Numerical Methods for ODE

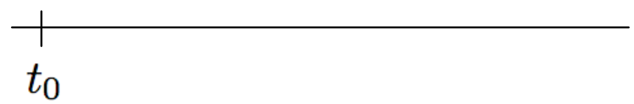
“Mathematics is an experimental science, and definitions do not come first, but later on,” Oliver Heaviside

Initial Versus Boundary Value Problems

Initial Value Problems (IVP):

$$\frac{du}{dt} = f(t, u(t)) \ , \ t \geq t_0$$

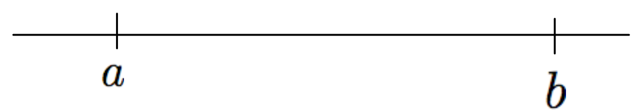
$$u(t_0) = u_0$$



Boundary Value Problems (BVP):

$$y''(x) = f(x, y, y') \ , \ a \leq x \leq b$$

$$y(a) = \alpha \ , \ y(b) = \beta$$

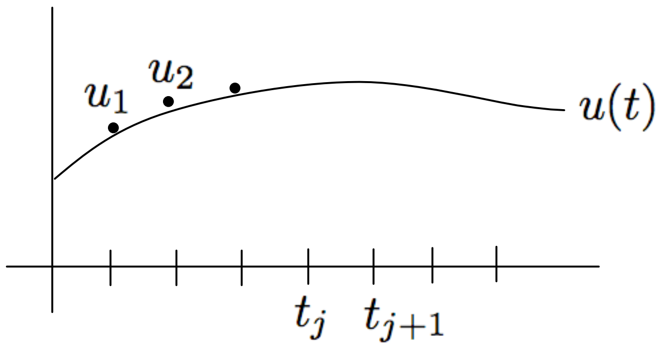


Numerical Methods for IVP: Euler's Method

Initial Value Problem:

$$\frac{du}{dt} = f(t,u)\ ,\ t \geq 0$$

$$u(0) = u_0$$



Notation: $t_j = jk$ for $j = 0, 1, \cdots$

$$u_j \approx u(t_j)$$

Taylor Series:

$$u(t_{j+1}) = u(t_j) + k\frac{du}{dt}(t_j) + \frac{k^2}{2}\frac{d^2u}{dt^2}(t_j) + \cdots + \frac{k^n}{n!}\frac{d^nu}{dt^n}(t_j) + \frac{k^{n+1}}{(n+1)!}\frac{d^{n+1}u}{dt^{n+1}}(\xi)$$

Euler's Method:

$$u(t_{j+1}) = u(t_j) + k\dot{u}(t_j) + \mathcal{O}(k^2)$$

$$\Rightarrow u_{j+1} = u_j + kf(t_j, u_j)$$

Accuracy: Local truncation error $\mathcal{O}(k^2)$

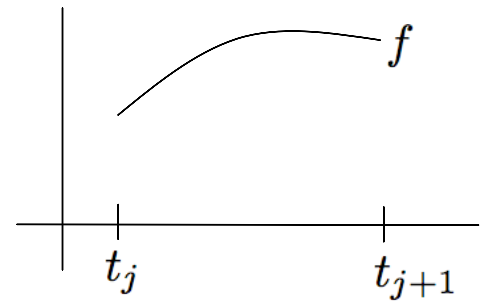
Global truncation error $\mathcal{O}(k)$

Assumptions:

Euler and Implicit Euler Methods

Note:

$$u(t_{j+1}) = u(t_j) + \int_{t_j}^{t_{j+1}} f(t, u(t))dt$$



Euler's Method: Left Endpoint

$$u_{j+1} = u_j + kf(t_j, u_j)$$

Implicit Euler: Right Endpoint

$$u_{j+1} = u_j + kf(t_{j+1}, u_{j+1})$$

Stability: Apply method to

$$\dot{u}(t) = \lambda u \ , \ \ \lambda < 0$$

$$u(0) = u_0$$

Forward Euler

$$\begin{aligned} u_{j+1} &= u_j + \lambda k u_j \\ &= (1+\lambda k)^{j+1} u_0 \end{aligned}$$

$$\Rightarrow |1 + \lambda k| < 1$$

$$\Rightarrow k < \frac{-2}{\lambda}$$

Implicit Euler

$$\begin{aligned} u_{j+1} &= u_j + \lambda u_{j+1} \\ &= \left(\frac{1}{1-\lambda k}\right)^{j+1} u_0 \end{aligned}$$

$$\Rightarrow 1 < |1 - \lambda k|$$

$$\Rightarrow k > 0$$

Runge-Kutta-Feylberg Methods

4th Order Runge-Kutta:

$$k_1 = kf(t_j, u_j)$$

$$k_2 = kf\left(t_j + \frac{k}{2}, u_j + \frac{k_1}{2}\right)$$

$$k_3 = kf\left(t_j + \frac{k}{2}, u_j + \frac{k_2}{2}\right)$$

$$k_4 = kf(t_{j+1}, u_j + k_3)$$

$$u_{j+1} = u_j + \frac{1}{6}\left(k_1 + k_2 + k_3 + k_4\right)$$

Accuracy: Local Truncation error is 4th-order if u(t) has five continuous derivatives.

Runge-Kutta-Feylberg: Use R-K method with 5th order truncation error to estimate local error in 4th order R-K method to choose appropriate stepsize.

MATLAB ODE Routines

Algorithms: From the MATLAB ODE documentation

- **ode45** is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a one-step solver - in computing y(tn), it needs only the solution at the immediately preceding time point, y(tn-1). In general, ode45 is the best function to apply as a "first try" for most problems.

- **ode23** is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It may be more efficient than ode45 at crude tolerances and in the presence of moderate stiffness. Like ode45, ode23 is a one-step solver.

- **ode113** is a variable order Adams-Bashforth-Moulton PECE solver. It may be more efficient than ode45 at stringent tolerances and when the ODE file function is particularly expensive to evaluate. ode113 is a multistep solver - it normally needs the solutions at several preceding time points to compute the current solution.

- The above algorithms are intended to solve nonstiff systems. If they appear to be unduly slow, try using one of the stiff solvers below.

- **ode15s** is a variable order solver based on the numerical differentiation formulas (NDFs). Optionally, it uses the backward differentiation formulas (BDFs, also known as Gear's method) that are usually less efficient. Like ode113, ode15s is a multistep solver. Try ode15s when ode45 fails, or is very inefficient, and you suspect that the problem is stiff, or when solving a differential-algebraic problem.

- **ode23s** is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it may be more efficient than ode15s at crude tolerances. It can solve some kinds of stiff problems for which ode15s is not effective.

- **ode23t** is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve DAEs.

- **ode23tb** is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages. Like ode23s, this solver may be more efficient than ode15s at crude tolerances.

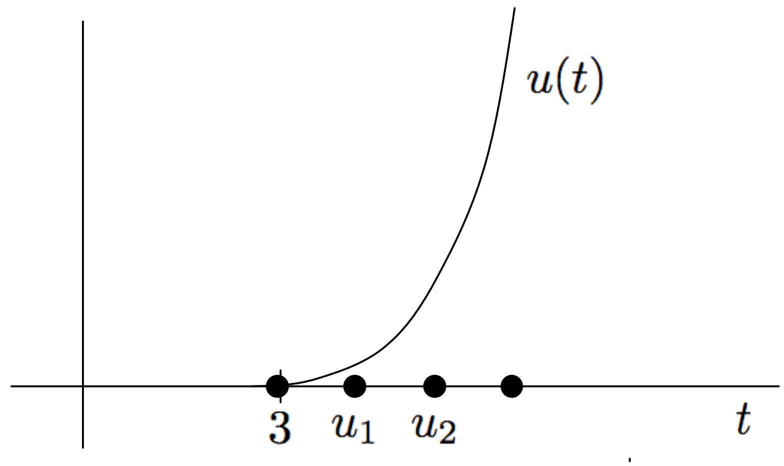
MATLAB ODE Routines: From the Documentation

Solver	Problem Type	Order of Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. This should be the first solver you try.
ode23	Nonstiff	Low	For problems with crude error tolerances or for solving moderately stiff problems.
ode113	Nonstiff	Low to High	For problems with stringent error tolerances or for solving computationally intensive problems.
ode15s	Stiff	Low to Medium	If ode45 is slow because the problem is stiff
ode23s	Stiff	Low	If using crude error tolerances to solve stiff systems and the mass matrix is constant.
ode23t	Moderately Stiff	Low	For moderately stiff problems if you need a solution without numerical damping.
ode23tb	Stiff	Low	If using crude error tolerances to solve stiff systems.

Example 1

Problem:

$$\dot{u}(t) = u^{2/3}$$
$$u(3) = 0$$



Analytic Solution:

$$u(t) = \left(\frac{1}{3}t - 1\right)^3$$

Euler's Method: $u_{j+1} = u_j + k u_j^{2/3}$

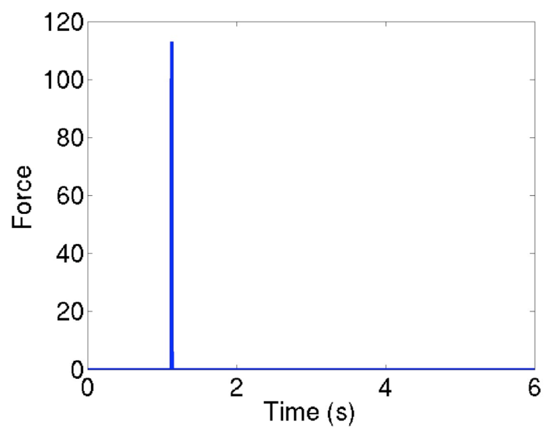
4th Order R-K: $k_1 = k_2 = k_3 = k_4 = 0 \Rightarrow u_1 = 0$

Similarly, $u_2 = u_3 = \cdots = 0$

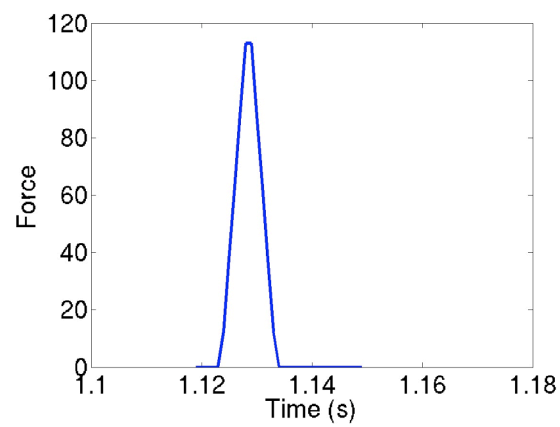
What is going on?

Example 2

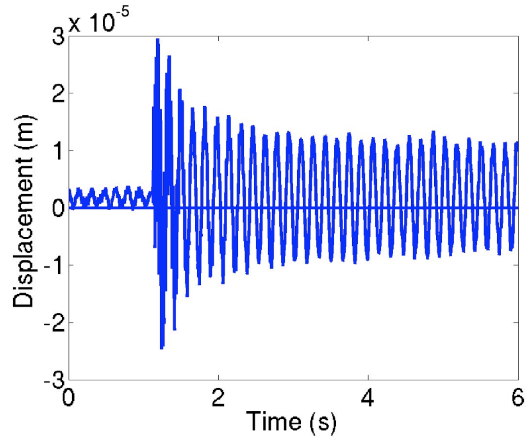
Experimental Beam Data:



Voltage Input



Voltage Input (Zoomed View)



Displacement

Lumped Model:

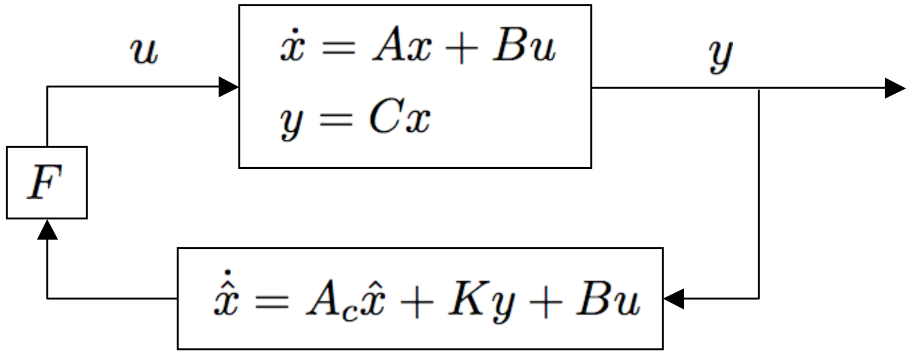
$$m\ddot{u} + c\dot{u} + ku = \widehat{\delta}(t - 1.128)$$

Notes:

- Initial conditions?
- Experimental input $\widehat{\delta}(t - 1.128) \approx \delta(t - 1.128)$
- How will ODE solvers accommodate the experimental input?
- How can you test numerical codes?

Example 3

Feedback Control Design:



State Estimator:

$$\begin{aligned}\dot{\hat{x}} &= A\hat{x} + Bu + K(y - \hat{y}) \\ &= (A - KC)\hat{x} + Bu + Ky \\ &= A_c\hat{x} + Ky + Bu\end{aligned}$$

Feedback Control: $u = F\hat{x}$

Implementation:

$$\dot{x} = Ax + BF\hat{x} \quad \text{(Nature)}$$

$$\dot{\hat{x}} = (A - KC + BF)\hat{x} + Ky$$

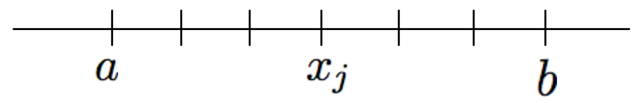
Issues:

- Estimator must be integrated in real time!
- Observations are available only at discrete times $y(t_j)$

Numerical Methods for BVP: Finite Differences

Problem:

$$y'' = p(x)y' + q(x)y + f(x) \ , \ a \le x \le b$$
$$y(a) = \alpha \ , \ y(b) = \beta$$



Grid: $x_j = a + jh \ , \ h = (b-a)/(N+1)$

Note: N interior grid points

Centered Difference Formulas: (From Taylor expansions)

$$y''(x_j) = \frac{1}{h^2}\left[y(x_{j+1}) - 2y(x_j) + y(x_{j-1})\right] - \frac{h^2}{24}y^{(4)}(\xi_j)$$
$$y'(x_j) = \frac{1}{2h}\left[y(x_{j+1}) - y(x_{j-1})\right] - \frac{h^2}{6}y'''(\eta_j)$$

System:

$$\begin{aligned}\frac{y(x_{j+1}) - 2y(x_j) + y(x_{j-1})}{h^2} &= p(x_j)\left[\frac{y(x_{j+1}) - y(x_{j-1})}{2h}\right] + q(x_j)y(x_j) \\ &\quad + f(x_j) - \frac{h^2}{12}\left[2p(x_j)y'''(\eta_j) - y^{(4)}(\xi_j)\right]\end{aligned}$$

Finite Difference Method for BVP

Finite Difference System: Define $y_0 = \alpha$, $y_{N+1} = \beta$ and consider

$$\left(\frac{2y_j - y_{j+1} - y_{j-1}}{h^2}\right) + p(x_j)\left(\frac{y_{j+1} - y_{j-1}}{2h}\right) + q(x_j)y_j = -f(x_j)$$

$$\Rightarrow -\left(1 + \frac{h}{2}p(x_j)\right)y_{j-1} + (2 + h^2 q(x_j))y_j - \left(1 - \frac{h}{2}p(x_j)\right)y_{j+1} = -h^2 f(x_j)$$

for $j = 1, 2, \cdots, N$

Matrix System:

$$\begin{bmatrix} 2+h^2q(x_1) & -1+\frac{h}{2}p(x_1) & 0 & \\ -1-\frac{h}{2}p(x_2) & 2+h^2q(x_2) & -1+\frac{h}{2}p(x_2) & \\ & \ddots & \ddots & \ddots \\ & & & -1+\frac{h}{2}p(x_{N-1}) \\ & 0 & -1-\frac{h}{2}p(x_N) & 2+h^2q(x_N) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} -h^2f(x_1) + \left(1+\frac{h}{2}p(x_1)\right)\alpha \\ -h^2 f(x_2) \\ \vdots \\ -h^2 f(x_{N-1}) \\ -h^2f(x_N) + \left(1-\frac{h}{2}p(x_N)\right)\beta \end{bmatrix}$$

Galerkin Methods

Boris Galerkin: 1871-1945; Mathematician and Engineer

Consider

$$Au = f$$

on inner product space $X, \langle \cdot, \cdot \rangle$. For finite dimensional spaces, $V^N = \text{span}\{\phi_1, \cdots, \phi_N\}$ and $Y^N = \text{span}\{\varphi_1, \cdots, \varphi_N\}$, find $u^N \in V^N$ that satisfies

$$\langle Au^N - f, \varphi_i \rangle = 0 \; , \; i = 1, \cdots, N$$

Employ

$$u^N = \sum_{j=1}^{N} u_j \phi_j$$

which yields

$$\sum_{j=1}^{N} \langle A\phi_j, \varphi_i \rangle \, u_j = \langle f, \varphi_i \rangle$$

for $i = 1, \cdots, N$.

Terminology:

- φ_i weight or test functions
- ϕ_j basis, trial or shape functions

Galerkin Methods

Rayleigh-Ritz: Take $\varphi_i = \phi_i$ so

$$\sum_{j=1}^{N} \langle A\phi_j, \phi_i \rangle \, u_j = \langle f, \phi_i \rangle$$

When A is symmetric and positive definite, this is the R-R method and solution is equivalent to that obtained by minimizing

$$J(u) = \langle Au, u \rangle - \langle f, u \rangle$$

with respect to $u \in V^N$.

Finite Element: Employ piecewise polynomials for the test and trial functions. Operator A does not have to be symmetric.

Least Squares: Take $\varphi_i = A\phi_i$ so

$$\sum_{j=1}^{N} \langle A\phi_j, A\phi_i \rangle \, u_j = \langle f, A\phi_i \rangle$$

Collocation: Take $\varphi_i(x) = \delta(x - x_i)$

Finite Element Method for BVP

Problem:

$$-\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) + q(x)y = f(x)\ ,\ 0 \le x \le \ell$$

$$y(0) = y(\ell) = 0$$

Assumptions: $p(x) \ge \delta > 0\ ,\ q(x) \ge 0$

Weak Formulation:

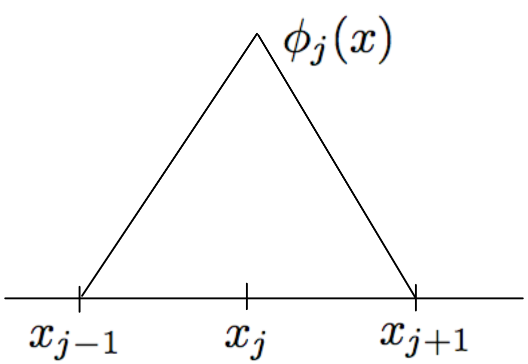
$$\int_0^\ell p\frac{dy}{dx}\frac{d\phi}{dx}dx + \int_0^\ell qy\phi dx = \int_0^\ell f\phi dx$$

for all $\phi \in H_0^1(0,\ell) = \{\phi \in H^1(0,\ell) \,|\, \phi(0) = \phi(\ell) = 0\}$

Grid: $x_j = jh\ ,\ h = \ell/N$

Linear Basis: $j = 1, \cdots, N-1$

$$\phi_j(x) = \frac{1}{h}\begin{cases} x - x_{j-1}\ , & x_{j-1} \le x < x_j \\ x_{j+1} - x\ , & x_j \le x \le x_{j+1} \\ 0\ , & \text{otherwise} \end{cases}$$



Finite Element Method for BVP

Approximate Solution:

$$y^N(x) = \sum_{j=1}^{N-1} y_j \phi_j(x)$$

System:

$$\sum_{j=1}^{N-1} y_j \left(\int_0^\ell p\phi_j'\phi_i' dx + \int_0^\ell q\phi_j\phi_i dx \right) = \int_0^\ell f\phi_i dx$$

for $i = 1, \cdots, N-1$

Matrix System: $A\vec{y} = \vec{f}$ where

$$\vec{y} = [y_1, \cdots, y_{N-1}]^T$$

$$[A]_{ij} = \int_0^\ell \left(p\phi_j'\phi_i' + q\phi_j\phi_i\right) dx$$

$$[\vec{f}]_i = \int_0^\ell f\phi_i dx$$

Integrals: Gaussian quadrature; e.g., 2 pt

$$x_1 = x_{j-1} + \frac{h}{2}\left(1 - 1/\sqrt{3}\right) \quad , \quad w_1 = \frac{h}{2}$$

$$x_2 = x_{j-1} + \frac{h}{2}\left(1 + 1/\sqrt{3}\right) \quad , \quad w_2 = \frac{h}{2}$$

Reference: Smith, Chapter 8

Error Estimates

Linear Splines:

$$\left|y(x) - y^N(x)\right| = \mathcal{O}\left(h^2\right)\ ,\ 0 \leq x \leq \ell$$

Cubic Splines:

$$\left|y(x) - y^N(x)\right| = \mathcal{O}\left(h^4\right)\ ,\ 0 \leq x \leq \ell$$

Finite Difference Versus Galerkin Methods

Galerkin (Finite Element) Advantageous:

- Model derived using energy principles
- Complicated geometries
- Natural boundary conditions
- Coupled systems or multiphysics problems
- Rigorous error analysis in various norms

Finite Difference Advantageous:

- Easier to program for certain problems
- Error analysis based on Taylor theory