf{y}), \quad x \in [a, b]$$

with boundary conditions

$$B(\mathbf{y}(a), \mathbf{y}(b)) = \vec{0}.$$

As with ODE IVPs, this 'standard' form is typically used for numerical computation (e.g. Matlab's `bvp4c` requires the ODE written this way), although there are often simplifications that can be made given some structure.

For example, the **two-point** boundary value problem (i.e. a BVP with a second order ODE and two boundary conditions)

$$y'' = 4y - y', \ x \in [a, b], \quad y(a) = c, \quad y'(b) = d$$

can be written in this form, where $\mathbf{w} = (y_1, y_2)$ with $w_1 = y$ and $w_2 = y'$ and

$$\begin{aligned} w_1' &= w_2 \\ w_2' &= 4w_1 - w_2 \end{aligned} \qquad B(\mathbf{y}(a), \mathbf{y}(b)) = \begin{bmatrix} w_1(a) - c \\ w_2(b) - d \end{bmatrix}$$

General boundary conditions can be difficult to deal with. The most common (and simplest) type are linear boundary conditions, where

$$B_a\mathbf{y}(a) + B_b\mathbf{y}(b) = \mathbf{c}, \qquad B_a, B_b = m \times m \text{ matrices.}$$

The boundary conditions are **separated** if they can be written as

$$B_a\mathbf{y}(a) = \mathbf{c}_a, \qquad B_b\mathbf{y}(b) = \mathbf{c}_b$$

e.g. for the example above (where $B_a = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right]$ and $B_b = \left[\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right]$).

For an IVP, the effect of the initial condition is carried forward by the ODE from the initial point. We can follow this data and take steps one at a time, which is natural and straightforward.

In contrast, a BVP is much harder to solve because the ODE carries information from the boundaries throughout the interval. We cannot start at one point and step forward, because that would ignore the other boundary data. Thus, more work is required.

1.1 Stability

The stability of BVPs is a substantial topic; a quick study will help to illustrate the issue. First, suppose we have a linear **IVP**

$$\mathbf{y}' = A\mathbf{y} + p(t), \quad \mathbf{y}(0) = \mathbf{y}_0$$

where A is an $m \times m$ matrix. If the ODE function is perturbed slightly,

$$\tilde{y}' = A\tilde{y} + \tilde{p}(t), \quad \mathbf{y}(0) = 0$$

then the difference $u = \tilde{y} - y$ evolves according to

$$u' = Au + \delta(t), \quad u(0) = 0, \qquad \delta := \tilde{p}(t) - p(t).$$

From ODE theory, we know that the eigenvalues of A determine stability in the usual way (stable if $\mathrm{Re}(\lambda) < 0$). A similar result holds for changes in the initial condition, and a linear stability analysis (as we did before) extends this to the non-linear case.

Now contrast with the BVP case. Consider a linear, separated BVP

$$\mathbf{y}' = A\mathbf{y} + p(t), \quad B_a\mathbf{y}(a) = \mathbf{c}_a, \ B_b\mathbf{y}(b) = \mathbf{c}_b$$

and a perturbed problem (assuming the BCs do not change for simplicity)

$$\tilde{\mathbf{y}}' = A\tilde{\mathbf{y}} + \tilde{p}(t), \quad B_a\tilde{\mathbf{y}}(a) = \mathbf{c}_a, \ B_b\tilde{\mathbf{y}}(b) = \mathbf{c}_b.$$

Then the difference $u = \tilde{y} - y$ solves

$$u' = Au + \delta(x), \quad B_a u(a) = B_b u(b) = 0.$$

That is, the (linear) stability of the BVP depends on the properties of a *boundary value problem*. The solution can be written in terms of a Green's function:

$$u(x) = \int_a^b (\tilde{p}(y) - p(y))G(x,y)\,dy$$

and thus stability really depends on the nature of the Green's function (which we will not discuss here!). In particular,

$$\|u\| \le \|G\|_\infty \int_a^b |\tilde{p} - p|\,dy, \qquad \|G\|_\infty := \max_{a\le x,y\le b} |G(x,y)|$$

which describes the sensitivity of the solution to changes in the ODE function.[1]

The point here is that the notion of stability from IVPs, e.g. that $y' = \lambda y$ is stable as t increases if $\mathrm{Re}(\lambda) < 0$, does not qutie apply here. The boundary conditions can both create new instability or prevent it.

As a trivial but useful example, consider, for $c > 0$,

$$\mathbf{y}' = A\mathbf{y}, \quad A = \begin{bmatrix} -c & 0 \\ 0 & c \end{bmatrix}, \qquad y_1(0) = 1, \;\; y_2(b) = 1$$

This is nothing more than two uncoupled IVPs:

$$y_1' = -cy_1, \quad y_1(0) = 1$$

$$y_2' = cy_2, \quad y_2(b) = 1.$$

The equation for y_1 is solved forward (from $t = 0$ up to $t = 1$), and is stable since $\mathrm{Re}(c) > 0$; the result for IVPs apply here since it does not care about the boundary condition on the other side.

The equation for y_2 is solved backward (down to $t = 0$); it is equivalent to the IVP

$$\frac{dy_2}{d\tau} = -cy_2, \quad y_2(\tau = 0) = 1$$

for $\tau = b - t$, and so it is also stable. It follows that the BVP must be stable as well (by any reasonable definition).

However, an initial value problem for the system,

$$\mathbf{y}' = A\mathbf{y}, \quad \mathbf{y}(0) = \mathbf{y}_0$$

[1]Note: If the BCs are also allowed to be perturbed, then we instead get

$$\|u\| \le (\max\{\|\Phi\|, \|G\|\})(\text{change in BCs} +_\infty \int_a^b |\tilde{p} - p|\,dy)$$

where Φ is a fundamental matrix for $\mathbf{y}' = Ay$.

is clearly not stable since one component grows exponentially (and this is also true in the other direction). The 'instability' of the IVP does not apply to the BVP because the boundary stops any unstable growth.

The point: The key practical result here is that the stability of the boundary value problem and the associated initial value problem (in whatever direction) are **not necessarily the same**. Thus, using 'IVP' techniques to solve the boundary value problems can introduce new (artificial) instability [we will see this shortly].

The second point is that the 'stiffness' condition for a BVP is different. The linear, constant coefficient BVP

$$y' = \lambda y, \ x \in [0, b] \quad ...\text{BCs}...$$

may be considered stiff if

$$b|\text{Re}(\lambda)| \gg 1.$$

The condition here generalizes as in the IVP case. This means that a solution $y = e^{\lambda x}$ may grow rapidly relative to the interval size (plug in $x = b$). Because there is no longer a 'forward' direction as in an IVP, stiffness does not require $\text{Re}(\lambda) < 0$.

Stiff BVPs can be challenging to solve because they often involve **boundary layers**, thin regions of rapid variation near the boundaries (or elsewhere).

1.2 Uniqueness

One other subtlety is that existence/uniqueness is more complicated for BVPs.

Recall that for an IVP

$$\mathbf{y} = F(t, \mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0$$

there is a unique (local) solution under mild assumptions - namely, that F is continuous and Lipschitz in $\mathbf{y}$. In contrast, a BVP may have:

i) A unique solution

ii) multiple solutions

iii no solution at all

Obviously, a problem with no solution cannot be solved numerically. Case (iii) also has consequences for solvable problems - it means that even if a problem has a solution, nearby problems (which we may encounter in the approximation process) might not.

From a numerical perspective, (ii) is not a serious problem, since numerical methods will naturally select one solution in the computation. We just need to be careful to find the right solution - for instance, by using an initial guess close to the desired solution.

One important example is an **eigenvalue problem** such as

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

which has solutions $y(x) = \sin(kx)$ for $\lambda = k$ an integer and no solutions otherwise. Note that the eigenvalue problem can be regarded as a system of ODEs; set $w_1 = y$ and $w_2 = y'$ and add a third trivial variable λ:

$$w_1' = w_2, \quad w_2' = -\lambda w_1, \quad \lambda' = 0.$$

This is a first order system of three ODEs for (w_1, w_2, λ). The trick of replacing a parameter with a trivial ODE is a 'reformulation trick', which we can use to solve for unknown constants.[2]

2 Boundary value problems (shooting, part I)

To start, we consider a typical **two-point** boundary value problem

$$y'' = f(x, y, y'), \ x \in [a, b], \quad y(a) = c, \quad y(b) = d$$

for a function $y(x)$. Unlike an initial value problem, there are conditions involving y at **both endpoints of the interval**, so we cannot just start at $x = a$ and integrate up to $x = b$.

One approach (to be considered later) is to approximate $y(x)$ by some finite difference or other approximation scheme, then arrive at a system for the discrete values $y(x_i)$ that can be solved all at once. The downside is that one needs to typically solve large linear systems that require some effort.

However, there is a cheaper way. Note that $y(a)$ is specified. We can use an IVP solver to compute a solution $y(x; s)$ given initial conditions

$$y(a; s) = c, \quad y'(a; s) = s$$

where s is a 'guess' at the correct value of $y'(a)$. This solution will hit $x = b$ at some value that is probably not equal to the correct value, $y(b) = d$. However, we can then adjust the value of s, refining the guess until is is (nearly equal to) the right value.

This method is called **shooting** - the analogy here is to shooting a ball [insert relevant sport here] at a goal, determining the unknown correct velocity by throwing it too fast/too slow until it hits the goal exactly.

[2] In particular, it means that if you have a routine to solve a BVP in standard form, you can also solve BVPs with unknown parameters with the same code by adding trivial equations.

2.1 The details

Here are the details for using shooting to solve the two-point BVP

$$y'' = f(x, y, y'),\ x \in [a, b], \quad y(a) = c, \quad y(b) = d. \tag{BVP}$$

1) Setup the IVP: First, we set up the shooting initial value problem

$$y'' = f(x, y, y'), \quad y(a) = c,\ y'(a) = s. \tag{IVP$_s$}$$

The solution to (BVP) is a solution to (IVP$_s$) for a certain value s^* of s that we must find. Let us define

$$y(x; s) = \text{ solution to (IVP}_s\text{) for that value of } s.$$

2) Define the goal: Next, we define the 'goal function' that tells us what the correct value s^* must satisfy in the form $G(s) = 0$. Here, the goal function is

$$g(s) = y(x; s)\Big|_{x=b} - d.$$

We seek s^* such that $g(s^*) = 0$. Thus, the problem has been reduced to a non-linear equation that can be solved via the method of our choice. Note that computing $g(s)$ involves solving (IVP$_s$), so each evaluation of g is fairly expensive. Newton's method is then a desirable method due to its fast convergence.

2b) Setup variational problem for Newton: If using a 'derivative free' method like the secant method, this step can be skipped.[3] To use Newton's method, we also need the derivative of g. This requires knowing the derivative of y **with respect to** s. Let

$$z(x; s) = \frac{\partial y(x; s)}{\partial s}.$$

Then, by differentiating the DE and initial conditions, we get the variational ODE

$$z'' = \frac{\partial f}{\partial y} z + \frac{\partial f}{\partial y'} z', \quad z(a; s) = 0,\ z'(a; s) = 1 \tag{IVPZ$_s$}$$

which describes the change in the IVP solution with respect to its parameter s.

The derivative of g involves z at the endpoint $x = b$:

$$g'(s) = z(b; s).$$

Thus, computing $g'(s)$ requires solving the IVP (IVPZ$_s$). Putting this together, we now can compute $g(s)$ and $g'(s)$ so one just has to use Newton's method:

$$s_{n+1} = s_n - \frac{g(s_n)}{g'(s_n)}$$

[3] There are two reasons to do so: First, if the variational ODE is complicated or not smooth enough and the advantages of Newton's method is not needed, a method that needs to know only the ODE function is desirable. Second, the variational ODE might be ill-behaved, introducing more possibility of error/instability/etc.

which will hopefully converge to the correct value s^* if the initial value is good enough. Explicitly, the iteration reads

$$s_{n+1} = s_n - \frac{y(b; s_n) - d}{\frac{\partial y}{\partial s}(b; s_n)}.$$

To summarize, when using Newton's method, the steps are:

- Setup the shooting problem and define the goal function $g(s)$
- Use Newton's method to solve $g(s) = 0$ for the 'correct' value s^*. At each step,

  -solve the shooting IVP (IVP_s) to compute g

  -solve the variational ODE (IVPZ_s) **on the same grid** and obtain g'

  Take a step of Newton using g and g'
- Done; the BVP solution is the solution to the IVP with the computed value of s^*

Note that the variational ODE for $z(x; s)$ is solved on the same grid because it must know the IVP solution $y(x; s)$ at the grid points. You could also just put the two systems together and solve them at once.

Also note that shooting really only requires

- an initial value problem solver (e.g. RKF)
- a non linear system solver (e.g. Newton)

and any method of each type can be used. The appropriate choice will vary by problem.

2.2 Two-point BVP example

The two-point boundary value problem

$$y'' = 2y^3 - 6y - 2x^3, \qquad y(1) = 2, \;\; y(2) = 5/2 \tag{2.1}$$

has a solution $y(x)$ that happens to be given exactly by

$$y(x) = x + \frac{1}{x}, \quad x \in [1, 2].$$

Let $y(x; s)$ be the solution to the shooting IVP

$$y'' = 2y^3 - 6y - 2x^3, \qquad y(1) = 2, \;\; y'(1) = s. \tag{2.2}$$

The goal function is

$$g(s) = y(2; s) - 5/2 \tag{2.3}$$

so that the solution to the IVP with s such that $g(s) = 0$ is also a solution to the BVP (2.1).

For computation, set $w_1 = y, w_2 = y'$ and $z_1 = \partial y/\partial s$ and $z_2 = \partial y'/\partial s$. Then

$$\begin{aligned} w_1' &= w_2 \\ w_2' &= 2w_1^3 - 6w_1 - 2x^3 \\ z_1' &= z_2 \\ z_2' &= (6w_1^2 - 6)z_1 \end{aligned}$$

with initial condition

$$(w_1, w_2, z_1, z_2) = (0, s, 0, 1).$$

The goal function and its derivative are then

$$g(s) = w_1(2; s) - 5/2, \quad g'(s) = w_3(2; s).$$

Note that the first two equations can be solved first, and the variational ODE (the last two equations) can be solved after if the same grid is used.

The ODE has the feature that if $y'(1)$ is too large (e.g. $y'(1) > 1/2$) then the solution to the IVP (2.2) has a singularity in $[1, 2]$. Thus, it is necessary to choose a starting value $s < 1/2$, and must be close enough to 0 that it never enters this singular region. Or, some other, safer method could be used (e.g. bisection).

2.3 Perils

It is important to note that the stability of shooting **depends on the stability of the IVP**, even though (as discussed) the actual stability of the BVP may be different.

In particular, a stable BVP may have a shooting IVP that is very stiff or otherwise ill-behaved. That is, by using shooting we may introduce numerical issues hat are not inherent to the problem, which is a significant disadvantage.

Moreover, note that the IVPs must be solved for guesses s that are not correct, so we may need to solve nearby problems that are worse than the true solution - e.g. if the IVP is singular for certain values of s.

There are two cheap ways to 'fix' shooting if the IVP is not well behaved.

First, if the IVP blows up only in one direction, we can try to shoot from the other side. [The application is left as an exercise].

A stronger technique is **multiple shooting**. The idea is that if the IVP wants to blow up after some distance $< b$ for a BVP in the interval $[0, b]$, we can 'restart' the solve by guessing values inside the interval and then connect the pieces. For the two-point problem

$$y'' = f(x, y, y'), \quad y(a) = c, \quad y(b) = d$$

multiple shooting with M segments proceeds by picking points $x_1, x_2, \cdots, x_{M-1}$ in $(0, b)$ (with $x_0 = a$) and providing a guess at each point. We solve the IVPs

$$y'' = f(x, y, y'), \quad y'(x_j) = s_j$$

for a solution $y(x; \mathbf{c})$, dependent on the guess $\vec{s} = (s_0, \cdots s_{M-1})$ for the starting value for y' at each point. The condition on y is a continuity condition; the values $y(x_j; s)$ are of course available if we solve the IVPs in order ($j = 0, 1, \cdots$.)

The goal function is the same as before, $g(s) = y(b; s) - d$, but now depends on an M-dimensional vector. The non-linear system can be solved via the usual techniques (e.g. Newton's method), but takes some work.

For this reason, multiple shooting is most useful when shooting almost works and it is reasonably easy to set it up (requiring a small M, or some nice structure). Otherwise, other techniques (e.g. finite difference methods, to be discussed) should be used. Note that if the IVP is extremely poorly behaved, then M has to be quite large, which is undesirable as the size of the system $g(s) = 0$ to be solved puts a serious demand on the root-finder.

2.4 Continuation

A boundary value problem encountered in plasma physics[4] is **Troesch's problem**

$$y'' = \mu \sinh(\mu y), \quad y(0) = 0, \; y(1) = 1$$

where $\mu > 0$ is a parameter. The IVP for shooting and variational ODE for $z = \partial y / \partial s$ are

$$\begin{aligned} y'' &= \mu \sinh(\mu y), \quad y(0) = 0, \; y'(0) = s \\ z'' &= -\mu^2 \cosh(\mu y) z, \quad z(0) = 0, \; z'(0) = 1 \end{aligned}$$

and the goal function is

$$g(s) = y(1; s) - 1.$$

It can be shown that

$$y'(0) > 8e^{-\mu} \implies \text{the IVP solution has a singularity in } [0, 1].$$

Note that this implies that the correct value of s decays at least like $e^{-\mu}$ as μ increases, so the right value can become quite small (this means care is required in using relative vs. absolute error tolerances in Newton's method).

More importantly, when μ is not small (about $\mu > 5$), the ODE is sensitive to the value of $y'(0)$, and one has to pick a very good initial guess to get Newton's method to converge.

If shooting is used, this problem is a good candidate for continuation; the problem is easy to solve for $\mu = 0$, and one can increase μ from there.

[4] See https://doi.org/10.1016/0021-9991(73)90165-4.

3 Non-scalar problems

Consider a boundary value problem

$$\mathbf{y}' = F(x, \mathbf{y}),\ x \in [a, b], \quad B(\mathbf{y}(a), \mathbf{y}(b)) = 0$$

with a first order system of m equations. The shooting IVP is

$$\mathbf{y}' = F(x, \mathbf{y}),\ x \in [a, b], \quad \mathbf{y}(a) = \mathbf{s}$$

which has a solution $\mathbf{y}(x; \mathbf{s})$. Note that the guess is, at worst, a vector in $\mathbb{R}^m$. If some components at $x = a$ are known, we can reduce the dimension (e.g. if $y_1(a) = 0$).

The goal function is then

$$G(\mathbf{s}) = B(\mathbf{y}(a; \mathbf{s}), \mathbf{y}(b; \mathbf{s})), \quad G : \mathbb{R}^m \to \mathbb{R}^m$$

again noting that its dimension may be less if it can be simplified.

We could use a derivative-free method to solve $G(\mathbf{s}) = 0$ (and be done); the details of such methods will not be discussed here. To use Newton's method, we need the Jacobian of G. Define the matrix-valued function (the variation of the solution with respect to $\mathbf{s}$)

$$Z(x; \mathbf{s}) = \frac{\partial \mathbf{y}(x; \mathbf{s})}{\partial \mathbf{s}}.$$

That is, Z has columns $[z_1, \cdots, z_m]$ where

$$z_j = \frac{\partial \mathbf{y}(x; \mathbf{s})}{\partial s_j} = \text{variation of } \mathbf{y} \text{ with respect to } s_j.$$

By differentiating the IVP, we see that Z solves the variational ODE

$$Z' = J(x, \mathbf{y})\, Z, \quad Z(a; \mathbf{s}) = I \tag{3.1}$$

where $J(x, \mathbf{y}) = \partial F / \partial \mathbf{y}$ is the Jacobian of F with respect to y. Note that the ODE (3.1) is really a collection of m equations for

$$\mathbf{z}_j(x; \mathbf{s}) = \frac{\partial \mathbf{y}(a; \mathbf{s})}{\partial s_j}, \quad j = 1, \cdots, m$$

which is how they would be solve numerically. Because the ODE is linear and the Jacobian is shared by all m equations, all the z_j's can be computed efficiently at once.

With Z known, we can then calculate the Jacobian of the goal function needed for shooting.

3.1 Period finding (VDP)

The Van der Pol equation is (in system form)

$$x_1' = x_2, \quad x_2' = \mu(1 - x_1^2)x_2 - x_1$$

which describes a non-linear oscillator with negative damping when its amplitude is small.

For certain values of μ (when $\mu > 0$) this system has a limit cycle - a unique periodic solution. This can be formulated as a boundary value problem. Let the period be T, and suppose that $x_1(0) = 0$ (it is not hard to show that the limit cycle must cross the x_2-axis, so we can pick that crossing point as the $t = 0$ point as the system is autonomous).

The BVP to solve is

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= \mu(1 - x_1^2)x_2 - x_1 \end{aligned} \qquad \begin{aligned} x_1(0) &= 0 \\ x_1(0) &= x_1(T) \\ x_2(0) &= x_2(T) \end{aligned}$$

where T is also an unknown. We can convert to a nicer form using two tricks. First, scale to get a fixed domain; set $\tau = t/T$ as a new time variable. Second, regard T as an independent variable with $T' = 0$ to get a first order system for (x_1, x_2, T). The result is

$$\begin{aligned} \frac{dx_1}{d\tau} &= Tx_2 \\ \frac{dx_2}{d\tau} &= T(\mu(1 - x_1^2)x_2 - x_1) \\ \frac{dT}{d\tau} &= 0 \end{aligned} \qquad \begin{aligned} x_1(0) &= 0 \\ x_1(0) &= x_1(1) \\ x_2(0) &= x_2(1) \end{aligned}$$

which is a first-order system in a standard form (with three BCs as expected). The fact that $T' = 0$ means that the process can be simplified a bit.

To apply shooting to solve this BVP, we solve an IVP of the form

$$\frac{d\mathbf{x}}{d\tau} = TF(\mathbf{x}), \quad \mathbf{x}(0) = (0, s) \text{ with a guess } (s, T)$$

noting that the $T' = 0$ equation doesn't actually need to be included in the ODE as there is nothing to solve. The goal function is

$$G(s, T) = \begin{bmatrix} x_1(1; s) \\ x_2(1; s) - s \end{bmatrix}.$$

If Newton'as method is to be used, we need the appropriate variational ODEs. Let

$$\mathbf{z} = \frac{\partial x_1}{\partial s}, \quad \mathbf{w} = \frac{\partial x_2}{\partial T}.$$

Then, with $J(\mathbf{x})$ the Jacobian of $F(\mathbf{x})$,

$$\frac{d\mathbf{z}}{d\tau} = TJ(\mathbf{x})\mathbf{z}, \quad \frac{d\mathbf{w}}{d\tau} = F(\mathbf{x}) + TJ(\mathbf{x})\mathbf{w}$$

with initial conditions

$$z_1(0) = 0,\ z_2(0) = 1, \quad w_1(0) = w_2(0) = 0.$$

The Jacobian of G can then be computed using $\mathbf{z}$ and $\mathbf{w}$ at $t = 1$, e.g.

$$\frac{\partial G_2}{\partial s} = \frac{\partial}{\partial s}\left(x_2(1; s) - s\right) = z_2(1; s).$$

Note that one could instead think of the system as a three-equation system $\mathbf{x}' = H(\mathbf{x})$ for (x_1, x_2, x_3) with $x_3' = 0$ (as T) and with a shooting parameter $s = (s_1, s_2)$ and initial conditions

$$x_1(0) = 0,\ x_2(0) = s_1,\ x_3(0) = s_2.$$

Then the variational ODE for the matrix $Z = \partial\mathbf{x}/\partial\mathbf{s}$ is

$$Z' = \frac{\partial H}{\partial \mathbf{x}} Z.$$

The result is the same as the calculation above after simplifying (we have essentially solved the $x_3' = 0$ equation and split Z into its columns).

The shooting method has trouble when the ODE starts to become harder to solve; as μ increases, the initial guess must be very accurate (as well as the ODE solution). Continuation can help, but one may be better off using another method entirely.

4 Finite differences

Suppose, for example, we wish to solve the boundary value problem

$$\begin{aligned} y'' &= f(x, y, y'), \quad x \in [0, b] \\ &\text{(BCs at } x = 0 \text{ and } x = b) \end{aligned} \tag{4.1}$$

This problem can be solved by the following direct process:

1) **Discretize the domain:** Choose grid points $\{x_n\}$ for the approximation $\{u_n\}$

2) **Discretize the ODE:** Approximate derivatives with finite differences in the interior of the domain (where BCs do not apply).

3) **Boundary conditions:** Discretize the boundary conditions, and modify (2) as needed.

4) Solve the system of equations from (2) and (3) by the usual methods.

Unlike shooting, the entire solution is solved for at once, which avoids the potential instabilities in the auxiliary IVPs. The details depend, of course, on the choice of grid points (uniform or non-uniform) and the discretization (which formulas to use).

4.1 The basic approach

We solve the example problem (4.1) with 'Dirichlet' BCs

$$y(0) = c, \quad y(b) = d \tag{4.2}$$

using a uniform grid

$$x_n = hn,\ n = 0, 1, \cdots, N, \qquad h := b/N.$$

Since the values of y are known at $n = 0$ and $n = N$, our unknowns are $(y_1, \cdots, y_{N-1})$.

Using the standard centered difference formula for y'' and y' and replacing $y(x_n)$ with u_n (the approximation to be found) gives

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} = f(x_n, u_n, \frac{u_{n+1} - u_{n-1}}{2h}), \quad n = 1, \cdots, N-1, \tag{4.3}$$

and applying the boundary conditions gives (trivially)

$$u_0 = c, \quad u_N = d.$$

This is a system of $N+1$ equations for the $N+1$ values of $\{u_n\}$ on the grid, or alternatively a system of $N-1$ equations for the 'interior' values $(u_1, \cdots, u_{N-1})$. The latter is preferable for computation. The result is a system of equations for $\mathbf{u} = (u_1, \cdots, u_{N-1})$,

$$G(\mathbf{u}) = 0, \qquad G : \mathbb{R}^{N-1} \to \mathbb{R}^{N-1}.$$

The equations are obtained by taking (4.5) and 'plugging in' for the boundary values u_0, u_N:

$$\begin{aligned} G_1(\mathbf{u}) &= u_2 - 2u_1 + c - h^2 f(x_1, u_1, \frac{u_2 - c}{2h}), \\ G_n(\mathbf{u}) &= u_{n+1} - 2u_n + u_{n-1} - h^2 f(x_n, u_n, \frac{u_{n+1} - u_{n-1}}{2h}), \quad n = 2, \cdots, N-2, \\ G_{N-1}(\mathbf{u}) &= d - 2u_{N-1} + u_{N-2} - h^2 f(x_{N-1}, u_{N-1}, \frac{d - u_{N-2}}{2h}), \end{aligned}$$

This system can then be solved using Newton's method. Note that the Jacobian $J = \partial G/\partial \mathbf{u}$ is **tridiagonal**, i.e. has non-zero entries only on the center diagonal and one above/below; the linear system for Newton steps can then be solved efficiently (each linear solve requires $O(N)$ operations).

Note that if the BVP is **linear** then the system has the form $Au = \vec{b}$, so it can be solved as a linear system. For example, if $f = x^2$ (so $y'' = x^2$) then

$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} -c \\ 0 \\ \vdots \\ 0 \\ -d \end{bmatrix} + h^2 \begin{bmatrix} x_1^2 \\ x_2^2 \\ \vdots \\ x_{N-2}^2 \\ x_{N-1}^2 \end{bmatrix}$$

Note that the boundary conditions add to the right hand side - every term that does not depend on u gets moved to the RHS (b) and every term involving u is on the LHS (Au).

4.2 Example (linear)

The **Airy function** $\mathrm{Ai}(x)$ is a multiple of[5] the unique solution to

$$y'' = xy, \quad y(0) = 1, \quad \lim_{x \to \infty} y(x) = 0$$

Because the ODE has another solution $\mathrm{Bi}(x)$ that grows rapidly (see homework 6), using shooting can be difficult (it is sensitive to the choice of $y'(0)$, and solutions do not want to go to zero).

We solve the BVP instead, replacing infinity with a large value L. Letting

$$x_n = nh, \quad h = L/N,$$

we discretize as discussed above to get

$$u_{n+1} - 2u_n + u_{n-1} = h^2 x_n u_n, \quad n = 1, \cdots, N-1$$

[5] to be precise, the Airy function has $\mathrm{Ai}(0) = 1/(3^{2/3}\Gamma(2/3))$.

Then the system to solve (which is linear) is $Au = b$ where

$$A = \begin{bmatrix} -2-h^2x_1 & 1 & 0 & \cdots & 0 \\ 1 & -2-h^2x_2 & 1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2-h^2x_{N-2} & 1 \\ 0 & \cdots & 0 & 1 & -2-h^2x_{N-1} \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

The method has no trouble computing a solution. Note that because $x > 0$, the matrix A is **diagonally dominant** (the diagonal entry is larger in size than the absolute sum of other entries in the row). It follows that the matrix is invertible and that the system can be stably solved using LU decomposition without pivoting.

4.3 Neumann (no known values)

If a derivative is specified ('Neumann boundary conditions'), such as

$$y'(0) = c, \quad y'(b) = d \tag{4.4}$$

there is one subtlety. We set up the grid as before and discretize the ODE in the same way for interior points.

However, no values of y are known, so all $N+1$ values of $\{u_n\}$ (from $n = 0$ to $n = N$) are unknowns. We have, discretizing the ODE the same way,

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} = f(x_n, u_n, u'_n), \quad n = 1, \cdots, N-1 \tag{4.5}$$

where $u'_n = (u_{n+1} - u_{n-1})/2h$ and $u'_0 = c$ and $u'_N = d$.

The above formula not work for $n = 0$ or $n = N$ because u_{-1} and u_{N+1} do not exist - the stencil for the finite difference at point x_n (using x_{n-1}, x_n, x_{n+1}) would leave the domain.

Our last two equations come from the discretizing the boundary condition. It is not a good idea to use a forward difference

$$\frac{y_1 - y_0}{h} \approx c$$

(why not?), and while a second-order forward difference could be used, it is not the ideal solution. Instead, the typical approach is to use a **centered difference** to get

$$\frac{y_1 - y_{-1}}{2h} \approx c$$

where $y{-}1$ denotes $y(x_{-1}) = y(-h)$. We now have an equation to extrapolate the value of y at x_{-1}, which yields an equation at the fictitious point x_{-1}:

$$u_{-1} = u_1 - 2hc. \tag{4.6}$$

Similarly,

$$u_{N+1} = u_{N-1} + 2hc. \tag{4.7}$$

Now we can apply the discretized formula (4.5) for the ODE at x_0, since $x-1$ is now an available point, leading to the single formula

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} = f(x_n, u_n, u'_n), \quad n = 0, \cdots, N$$

where u_{-1} and u_{N+1} are given by (4.6), (4.7). We now have a system of $N+1$ equations for the $N+1$ unknowns that can be solved.

Essentially, the boundary conditions were used to pad the domain with 'ghost points' that are not part of the solution, but are needed because the discretization extends past the endpoints. Depending on the problem, the trick here can be more delicate, but we will omit a detailed discussion of the nuances here.

4.4 Midpoint discretization

Now consider the boundary value problem (in first-order form)

$$\mathbf{y}' = F(x, \mathbf{y}), \quad x \in [0, b]$$

with m equations (so $\mathbf{y} : \mathbb{R} \to \mathbb{R}^m$) with linear boundary conditions

$$B_0\mathbf{y}(0) + B_b\mathbf{y}(b) = \mathbf{c}.$$

Consider a grid $\{x_n\}$ with $h_n = x_{n+1} - x_n$ and $x_0 = 0, x_N = b$ and let $x_{n+1/2}$ denote the midpoint of $[x_n, x_{n+1}]$. We can use the **midpoint** formula to discretize the ODE - we use a centered difference at each midpoint $x_{n+1/2}$, which uses the values at x_n and x_{n+1}:

$$\frac{\mathbf{u}_{n+1} - \mathbf{u}_n}{h_n} = F(x_{n+1/2}, \frac{\mathbf{u}_{n+1} + \mathbf{u}_n}{2}), \quad n = 0, \cdots, N-1$$

noting that $\mathbf{y}$ at $n + 1/2$ has been approximated with an average.[6]

The boundary conditions provide one more set of equations, so there are mN equations from the ODE and m from the BCs. This matches the $m(N+1)$ unknowns, so we have a complete system that can be solved via the usual methods (e.g. Newton).

Note also that the resulting linear systems for Newton's method will have a nice structure: except for the boundary conditions, the matrix will be **banded** - only elements a certain distance from the main diagonal are non-zero.

If we let

$$J_{n+1/2} = \frac{\partial F}{\partial \mathbf{y}}(x_{n+1/2}, \mathbf{u}_{n+1/2})$$

[6]Note that there are other ways the discretization could be done; the version here is used as an example.

and

$$V_n = -I - \frac{1}{2}h_n J_{n+1/2}, \quad W_n = I - \frac{1}{2}h_n J_{n+1/2},$$

then the Jacobian for $G(\mathbf{u}) = 0$ with

$$G_n(\mathbf{u}) = \mathbf{u}_{n+1} - \mathbf{u}_n - F(x_{n+1/2}, \mathbf{u}_{n+1/2}), \quad n = 0, \cdots, N-1$$

and

$$G_N(\mathbf{u}) = B_0\mathbf{u}_0 + B_b\mathbf{u}_N - \mathbf{c}$$

is given in **block** form by

$$; \begin{bmatrix} V_0 & W_0 & 0 & \cdots & 0 \\ 0 & V_1 & W_1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & V_{N-1} & W_{N-1} \\ B_0\text{`} & \cdots & 0 & 0 & B_b \end{bmatrix}.$$

If the boundary conditions are separated, i.e. $\tilde{B}_0 y(0) = \tilde{B}_b y(b) = 0$, we can split the boundary equations apart and put the $x = 0$ ones at the top rows and the $x = b$ ones at the bottom, preserving the band structure:

$$; \begin{bmatrix} \tilde{B}_0 & 0 & \cdots & \cdots & 0 \\ V_0 & W_0 & 0 & \cdots & 0 \\ 0 & V_1 & W_1 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & V_{N-1} & W_{N-1} \\ 0 & \cdots & 0 & 0 & \tilde{B}_b \end{bmatrix}.$$

Because it is a 'standard' first order system, the method can be used on a variety of problems (including the second-order BVP of the previous section). The primary advantage is that the first order system allows us to discretize only first-order derivatives, which only take two points (x_n and x_{n+1}). This small stencil makes using a non-uniform grid simple, because the discretized ODE at each point $x_{n+1/2}$ depends only only one of the grid spacings ($x_{n+1} - x_n$).

4.5 Upwind example

The 'centered' difference is not always the natural way to discretize. As an illustrative example, consider

$$\mathbf{y}' = \begin{bmatrix} -\lambda & 0 \\ 0 & \lambda \end{bmatrix} \mathbf{y}, \quad y_1(0) = 1, \quad y_2(3) = 1$$

Observe that:

- The solution is the sum of two independent 'modes' ($(y_1(t), 0) + (0, y_2(t))$)
- Information propagates **only to the right** for y_1 (from the boundary at $x = 0$)

- Information propagates only to the left for y_2 (from $x = b$)

The centered difference for the ODE at x_n,

$$\mathbf{y}'_n \approx \frac{\mathbf{y}_{n+1} - \mathbf{y}_{n-1}}{2h}$$

depends on information in **both** directions (at indices $n-1$ and $n+1$). We might instead want to have a method that has the same 'propagation' behavior as the solution. This means that we should use backward differences for y_1 and forward differences for y_2, e.g.

$$(y_1)'_n \approx \frac{(y_1)_n - (y_1)_{n-1}}{h}, \quad (y_2)'_n \approx \frac{(y_2)_{n+1} - (y_2)_n}{h}.$$

As seen in the stability discussion, this system is really just two IVPs in opposite directions. If we use the upwind discretization, then we are using Backward Euler for both components, so the method will be nice and stable even when λ is large.

If another discretization is used and λ is large, stiffness may be an issue.

The catch is that this approach is hard to generalize, since for a general system

$$\mathbf{y}' = F(\mathbf{y})$$

it is difficult or impossible to separate into modes. This sort of discretization plays an important role in solving PDEs with propagating waves (e.g. the wave equation, shocks formed in gas flow, etc.), where there is a natural direction to the flow of information.

5 The linear algebra

We will not cover the numerical linear algebra here in detail[7]. The focus here is on highlighting what one needs to know to use the algorithms.

A **banded matrix** A is a matrix such that

$$a_{ij} = 0 \text{ for } -m_1 \leq j - i \leq m_2$$

where $0 < m_1, m_2$ and $m = m_1 + m_2 + 1$ is the 'bandwidth', which should be much smaller than the matrix dimensions. That is, A has non-zero diagonals only close to the center diagonal. A common special case that we have seen is a **tridiagonal matrix**, for which $m_1 = m - 2 = 1$ and the bandwidth is three:

$$; \begin{bmatrix} \beta_1 & \gamma_1 & 0 & \cdots & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \alpha_{N-1} & \beta_{N-1} & \gamma_{N-1} \\ 0 & \cdots & 0 & \alpha_N & \beta_N \end{bmatrix}.$$

An **LU decomposition** of a square matrix is a factorization

$$A = LU, \quad L = \text{lower triangular}, \quad U = \text{upper triangular}.$$

This factorization can be used to solve linear systems $Ax = b$ in two steps:

1) (Factor) Compute L and U such that $A = LU$

2) (Solve) Solve $Ly = b$,then solve $Ux = y$ (with forward/back substitution)

Once (1) is done, step (2) can be done for many RHS's b without re-factoring the matrix, so step (1) is often an initial 'processing' of the matrix to prepare it for solving linear systems.

In practice: When solving systems with LU decomposition, you should make use of a specialized solver. This means, typically, calling code for (1) (e,g, `L, U = lu(A)`) and then calling code for (2) to solve the system (e.g. `x = solve(L,U,b)`).

The LU factorization algorithm is straightforward (not described here), and the second step is trivial. If A is $N \times N$, then in general the factorization takes $O(N^3)$ steps and the second part takes $O(N^2)$ steps. This means that in general, solving linear systems is expensive and scales poorly as the size of the matrix increases.

[7] See, for instance, *Numerical Recipes* for a good practical source or Golub and Van Loan's *Matrix computations* for details and theory

5.1 LU for banded matrices

However, if A is banded then the solution process is much more efficient. The result:

(Banded LU) Let A be a banded matrix with m_1 lower and m_2 upper bands and bandwidth $m = m_1 + m_2 + 1$. Then it can be factored as $A = LU$ where

- L is lower triangular with m_1 lower bands
- U is upper triangular with $\leq m_1 + m_2$ upper bands

Computing $A = LU$ takes $O(m^2 N)$ operations, as does solving $Ax = b$.

Because A only contains at most mN elements, we only need to store it in a 'compressed' form as an $N \times m$ array of diagonals. For example,

$$\begin{bmatrix} \beta_1 & \gamma_1 & 0 & \cdots & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \alpha_{N-1} & \beta_{N-1} & \gamma_{N-1} \\ 0 & \cdots & 0 & \alpha_N & \beta_N \end{bmatrix} \implies \begin{bmatrix} 0 & \beta_1 & \gamma_1 \\ \alpha_2 & \beta_2 & \gamma_2 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \alpha_N & \beta_N & 0 \end{bmatrix}$$

Important note (padding): Because off-diagonals have less entries than the main diagonal, there will be some unused entries in the array. A choice must be made on how to arrange the data ('padding' the array with zeros). The padded zeros are shown in red above, using the convention that the **j-th row** of A is the j-th row of the compressed array[8].
As another example,

$$A = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 7 \\ 0 & 8 & 9 & 10 \\ 0 & 0 & 11 & 12 \end{bmatrix}, \; m_1 = 1, \, m_2 = 2, \implies A \text{ (array)} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 0 \\ 11 & 12 & 0 & 0 \end{bmatrix}$$

Using the algorithms: A banded LU solver has two routines: a routine that computes $A = LU$ given A in compress form, and a routine that solves $Ax = b$ given L, U (also compressed) and b (see example code for typical usage). To solve $Ax = b$:

- Construct A in compressed form (be careful with indexing!)
- Call the routine to obtain L and U
- Call the routine to solve $LUx = b$

A brief note about other methods: If the matrix is **sparse** (has mostly non-zero entries, e.g. an $N \times N$ matrix with $O(N)$ entries) but not banded, then other techniques must

[8] Matlab uses the opposite convention for square matrices where columns line up; see the documentation of `spdiags` for details.

be used. There are two common ones: **iterative methods** (e.g. GMRES) that construct a sequence of approximations and do not require a specific pattern of non-zeros, and **re-ordering** techniques, where rows and columns and other transforms are applied to move the non-zeros near the diagonal.

Lastly, if the matrix is 'almost banded', such as the 'periodic' tridiagonal matrix

$$\begin{bmatrix} \beta_1 & \gamma_1 & 0 & \cdots & \alpha_1 \\ \alpha_2 & \beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \alpha_{N-1} & \beta_{N-1} & \gamma_{N-1} \\ \gamma_N & \cdots & 0 & \alpha_N & \beta_N \end{bmatrix}$$

That arises in solving problems with periodic boundary conditions, then the factorization can be converted to a series of banded solves using the **Sherman-Morrison** formula, which gives the inverse of a matrix $A + \mathbf{u}\mathbf{v}^T$ in terms of A^{-1} (where $\mathbf{u}, \mathbf{v}$ are vectors and $\mathbf{u}\mathbf{v}^T$ is an outer product).

6 More notes on BVPs

6.1 Quick example of a system

Let's consider the BVP

$$(q(y)y')' + cy' = f(y), \quad y(0) = 0, \quad y(1) = 3$$

with point $x_n = nh$ and $h = 1/(N+1)$. Suppose we want to convert to a first order system and use the midpoint rule (this is useful e.g. if we have some non-uniform mesh to use). For convenience, we use u and v as system variables, setting

$$u = y, \quad v = qy', \quad \mathbf{u} = (u, v), \quad \mathbf{w} = (u_0, v_0, u_1, v_1, \cdots, u_{N+1}, v_{N+1}).$$

Remark (picking the right variable) It's worth highlighting that the natural choice of variables is (y, qy') and not (y, y'). It's often a good idea to choose your variables to simplify the system; for physical problems, physical intuition can guide you. Here, qy' represents a flux in a diffusion problem, and is a natural quantity to consider.

Then the problem to solve is

$$qu' = v, \quad v' = f(u) - cv, \qquad u(0) = u(1) = 0.$$

Applying the midpoint discretization (assuming a uniform step size for simplicity), we get

$$q_{n+1/2}\frac{u_{n+1} - u_n}{h} = v_{n+1/2}, \quad \frac{v_{n+1} - v_n}{h} = f_{n+1/2} - cv_{n+1/2}, \quad n = 0, \cdots, N$$

along with $u_0 = 0, \quad u_{N+1} = 3$. This provides $2(N+2)$ equations for all the values in $\mathbf{w}$, with two of them specified exactly. For the values at $n + 1/2$, an average is used. In some cases, there are two choices, e.g.

$$q_{n+1/2} = \frac{1}{2}(q(u_n) + q(u_{n+1})) \text{ or } q_{n+1/2} = q(\frac{u_n + u_{n+1}}{2})$$

both of which have the same order but slightly different properties (depending on the problem); not discussed here.

To be explicit, suppose $q(y) = 1$ and $f(y) = ay$ and $c = 0$, so

$$y'' = ay, \quad y(0) = 0, \quad y(1) = 3$$

As before, we set $u = y, v = y'$ and $\mathbf{w} = (u_0, v_0, \cdots, u_{N+1}, v_{N+1})$ and set

$$G_n(\mathbf{w}) := u_{n+1} - u_n - \frac{h}{2}(v_n + v_{n+1}),$$
$$H_n(\mathbf{w}) := v_{n+1} - v_n - \frac{ah}{2}(u_n + u_{n+1}), \quad n = 0, \cdots, N$$

along with $u_0 = u_{N+1} = 0$. Stacking the equations in the order

$$u_0 = 0, \quad G_0, H_0, G_1, H_1, \cdots G_N, H_N, \cdots u_{N+1} = 0.$$

we get the following linear system:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ A_0 & B_0 & 0 & \cdots & 0 \\ 0 & A_1 & B_1 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & A_N & B_N \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_0 \\ v_0 \\ u_1 \\ \vdots \\ u_{N+1} \\ v_{N+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 3 \end{bmatrix}$$

where

$$A_n = \begin{bmatrix} \dfrac{\partial G_n}{\partial u_n} & \dfrac{\partial G_n}{\partial v_n} \\ \dfrac{\partial H_n}{\partial u_n} & \dfrac{\partial H_n}{\partial v_n} \end{bmatrix} = \begin{bmatrix} -1 & -h/2 \\ -\frac{ah}{2} & -1 \end{bmatrix}$$

$$B_n = \begin{bmatrix} \dfrac{\partial G_n}{\partial u_{n+1}} & \dfrac{\partial G_n}{\partial v_{n+1}} \\ \dfrac{\partial H_n}{\partial u_{n+1}} & \dfrac{\partial H_n}{\partial v_{n+1}} \end{bmatrix} = \begin{bmatrix} 1 & -\frac{h}{2} \\ -\frac{ah}{2} & 1 \end{bmatrix}$$

Note that to be efficient, one should 'plug in' the values of u_0 and u_{N+1} and reduce the system size by two; it is omitted here just to keep the structure simple.

6.2 Non-uniform meshes

(See Sec. 18.5 of *Numerical Recipes*, from which this section is adapted). We return to the midpoint approach to the first order BVP

$$\mathbf{y}' = F(x, \mathbf{y}), \quad \cdots .$$

With uniform spacing h , a midpoint discretization could be

$$\frac{\mathbf{u}_{n+1} - \mathbf{u}_n}{h} = F(x_{n+1/2}, \frac{\mathbf{u}_{n+1} + \mathbf{u}_n}{2}).$$

Often, it is better to have a non-uniform distribution of points where there are more points in areas that need it (e.g. fast-varying parts of the solution needing more points).

One approach is to turn x into a **dependent** variable $x(q)$, where the new independent variable q is discretized with equal spacing. We choose the map from $q \to x$.

Often we want a **mesh density** $\rho(x)$ (a function that gives the density of points on the grid, so $\int_a^b \rho(x) = 1$ and the number of points between x_1 and x_2 is $\approx N \int_{x_1}^{x_2} \rho(x)\, dx$).

Example problem: Suppose, for the sake of an example, the density should be higher where the slope of the first component y_1 is large:

$$|\frac{dy_1}{dx}| \text{ large} \implies \text{high density } \rho(x). \tag{6.1}$$

Details: First, let us assume that q varies from 0 to 1 with uniform spacing Δq. Since

$$\frac{d\mathbf{y}}{dq} = \frac{d\mathbf{y}}{dx}\frac{dx}{dq} = F\frac{dx}{dq}$$

we can discretize the above as an ODE with respect to q using the midpoint scheme. The function $x(q)$ could be chosen to get the density we want, but we may not know exactly where the points should be in advance. Moreover, we need to ensure that q varies from 0 to 1 (our fixed interval). We seek q and a constant K so that

$$q(x) = \frac{1}{K}\int_0^x \rho(\xi)\, d\xi, \qquad q(1) = 1.$$

In addition, let

$$Q(x) = \int_0^x \rho(\xi)\, d\xi,$$

the anti-derivative of ρ. The value K (the normalization constant) is unknown, so we use the constant to dependent variable trick. This gives ODEs (all with respect to q)

$$\frac{dQ}{dq} = K, \quad \frac{dK}{dq} = 0, \quad \frac{dx}{dq} = \frac{K}{\rho}$$

which is then added to the main ODE

$$\frac{d\mathbf{y}}{dq} = F\frac{dx}{dq}$$

leading to a system for $(\mathbf{y}, x, K, Q)$ with independent variable q. The density can then be chosen with the desired features, for instance

$$\rho(x) = h_0 + C|\frac{dy_1}{dx}|^p$$

where h_0, C and p are constants to be configured. Note that ρ can depend on the solution and its derivatives because it is part of the ODE, so it has access to y etc.

Obviously, a good choice of density will greatly improve efficiency (and often stability) by using less points where they are not needed.

In addition, the strategy here is powerful because it provides a partly automated way to select grid points, allowing one to use the solver to explore the features of the equation without knowing everything in advance.

6.3 Galerkin

The Galerkin approach also represents the solution in terms of a basis, but imposes a very different condition on the approximation. The idea of the Galerkan method, which is typically what is known as 'the' **finite element method**, goes as follows. Again, for example, let us consider the equation

$$y'' + cy = f(x), \quad y(0) = y(1) = 0.$$

We consider approximation in terms of a basis $\{\phi_k\}_{k=0}^M$ for an approximating subspace of the set

$$\{y : y(0) = y(1) = 0\}.$$

The first step is to derive the **weak form** of the problem. We define the inner product

$$\langle f, g\rangle = \int_0^1 f(x)g(x)\,dx$$

and take the inner product of the ODE with an arbitrary (smooth)[9]function ψ satisfying the boundary conditions to get

$$-\langle y', \psi'\rangle + c\langle y, \psi\rangle = \langle f, \psi\rangle \quad \text{for all test functions } \psi \text{ with } \psi(0) = \psi(1) = 0 \tag{6.2}$$

which is the weak form. Note that the BCs were used to integrate by parts and have otherwise vanished. We now suppose our solution can be approximated in terms of the given basis,

$$y(x) \approx \sum_{k=0}^{M} a_k\phi_k(x).$$

Obviously, we cannot require this approximation satisfies the weak form, since that must hold for a continuous set of (arbitrary) ψ's. Instead, we impose the condition that

$$(6.2) \text{ is satisfied for all test functions } \psi \text{ in } \operatorname{span}\{\phi_k\}$$

which leads to the equations

$$-\langle\sum_{k=0}^{M} a_k\phi_k'(x), \phi_n'\rangle + c\langle\sum_{k=0}^{M} a_k\phi_k(x), \phi_n\rangle = \langle f, \phi_n\rangle \quad \text{for } n = 0, \cdots M.$$

Note that for (6.2) to hold for all ψ's in the span of the ϕ_k's, it suffices to require that it holds for the basis functions.

In contrast to collocation, we do not require any condition to hold at a given mesh point; all conditions are in terms of the weak form (involving only integrals over the interval).

This yields the system

$$V\vec{a} = \vec{f}$$

[9]I am omitting some theory here that is not needed for the brief discussion of the numerics.

where

$$f_k = \langle f, \phi_k \rangle = \int_0^1 f(x)\phi_k(x)\,dx,$$

$$V_{nk} = -\langle \phi_k', \phi_n' \rangle + c\langle \phi_k, \phi_n \rangle.$$

Now the subtle part: how do we choose the basis functions? A full discussion is beyond the scope of the notes, but one simple choice should illustrate the point. Suppose we want an approximation at a set of mesh points $\{x_k\}$ (for $k = 0, \cdots, M$).

We want basis functions with the property that

$$\langle \phi_k, \phi_n \rangle = 0 \text{ if } k \text{ is not close to } n$$

so that the resulting system will be sparse (in this case, banded). In addition, the basis should be enough to represent any mesh function on the $\{x_k\}$'s.

The piecewise linear basis we encountered in discussing splines is a good candidate. Recall these are piecewise linear 'hats'

$$\phi_k(x) = \begin{cases} (\text{line from } (x_{k-1}, 0) \text{ to } ((x_k, 1)) & x_{k-1} < x < x_k \\ (\text{line from } (x_k, 1) \text{ to } ((x_{k+1}, 0)) & x_k < x < x_{k+1} \end{cases}$$

for $k = 1, \cdots, M-1$. This is the basis we would use, as it spans mesh functions on the given points that satisfy the boundary conditions. We can then compute the quantities $\langle \phi_k', \phi_n' \rangle$ and so on and solve the system.

The method can be generalized by taking other bases of 'elements' - overlapping functions with small support - to obtain better accuracy, stability or other properties. The method is popular for complicated problems in multiple dimensions because it does not care much about the location of the mesh points, allowing one to write general code that can handle complicated geometries.