CHAPTER 7: The Shooting Method

- A simple, intuitive method that builds on IVP knowledge and software.

Not recommended for general BVPs!

But OK for relatively easy problems that may need to be solved many times.

**Idea:** Guess all unknown initial values. (aim)

Integrate to $b$. (shoot)

(Try to hit BCs at $x = b$.)

Adjust initial guesses and repeat.

Fundamental disadvantage: directionality imposed on BVP.

$\rightarrow$ Shooting inherits stability of IVP (not just BVP).
7.1 Single Shooting

\[ y' = f(x, y), \quad a < x < b, \]
\[ g(y(a), y(b)) = 0, \quad m \text{ nonlinear equations.} \]

Let \( y(x) = y(x; y_a) \) be the solution of the ODE with initial value \( y_a \).

We want to choose \( y_a \) to solve

\[ h(y_a) = g(y_a; y(b; y_a)) = 0. \]

Software needs two parts:

- IVP solver (ode45, ode15s, your own, ...).
- Nonlinear algebraic equation solver (Newton; for scalar case: bisection, secant, ...).
• Bisection (scalar case only): find two initial values $y^{(1)}_a, y^{(2)}_a$ such that $h(y^{(1)}_a), h(y^{(2)}_a)$ differ in sign. 

Set 

$$y^{(3)}_a = \frac{1}{2}(y^{(1)}_a + y^{(2)}_a).$$

Evaluate 

$$h(y^{(3)}_a).$$

If $\text{sgn}(h(y^{(3)}_a)) = \text{sgn}(h(y^{(1)}_a))$, 

set 

$$y^{(4)}_a = \frac{1}{2}(y^{(3)}_a + y^{(2)}_a).$$

Else set 

$$y^{(4)}_a = \frac{1}{2}(y^{(3)}_a + y^{(1)}_a).$$

Repeat to convergence.

• Newton:


  - Quasi-Newton methods are more efficient in practice (freeze Jacobian, etc.)
7.1.1 Problems with Single Shooting

In converting from BVP to IVP, you convert stability of BVP to stability of IVP. 

presumably, this is ok this may be bad!

→ You can convert a nice problem into a nasty one! e.g., shooting assumes the IVPs have solutions all the way to \( x = b \) even for bad guesses of \( y_a \! \! \). 

Example 1. \( y' = A(x)y + q(x), \)  
where 

\[
A = \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
-2\lambda^3 & \lambda^2 & 2\lambda \\
\end{pmatrix},
\]

\( y_1(0) = \beta_1, \ y_1(1) = \beta_2, \ y_2(0) = \beta_3, \)
with exact solution

\[ y(x) = \begin{pmatrix} u(x) \\ u'(x) \\ u''(x) \end{pmatrix}, \]

\[ u(x) = \frac{e^{\lambda(x-1)} + e^{2\lambda(x-1)} + e^{-\lambda x}}{2 + e^{-\lambda}} + \cos(\pi x). \]

**Note 1.** \( q(x), \beta \) can be determined from exact solution (to make it be the exact solution).

For \( \lambda \approx 20 \), BVP is stable but IVP is not!

\[
\begin{align*}
\lambda &= 1 \quad \leftrightarrow \quad \text{Shooting ok.} \\
\lambda &= 10 \quad \leftrightarrow \quad \text{Wrong (but plausible!) solution.} \\
\lambda &= 20 \quad \leftrightarrow \quad \text{Error} \sim 200. \\
\lambda &= 50 \quad \leftrightarrow \quad \text{Error} \sim 10^{32}.
\end{align*}
\]
Figure 7.1: Exact (solid line) and shooting (dashed line) solutions for Example 7.2.

Figure 7.2: Exact (solid line) and shooting (dashed line) solutions for Example 7.2.
7.2 Multiple Shooting

Problems with single shooting are exacerbated when $b$ is large.

Idea: Restrict the sizes of the intervals over which the various IVPs are integrated.

Define a mesh

$$a = x_0 < x_1 < \cdots < x_{N-1} < x_N = b.$$ 

Solve $y' = f(x, y)$ on each subinterval $[x_{n-1}, x_n]$.

Then patch them together to form solution on $[a, b]$. 

------

Diagram: Graph showing intervals $[a, b]$ divided into subintervals with solution curves.
Let $y_n(x; c_{n-1})$ solve

$$
y'_n = f(x, y_n), \quad x_{n-1} < x < x_n,
$$
$$
y_n(x_{n-1}) = c_{n-1}, \quad n = 1, 2, \ldots, N.
$$

Assuming these IVPs are solved exactly, the exact solution to the BVP satisfies

$$
y(x) = y_n(x; c_{n-1}), \quad x_{n-1} \leq x \leq x_n, \quad n = 1, 2, \ldots, N,
$$

where

$$
y_n(x_n; c_{n-1}) = c_n, \quad n = 1, 2, \ldots, N - 1, \quad (1)
$$
$$
g(c_0, y_N(b; c_{n-1})) = 0.
$$

Equations (1) are patching (continuity) conditions.
\[ N \text{m algebraic equations for } N \text{m unknowns.} \]

\[
\mathbf{c} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{pmatrix}
\]

with each \( c_n, \ n = 0, 1, \ldots, N - 1 \), of length \( m \).

Write as nonlinear system

\[ h(\mathbf{c}) = 0. \]

Apply Newton’s method

\[
A(\mathbf{c}^{(\nu+1)} - \mathbf{c}^{(\nu)}) = -h(\mathbf{c}^{(\nu)}),
\]

\[
A = \frac{\partial h}{\partial \mathbf{c}} \bigg|_{\mathbf{c}^{(\nu)}}.
\]
\( A \) has a sparse block structure

\[
A = \begin{bmatrix}
  -Y_1(t_1) & I & & \\
  & -Y_2(t_2) & I & \\
  & & \ddots & \\
  & & & -Y_{N-1}(t_{N-1}) & I \\
  B_a & & & & B_b Y_N(b)
\end{bmatrix}.
\]

Variants of Gauss elimination that take advantage of sparsity can solve in \( \mathcal{O}(N) \) time. (In parallel, it can be \( \mathcal{O}(\log N) \).)

Note that the blocks \( Y_n(t_n) \) can also be constructed in parallel, so sometimes multiple shooting is known as parallel shooting.

Matrix \( A \) turns out to be the same as if you applied multiple shooting to the linearized BVP.
• Multiple shooting “solves” the most serious problems of single shooting (i.e., bad conditioning, finite escape time).
  
  e.g., Multiple shooting solves Example 1 for \( \lambda = 20 \) with no problem.

But it is not so simple to code anymore!

Also you may need many subintervals

\[ \uparrow \]

inefficient

\( (N \text{ grows linearly with } \lambda. ) \)