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

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

Let $\mathbf{y}(x) = \mathbf{y}(x; \mathbf{y}_a)$ be the solution of the ODE with initial value $\mathbf{y}_a$.

We want to choose $\mathbf{y}_a$ to solve

$$\mathbf{h}(\mathbf{y}_a) = \mathbf{g}(\mathbf{y}_a; \mathbf{y}(b; \mathbf{y}_a)) = \mathbf{0}.$$

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 $\mathbf{y}_a^{(1)}, \mathbf{y}_a^{(2)}$ such that $h(\mathbf{y}_a^{(1)}), h(\mathbf{y}_a^{(2)})$ differ in sign.

  Set $$\mathbf{y}_a^{(3)} = \tfrac{1}{2}(\mathbf{y}_a^{(1)} + \mathbf{y}_a^{(2)}).$$

  Evaluate $$h(\mathbf{y}_a^{(3)}).$$

  If $\text{sgn}\big(h(\mathbf{y}_a^{(3)})\big) = \text{sgn}\big(h(\mathbf{y}_a^{(1)})\big)$,
  set $\mathbf{y}_a^{(4)} = \tfrac{1}{2}(\mathbf{y}_a^{(3)} + \mathbf{y}_a^{(2)})$.

  Else set $$\mathbf{y}_a^{(4)} = \tfrac{1}{2}(\mathbf{y}_a^{(3)} + \mathbf{y}_a^{(1)}).$$

  Repeat to convergence.

- Newton:
  More complicated (see text pp. 178–180).

  \- 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 $\underbrace{\text{stability of BVP}}_{\text{presumably, this is ok}}$ to $\underbrace{\text{stability of IVP}}_{\text{this may be bad!}}$.

$\rightarrow$ 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 $\mathbf{y}_a$!

Example 1. $\mathbf{y}' = \mathbf{A}(x)\mathbf{y} + \mathbf{q}(x)$,
where

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

$$y_1(0) = \beta_1,\ y_1(1) = \beta_2,\ y_2(0) = \beta_3,$$

with exact solution

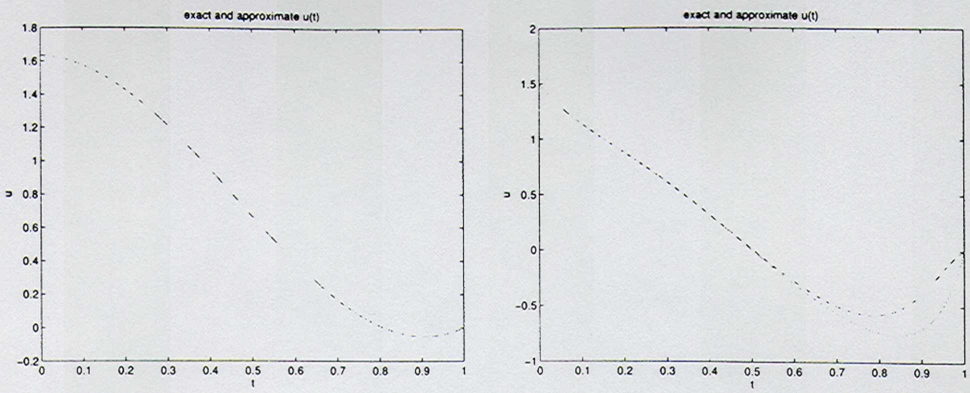
$$\mathbf{y}(x) = \begin{pmatrix} u(x) \\ u'(x) \\ u''(x) \end{pmatrix},$$

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

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

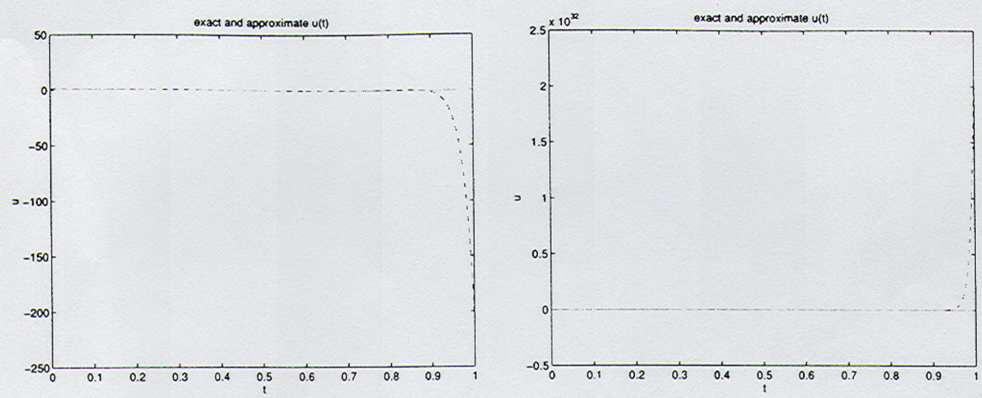
For $\lambda \approx 20$, *BVP is stable but IVP is not!*

$\lambda = 1 \quad \leftrightarrow$ *Shooting ok.*
$\lambda = 10 \quad \leftrightarrow$ *Wrong (but plausible!) solution.*
$\lambda = 20 \quad \leftrightarrow$ *Error* ~ 200.
$\lambda = 50 \quad \leftrightarrow$ *Error* $\sim 10^{32}$.



(a) $\lambda = 1$ (b) $\lambda = 10$

Figure 7.1: *Exact (solid line) and shooting (dashed line) solutions for Example* 7.2.



(a) $\lambda = 20$ (b) $\lambda = 50$

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.

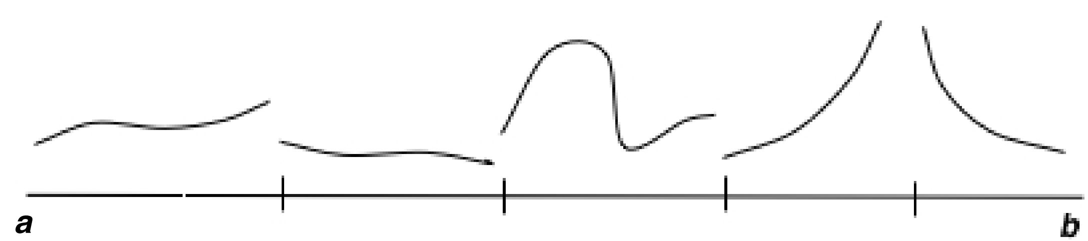
Idea: Restrict the sizes of the intervals over which the various IVPs are integrated.

Define a mesh

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

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

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



Let $\mathbf{y}_n(x; \mathbf{c}_{n-1})$ solve

$$\begin{aligned} \mathbf{y}_n' &= \mathbf{f}(x, \mathbf{y}_n), \qquad x_{n-1} < x < x_n, \\ \mathbf{y}_n(x_{n-1}) &= \mathbf{c}_{n-1}, \qquad n = 1, 2, \cdots, N. \end{aligned}$$

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

$$\mathbf{y}(x) = \mathbf{y}_n(x; \mathbf{c}_{n-1}),\ x_{n-1} \le x \le x_n,\ n = 1, 2, \cdots, N,$$

where

$$\begin{aligned} \mathbf{y}_n(x_n; \mathbf{c}_{n-1}) &= \mathbf{c}_n,\ n = 1, 2, \cdots, N-1, \quad (1) \\ \mathbf{g}(\mathbf{c}_0, \mathbf{y}_N(b; \mathbf{c}_{n-1})) &= \mathbf{0}. \end{aligned}$$

Equations (1) are *patching* (continuity) conditions.

$\rightarrow Nm$ algebraic equations for Nm unknowns.

$$\mathbf{c} = \begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{N-1} \end{pmatrix}$$

with each $\mathbf{c}_n$, $n = 0, 1, ..., N-1$, of length m.

Write as nonlinear system

$$\mathbf{h}(\mathbf{c}) = \mathbf{0}.$$

Apply Newton's method

$$\begin{aligned} \mathbf{A}(\mathbf{c}^{(\nu+1)} - \mathbf{c}^{(\nu)}) &= -\mathbf{h}(\mathbf{c}^{(\nu)}), \\ \mathbf{A} &= \left.\frac{\partial \mathbf{h}}{\partial \mathbf{c}}\right|_{\mathbf{c}^{(\nu)}}. \end{aligned}$$

$\mathbf{A}$ has a sparse block structure

$$\mathbf{A} = \begin{bmatrix} -\mathbf{Y}_1(t_1) & \mathbf{I} & & & \\ & -\mathbf{Y}_2(t_2) & \mathbf{I} & & \\ & & \ddots & \ddots & \\ & & & -\mathbf{Y}_{N-1}(t_{N-1}) & \mathbf{I} \\ \mathbf{B}_a & & & & \mathbf{B}_b\mathbf{Y}_N(b) \end{bmatrix}.$$

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 $\mathbf{Y}_n(t_n)$ can also be constructed in parallel, so sometimes multiple shooting is known as *parallel shooting*.

Matrix $\mathbf{A}$ 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

  $\updownarrow$

  inefficient

  (N grows linearly with λ.)