ODEs: Boundary Value Problems

1 ODE BVPs

A boundary value problem (BVP) for ODEs is a problem in which we give conditions on the solution to the ODE at different values (points) of the independent variable. Such conditions can be on the solution values, on the derivatives of the solution, or more general conditions involving nonlinear functions of the solution and derivative. A very **simple** BVP for an ODE is

$$\begin{cases} \dfrac{d^2y(x)}{dx^2} = f(x) \quad a \le x \le b, \\ y(a) = y_0, \\ y(b) = y_1. \end{cases} \tag{1}$$

The BVP (1) is a second-order ODE with two boundary conditions. The solution to the BVP is a function $y(x)$ that satisfies the ODE and the boundary conditions.

The general form of a BVP for a system of ODEs is

$$\begin{cases} \dfrac{d\mathbf{y}(x)}{dx} = F(x, \mathbf{y}(x)) \quad a \le x \le b, \\ B(\mathbf{y}(a), \mathbf{y}(b)) = \vec{0} \end{cases} \tag{2}$$

This standard form of BVP (2) is used for numerical computation of the solution. The function $F(x, \mathbf{y})$ is a vector function that defines the system of ODEs, and $B(\mathbf{y}(a), \mathbf{y}(b))$ is a vector function that defines the boundary conditions. The solution to the BVP is a vector function $\mathbf{y}(x)$ that satisfies the ODEs and the boundary conditions. Note, in Matlab, `bvp4c` is used to solve BVPs of the form (2).

The stability for BVPs are not as well understood as for IVPs. We will skip the stability analysis for BVPs in this course, and only focus on the numerical methods for solving BVPs.

The uniqueness of the solution to a BVP is not guaranteed, and it depends on the ODE and the boundary conditions. In general, a BVP may have **no solution, one solution, or multiple solutions**. The uniqueness of the solution to a BVP is a very complicated topic, and we will not cover it in this course.

1.1 Shooting method

To start, we consider a typical two-point boundary value problem (BVP) for a second-order ODE:

$$\frac{d^2y(x)}{dx^2} = f(x, y(x), y'(x)) \quad a \le x \le b,\ y(a) = c, \quad y(b) = d. \tag{3}$$

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 $x = b$.

The shooting method is a numerical method for solving BVPs. The idea is to convert the BVP into an IVP by guessing the value of the derivative at the initial point. The IVP is then solved using a numerical method, and the solution is checked against the boundary condition at the final point. If the solution does not satisfy the boundary condition, the guess for the derivative is adjusted and the IVP is solved again. This process is repeated until the solution satisfies the boundary condition.

So now we set up the procedure for the shooting method:

1. **set up IVP** Guess the value of $y'(a)$ by solving the IVP

$$\frac{d^2y(x)}{dx^2} = f(x, y(x), y'(x)) \quad x \in [a, b],\, y(a) = c, \quad y'(a) = s. \tag{4}$$

   The solution to (3) is a solution to (4) for a certain value s^* of s that we must find. Let us define

$$y(x; s) = \text{ solution to (4) for that value of } s \tag{5}$$

2. **set 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

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

   We want to find the value of s such that $G(s) = 0$. So we need to find the root of $G(s)$, and 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 (4), so each evaluation of G is quite expensive.

(a) **Bisection method** If $G(s)$ is continuous and changes sign on the interval $[s_1, s_2]$, then we can use the bisection method to find the root of $G(s)$: given any two initial guesses s_1 and s_2 we solve the initial value problem to obtain

$$G(s_1) = y(x; s_1)|_{x=b} - d, \quad G(s_2) = y(x; s_2)|_{x=b} - d.$$

If the sign of $G(s_1)G(s_2)$ is negative, then we can claim there exists a point s' in the interval $[s_1, s_2]$ such that $G(s^*) = 0$. We then split the interval $[s_1, s_2]$ into two subintervals $[s_1, s_3]$ and $[s_3, s_2]$ with $s_3 = (s_1 + s_2)/2$. And then we proceed as before.

(b) **Secant method** Similarly to the bisection method, we start with two initial guesses s_1 and s_2 and evaluate $G(s_1)$ and $G(s_2)$. Then we can construct the line passing through $(s_1, G(s_1))$ and $(s_2, G(s_2))$ and extrapolate such a line onto the x axis. By doing this iteratively, we obtain the sqeuence

$$s_{n+1} = s_n - G(s_n)\frac{s_n - s_{n-1}}{G(s_n) - G(s_{n-1})}, \ n = 2, 3, \ldots$$

(c) **Newton's method** If $G(s)$ is differentiable, we have the following iterative scheme

$$s_{n+1} = s_n - \frac{G(s_n)}{G'(s_n)} = s_n - \frac{y(b; s_n) - d}{G'(s_n)}, \ n = 1, 2, \ldots$$

where $G'(s)$ is the derivative of $G(s)$, i.e., $G'(s) = \partial y(b; s)/\partial s$.

3. Check if the solution satisfies the boundary condition at $x = b$.

1.2 Using Matlab

The function `bvp4c` in Matlab is used to solve BVPs. The function `bvp4c` requires the user to define the ODE and the boundary conditions in a function file.

The procedure for using `bvp4c` is as follows:

1. Convert Higher-Order ODEs to First-Order Systems: For an equation like $y'' = f(x, y, y')$, define:

$$y_1 = y, \quad y_2 = y' \quad \Rightarrow \quad \begin{cases} y_1' = y_2, \\ y_2' = f(x, y_1, y_2) \end{cases}$$

2. Define the ODE System: Use a function handle or MATLAB function to return the derivatives $\mathbf{y}' = f(x, \mathbf{y})$.

3. Specify Boundary Conditions: Create a function to compute residuals $\mathrm{BC}(y(a), y(b)) = 0$.

4. Provide an Initial Guess: Use ‘bvpinit‘ to define an initial mesh and guess for the solution.

5. Solve with ‘bvp4c‘: Pass the ODE function, BC function, and initial guess to ‘bvp4c‘.

6. Post-Process the Solution: Evaluate the solution at desired points using ‘deval‘ and plot results.

Example: Solve $y'' + e^y = 0$ with $y(0) = 0$, $y(1) = 0$.

1. Convert to First-Order System:

$$\begin{cases} y_1' = y_2, \\ y_2' = -e^{y_1} \end{cases}$$

2. Define the ODE Function:

```
odefun = @(x, y) [y(2); -exp(y(1))];
```

3. Define the Boundary Conditions:

```
bcfun = @(ya, yb) [ya(1); yb(1)];
```

4. Create Initial Guess** (Parabolic guess):

   ```
   guess = @(x) [4*x.*(1-x); 4*(1-2*x)]; % y = 4x(1-x), y' = 4(1-2x)\\
   solinit = bvpinit(linspace(0, 1, 10), guess);
   ```

5. Solve with ‘bvp4c‘:

   ```
   sol = bvp4c(odefun, bcfun, solinit);
   ```

6. Evaluate and plot results:

   ```
   x = linspace(0, 1, 100);
   y = deval(sol, x);
   subplot(2,1,1); plot(x, y(1,:)); title('Solution y(x)');
   subplot(2,1,2); plot(x, y(2,:)); title('Derivative y''(x)');
   ```

1.3 Finite Difference Method

The finite difference method is another numerical method for solving BVPs. The idea is to discretize the domain of the BVP into a grid of points and approximate the derivatives in the ODE using finite difference approximations. The resulting system of algebraic equations is then solved using a numerical method. The procedure is as follows:

1. Discretize the domain: Divide the interval $[a, b]$ into N subintervals of equal length $h = (b - a)/N$. Let $x_i = a + ih$ for $i = 0, 1, \ldots, N$.

2. Approximate the derivatives: Use finite difference approximations to approximate the derivatives in the ODE. For example, the first derivative can be approximated using the central difference formula

$$y'(x_i) \approx \frac{y(x_{i+1}) - y(x_{i-1})}{2h}.$$

3. Discretize the ODE: Substitute the finite difference approximations into the ODE to obtain a system of algebraic equations.

4. Discretize the boundary conditions: Substitute the boundary conditions into the system of algebraic equations, if necessary.

5. Solve the system of algebraic equations: The system of algebraic equations can be solved using a numerical method.

Example: Consider the Dirichlet Boundary Value Problem (BVP)

$$\begin{cases} u''(x) = f(x), \quad 0 < x < 1 \\ u(0) = \alpha \\ u(1) = \beta \end{cases}$$

- This is an easy problem that can be solved by integrating twice.

- We use this as a test problem to be able to solve more advances problems like $\Delta u = f(x, y)$, when we dicuss partial differential equations (PDEs).

Using finite differences over $0 < x < 1$: we divide the interval into $m+1$ subintervals of equal length $h = 1/(m+1)$.

- First discretize the grid $x_j = jh$ with $h = 1/(m+1)$, the meshwidth.
- $U_0 = u(0) = \alpha$, $U_j \approx u(x_j)$, $U_{m+1} = u(1) = \beta$.
- Use centered difference approximation to the second derivative

$$D^2 U_j = \frac{U_{j-1} - 2U_j + U_{j+1}}{h^2} = f(x_j)$$

  for $j = 1, 2, \ldots, m$

  - Write our formula for $j = 1$, $j = 2$, and $j = m$.
  - Form a linear system $A\mathbf{u} = \mathbf{f}$ and solve for U_j.

$$\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_m \end{bmatrix} = \begin{bmatrix} f(x_1) - \alpha/h^2 \\ f(x_2) \\ \vdots \\ f(x_m) - \beta/h^2 \end{bmatrix}$$

- How well does $\mathbf{u}$ approximate $u(x)$ on the entire interval $0 \le x \le 1$?
- We need to measure the errors $U_j - u(x_j)$ by discretized L^p norms.

Consistency

The **local truncation error** is defined by replacing U_j with the true solution $u(x_j)$ in the finite difference formula,

$$\tau_j = \frac{u(x_{j-1}) - 2u(x_j) + u(x_{j+1})}{h^2} - f(x_j) \ \text{ for } \ \ j = 1, 2, \ldots, m$$

- Use Taylor series to obtain the local truncation error (LTE).

- Let τ denote the vector with m entries τ_j.
- Let $\hat{U}$ denote the vector with m entries $u(x_j)$ (true solution at interior points)
- Let F denote the vector with m entries $f(x_j)$.
  - Then $\tau = A\hat{U} - F$ or $A\hat{U} = F + \tau$.

Global error: defined by $E = U - \hat{U}$, where U is a vector with entries U_j, an approximation to $u(x_j)$.

- Note that
$$AU = F. \tag{6}$$

- $$AE = A(U - \hat{U}) = F - (F + \tau) = -\tau \tag{7}$$

- Looking at AE component-wise
$$\frac{E_{j-1} - 2E_j + E_{j+1}}{h^2} = -\tau(x_j) \text{ for } \quad j = 1, 2, \ldots, m$$
  - What are E_0 and E_{m+1}?

- This is a discrete approximation of $e''(x) = -\tau$ on $0 < x < 1$ with $e(0) = e(1) = 0$.
  - $\tau(x) \approx u^{(4)}(x)h^2/12$
  - Integrate twice to obtain $e(x) \approx \dfrac{1}{12}h^2 u''(x) + ax + b$
  - What are a and b?

  - This tells us that the global error is also $\mathcal{O}(h^2)$
  - We are assuming $u(x_j) \approx U_j$ in the finite difference formula.

Consider the discrete system $A^h E^h = -\tau^h$, where h indicates the mesh spacing.

- Then $E^h = -(A^h)^{-1}\tau^h$.
- Taking norms,

$$||E^h|| = ||(A^h)^{-1}\tau^h|| \leq ||(A^h)^{-1}|| \, ||\tau^h|| \quad \text{(why?)}$$

  - Which are vector and which are matrix norms above?
  - We know that $\tau^h = \mathcal{O}(h^2)$
  - We want $||(A^h)^{-1}|| \leq C$ for all h sufficiently small.
  - Them we would have $||E^h|| \leq C||\tau^h||$.

Definition (Stability): Suppose a finite difference method for a linear BVP gives a sequence of matrix equations of the form $A^h U^h = F^h$ where h is the mesh width. We say that the method is stable if $(A^h)^{-1}$ exists for all h sufficiently small (for $h < h_0$, say) and if there is a constant C, independent of h, such that $||(A^h)^{-1}|| \leq C$ for all $h < h_0$.

We say that a method is **consistent** with the differential equation and boundary conditions if $||\tau^h|| \to 0$ as $h \to 0$.

- Typically have $||\tau^h|| = \mathcal{O}(h^p)$ for some integer $p > 0$.

A method is said to be **convergent** if $||E^h|| \to 0$ as $h \to 0$.

- From last time,

$$||E^h|| = ||(A^h)^{-1}\tau^h|| \leq ||(A^h)^{-1}||\,||\tau^h|| \leq C||\tau^h|| \to 0, \;\; h \to 0.$$

The fundamental theorem of finite difference methods states that

consistency + stability $\Longrightarrow$ convergence.

- Usually this translates to

$\mathcal{O}(h^p)$ local truncation error + stability $\Longrightarrow$ $\mathcal{O}(h^p)$ global error

1.3.1 Boundary copnditions

Consider the boundary value problem $u'' = f(x)$ with boundary conditions $u'(0) = \sigma$ and $u(1) = \beta$. How do we solve for U_0?

- Try using forward differences to discretize the boundary condition at $x = 0$

- Form the linear system to solve for $U_0, \ldots, U_{m+1}$

- Compute τ_0, the local truncation error at U_0.

Other approaches: Ghost cells

- Add U_{-1} to the system via centered differences

$$\frac{U_1 - U_{-1}}{2h} = \sigma$$

- Use equation for second derivative to eliminate variable U_{-1},

$$\frac{U_{-1} - 2U_0 + U_1}{h^2} = f(x_0)$$

Use one sided second order finite difference formula to approximate $u'(0)$

$$\frac{1}{h}\left(\frac{3}{2}U_0 - 2U_1 + \frac{1}{2}U_2\right) = \sigma$$

- How does this change the linear system?

Existence and Uniqueness: Consider the BVP $u'' = f(x)$, $0 < x < 1$ with boundary conditions $u'(0) = \sigma_0$ and $u'(1) = \sigma_1$.

- The resulting linear system is $A\vec{U} = \vec{F}$, where

$$A = \frac{1}{h^2}\begin{bmatrix} -h & h & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 1 \\ & & & & & h & -h \end{bmatrix}, \quad \vec{U} = \begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_m \\ U_{m+1} \end{bmatrix}$$

$$\vec{F} = \begin{bmatrix} \sigma_0 + \frac{h}{2} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_m) \\ -\sigma_1 + \frac{h}{2} f(x_{m+1}) \end{bmatrix}$$

◦ Multiply A by the vector $\vec{V} = [1,\, 1,\, \ldots,\, 1]^T$.

◦ The problem is not well-posed.

1.3.2 A general linear second order equation

Consider a more general linear equation with boundary conditions,

$$a(x)u''(x) + b(x)u'(x) + c(x)u(x) = f(x), \quad u(a) = \alpha, \quad u(b) = \beta$$

- Consider the second order discretizations

  $$a_i \left(\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} \right) + b_i \left(\frac{U_{i+1} - U_{i-1}}{2h} \right) + c_i U_i = f_i,$$

  where $a_i = a(x_i)$, $b_i = b(x_i)$, and $c_i = c(x_i)$.

- Form the matrix A and vectors $\vec{U}$, $\vec{F}$ to solve the linear system $A\vec{U} = \vec{F}$ for $U_i \approx u(x_i)$.