Numerical Lecture #5: Systems of ODEs, Runge-Kutta, and Adaptive Stepsize Algorithms

[Reference: "Numerical Recipes" 3rd ed. Press et al. (CUP) Chap 17 - Integration of ODEs]

I. Systems of ODEs

A. Reducing to a Couple Set of First-order ODEs

1. In terms of position $\underset{\sim}{x}(t)$, the motion of a single particle is described by a second-order ODE:

$$\frac{d^2\underset{\sim}{x}}{dt^2} = \frac{q}{m}\left(\underset{\sim}{E} + \frac{d\underset{\sim}{x}}{dt} \times \underset{\sim}{B}\right)$$

a. Since this is a vector equation, this is a set of 3 coupled second-order ODEs.

2. Can be reduced to a set of 6 first-order ODEs.

a. Use an auxiliary variable. Usually, you just choose a derivative of $\underset{\sim}{x}$.

$$\underset{\sim}{V} \equiv \frac{d\underset{\sim}{x}}{dt}$$

b. This separates the second-order ODE into two first-order ODEs.

$$\frac{d\underset{\sim}{x}}{dt} = \underset{\sim}{V} \quad \leftarrow \text{6 first-order ODEs.}$$

$$\frac{d\underset{\sim}{V}}{dt} = \frac{q}{m}\left(\underset{\sim}{E} + \underset{\sim}{V} \times \underset{\sim}{B}\right)$$

3. Thus, the problem has the general form

$$\boxed{\frac{dy_i(t)}{dt} = f_i(t, y_1, \ldots y_N) \qquad i = 1, \ldots, N}$$

I. A.3 (Continued)

a. This generalized form is useful in implementing numerical schemes.

II. General Approaches for ODE Integration Numerically

A. Three Major Types of Methods

1. Runge-Kutta Methods: Combine Euler-type steps to achieve higher-order convergence.

2. Richardson Extrapolation / Bulirsch-Stoer Methods: Extrapolate result to value that would be obtained for a much smaller step size.

3. Predictor-Corrector Methods: Use multiple steps, storing solution along the way, extrapolating to full step and correcting extrapolation by derivative information at the new point.

B. When to Use

1. Runge-Kutta is a workhorse:
   a. It almost always works
   b. But, it is not as efficient. Only competitive when evaluating the derivative is cheap
   c. Only moderate accuracy ($\sim 10^{-5}$)

2. Predictor-corrector has high overhead
   a. Good when evaluating derivative is expensive
   b. Most computationally efficient for smooth problems.

II. B. (Continued)

3. Burlisch-Stoer
   a. Most efficient method for most applications
   b. Yields high accuracy solutions
   c. Can fail for solutions that are stiff / not particularly smooth

III. Runge-Kutta Methods

A. Fundamental Concept

1. Consider a differential equation $\frac{dy}{dt} = f(t, y)$

2. Euler Method: $y_{n+1} = y_n + \Delta t\, f(t_n, y_n)$
   a. Error $\sim \mathcal{O}(\Delta t^2)$

3. Basic Idea:
   a. Take a "trial" step to midpoint of interval
   b. Use y and t at midpoint to compute step across whole interval

4. Implementation

   a. $\Delta y_n^{(1)} = \Delta t\, f(t_n, y_n)$

   b. $\Delta y_n^{(2)} = \Delta t\, f\left(\underbrace{t_n + \frac{\Delta t}{2}}_{= t_{n+\frac{1}{2}}}, \underbrace{y_n + \frac{\Delta y_n^{(1)}}{2}}_{= y_{n+\frac{1}{2}}}\right)$

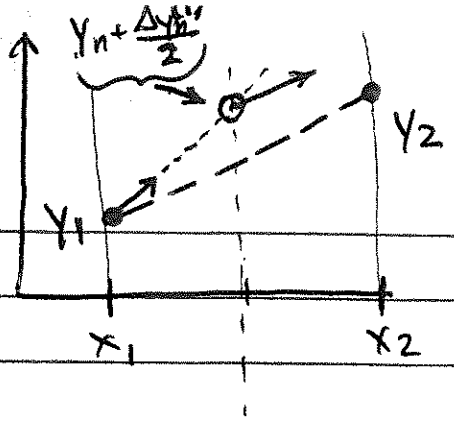
   c. $y_{n+1} = y_n + \Delta y_n^{(2)} + \mathcal{O}(\Delta t^3)$

   (Second-order Runge-Kutta Method, or Midpoint Method)

d. NOTE: This is similar to initializing Leapfrog using an Euler Step!

III. A.

5. Diagram of 2nd-Order Runge-Kutta



6. Using additional intermediate steps, you can combine each estimate with appropriate coefficients to eliminate error terms order by order

B. Fourth-Order Runge-Kutta

1. Mostly widely use, higher-order Runge-Kutta Method.

2. Method

$$k_1 = \Delta t\, f(t_n, y_n)$$

$$k_2 = \Delta t\, f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = \Delta t\, f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = \Delta t\, f\left(t_n + \Delta t, y_n + k_3\right)$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + \mathcal{O}(\Delta t^5)$$

Fourth-Order Runge-Kutta Method

RK4

3. This method requires four computations of the derivative, so it works well if this computation is cheap

III. B. (Continued)

4. Note that this fourth order method does not require a lower order initialization. Such initialization is effectively built into the scheme. Thus, it is well-suited to use with Adaptive Stepsize control.

IV. Adaptive Stepsize Control

A. 1. Numerical Integration Methods can be implemented in a way to monitor internal consistency.
2. Numerical errors can be kept within specified thresholds by using automatic, adaptive changing of the stepsize Δt.
3. For simple ODE integration, it is recommend to always use adaptive stepsize control.
4. Most numerical simulation of plasmas requires integration of PDEs (not ODEs), so numerical stability is an additional concern that may necessitate stepsize control

B. Implementation

1. Goal: Achieve a desired level of accuracy with minimum computational effort
a. Scheme tiptoes through highly structured regions, but takes great strides through smooth regions
b. Gains in efficiency can be factors of 10, 100, or more!
c. Stepsize can also be monitored with a conserved quantity, eg. energy

IV. B. (Continued)

2. Key: Algorithm must return an estimate of the truncation error

3. Technique: Step Doubling

a. Take one full step of size $2\Delta t$ (RK4)

b. Take two half steps of size Δt (RK4)

c. Ost:

i)

Four evaluations of derivative

ii)

Already have initial evaluation

Seven additional evaluations of derivative

$\Rightarrow$ Total: 11 evaluations

d. This must be compared to 8 evaluations for two Δt steps.

$\Rightarrow$ Cost $\frac{11}{8} = 1.375 \rightarrow$ 37.5% more costly.

4. What do we get?

a. Take $y(t+2\Delta t)$ as exact solution

b. For RK4, approximate solutions y_1 (one step $2\Delta t$) and y_2 (two steps Δt) have error

$$y(t+2\Delta t) = y_1 + (2\Delta t)^5 \phi + O(\Delta t^6)$$

$$y(t+2\Delta t) = y_2 + 2(\Delta t)^5 \phi + O(\Delta t^6)$$

Factor of two because two steps with error order Δt^5

Largest order error where $\phi \sim O\left(\frac{1}{5!}\frac{\partial^5 y}{\partial t^5}\Big|_t\right)$

IV. B. (Continued)

4. (Continued)

c. Estimate of magnitude of truncation error

$$\Delta \equiv y_2 - y_1$$

d. Adjusting Δt can be done to keep Δ lower than some threshold (and also not too small)

5. We can also combine y_1 and y_2 to eliminate 4th order error!

a. $y(t+2\Delta t) = y_2 + \frac{\Delta}{15} + O(h^6)$

$\Rightarrow$ The solution $y_2 + \frac{\Delta}{15}$ is 5th-order.

b. BUT, we have no way of estimating the truncation error of this 5th-order estimate.

6. Alternative: Embedded Runge-Kutta formulas

a. Instead of step doubling, you can combine different solutions with the same function evaluations to obtain both a 4th & 5th order estimate.

b. This enables truncation error of 4th-order to be estimated.

c. Implementations: (i) Fehlberg (ii) Cash-Karp

7. All of these schemes are deemed RK45 (4th-order Runge-Kutta with 5th-order correction)

IV. (Continued)

C. Step size Adjustment

1. Use local extrapolation of error.
2. Target error $\equiv \Delta_0$
3. Since $\Delta \propto \Delta t^5$, we can scale Δt to yield error of order Δ_0

$$\Rightarrow \quad \Delta t_{new} = \Delta t \left| \frac{\Delta_0}{\Delta} \right|^{\frac{1}{5}}$$

4. a. If $\frac{\Delta_0}{\Delta} < 1$, stepsize is reduced (smaller step to reduce error)

b. If $\frac{\Delta_0}{\Delta} > 1$, stepsize is increasing (bigger step to increase error)

5. Note: For a dimensionless implementation of equations, Δ will have an order of magnitude scaled to y.

a. If we expect $y \sim O(1)$, and we want a solution accurate to 1 part in $10^6 \rightarrow \Delta \sim 10^{-6}$, one can simply use Δ.

b. But, if different fields (y_1, y_2, e.g. x_1, v_1) have different magnitudes, we may want to scale

$$\frac{\Delta}{y_*} \leftarrow \text{fractional error,} \quad \text{where } y_* \text{ is a typical order of magnitude} \rightarrow y_* = |y_n|$$

c. This works unless y passes through zero.
$\Rightarrow$ Then you may want to use something else

$$y_* = |y_n| + \left| \Delta t \frac{dy}{dt}\Big|_{tn} \right|$$

V. Matlab Function ode45

A. Implementation

1. Matlab Function ode45 is a particle integrator using an embedded RK45 scheme (Dormand-Prince).

2. Syntax:

$$[t, y] = ode45(odefun, tspan, y_0)$$

a. t is column vector of times

b. y is matrix, each row is $[x\ y\ z\ v_x\ v_y\ v_z]$ at a specific time

c. $tspan = [t_i ; t_f]$ (column vector)

d. $y_0 = [x_0\ y_0\ z_0\ v_{x0}\ v_{y0}\ v_{z0}]$

e. odefun is a function handle

eg. lorhandle = @ lorentz;

where lorentz.m contains the function that computes derivatives.

3. Use formulation $\frac{dy_i}{dt} = f(t, y_1, \ldots y_6) \quad i = 1, 3 \ldots, 6$

where $y_1 = x$, $y_2 = y$, $y_3 = z$, $y_4 = v_x$, $y_5 = v_y$, $y_6 = v_z$

$\Rightarrow$ Function contains derivatives for each variable.