Ordinary Differential Equations

Basics

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

References

- S. Chapra and R. Canale, Numerical Methods for Engineers
- Carnahan and Wilkes, Applied Numerical Methods, University of Michigan, Class Notes, 1996.
- C. Moler, Numerical Computing with Matlab, available at `http://www.mathworks.com/moler/chapters.html`
- Pal, Numerical Analysis for Scientists and Engineers, 2007.
- M. Heath, Scientific Computing: An Introductory Survey
- wikipedia.org

Contents

Motivation

- Many physical systems change with time
- Examples: satellites orbiting, population dynamics, weather/climate, material degradation etc.
- Differential equations provide the language to describe *continuous* change.
- Ex: State of a system at time t be $\mathbf{y}(t)$, then differential equations tell us about the change in $\mathbf{y}(t)$ by describing the relationships between its time-derivatives.
- Newton's law:
$$m\frac{d^2\mathbf{y}(t)}{dt^2} = F(\mathbf{y}(t))$$
- Fourier's Heat Law
$$q = -k\frac{dT}{dx}$$

Classification of ODEs

- Explicit Differential Equation (Quadrature)

$$y' = \frac{dy(t)}{dt} = f(t).$$

- General Explicit Differential Equation

$$y' = f(y, t).$$

  This category subsumes "quadrature"-type ODEs

- Implicit Differential Equation

$$F(t, y, y') = 0.$$

  We will not consider these in this class

Classification of ODEs

- Order: Based on order of the highest derivative

$$\frac{d^2y}{dt^2} + y\frac{dy}{dt} = \sin(t)$$

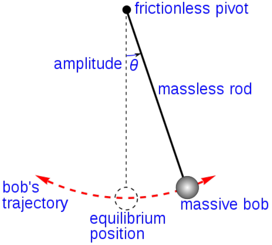
  Second order nonlinear ODE
- Linear ODE

$$a_3(t)\frac{d^3y}{dt^3} + a_2(t)\frac{d^2y}{dt^2} + a_1(t)\frac{dy}{dt} + a_0(t)y = g(t)$$

  Third order linear ODE (no terms like $y^{(n)}y^{(m)}$). If $g(t) = 0$ it is also called a homogeneous ODE.
- Linearization is the conversion of a nonlinear equation to an approximate linear equation. This is a very common practice in the applied sciences.

Example: Linearization

Consider a pendulum swinging under the action of gravity*



Newton's second law yields the equation

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0, \qquad \text{(order 2, nonlinear)}$$

*wikipedia.org

Example: Linearization

- If we restrict analysis to small θ, then $\sin\theta \approx \theta$.

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\theta = 0, \qquad \text{(order 2, linear)}$$

- Practically linear ODEs are nice, because they can be solved analytically
- For example, the equation above

$$\theta(t) = \theta_0 \cos\left(\sqrt{\frac{g}{\ell}}\, t\right),$$

  where $\theta_0 = \theta(t=0)$ is called the initial condition
- Linearization cannot always be invoked. Often we are interested in nonlinear response: what happens at large amplitudes?

Classification of ODEs

- System of n equations:

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_n' \end{bmatrix} = \begin{bmatrix} f_1(t, \mathbf{y}) \\ f_2(t, \mathbf{y}) \\ \vdots \\ f_n(t, \mathbf{y}) \end{bmatrix}$$

- Shorthand:

$$\boxed{\mathbf{y}' = \mathbf{f}(t, \mathbf{y}).}$$

- *Linear* if

$$\mathbf{f}(t, \mathbf{y}) = \mathbf{A}(t)\mathbf{y} + \mathbf{B}(t).$$

- *Homogeneous* if $\mathbf{B}(t) = 0$ above.

Order of ODEs

- Higher order ODEs can be reduced to a system of first order ODEs Example:

$$y''(t) + y(t) = 0$$

$$\begin{aligned} y(0) &= 0 \\ y'(0) &= 0 \end{aligned}$$

- Set $z = y'$

$$z'(t) + y = 0$$

$$y'(t) - z = 0$$

$$z(0) = 0; \quad y(0) = 0$$

- In general a N-order ODE can be reduced to a system of N, first order ODEs

IVP and BVP

- Based on boundary conditions for the same ODE.
- Initial Value Problems (IVP)

$$\frac{d^2y(t)}{dt^2} + y(t) = 0$$

$$y(0) = 0;\ y'(0) = 2.$$

- Boundary Value Problems (BVP)

$$\frac{d^2y(t)}{dt^2} + y(t) = 0$$

$$y(0) = 0;\ y(\pi/2) = 2.$$

- We will look at IVPs initially, and then consider some simple methods for solving BVPs

Numerical Solution: Basic Idea

- Consider an explicit scalar IVP

$$y'(t) = f(t, y) \quad y(0) = y_0$$

- Discretize t domain $\{t_0, t_1, t_2, ..., t_n, ...\}$
- Want to compute approximate numerical solution on the discretized domain

$$y_n = y(t_n) \approx Y_n = Y(t_n)$$

  where $Y(t)$ is the analytical solution to the IVP
- We will consider the family of Runge-Kutta (RK) methods, starting with its simplest member: Euler method.

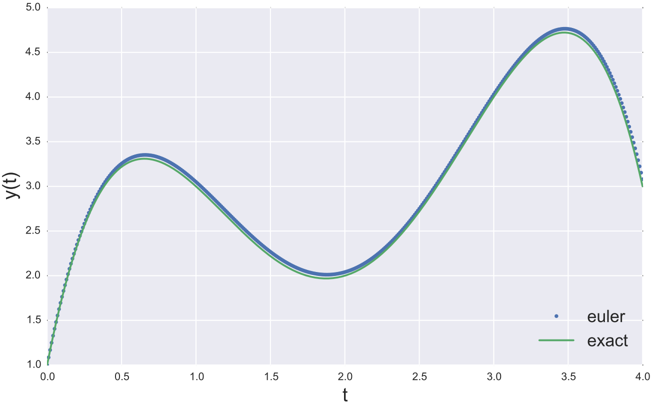
Euler

- ODE
$$y'(t) = f(t, y) \quad y(0) = y_0$$
- Discretize first derivative with time step $h = t_n - t_{n-1}$
$$y' = \frac{y_n - y_{n-1}}{h} = f(t_{n-1}, y_{n-1}) = f_{n-1}$$
- Rearranging, we get a time-marching algorithm
$$\boxed{y_n = y_{n-1} + hf_{n-1}}$$
- Let us solve a simple example with this method

Euler Example

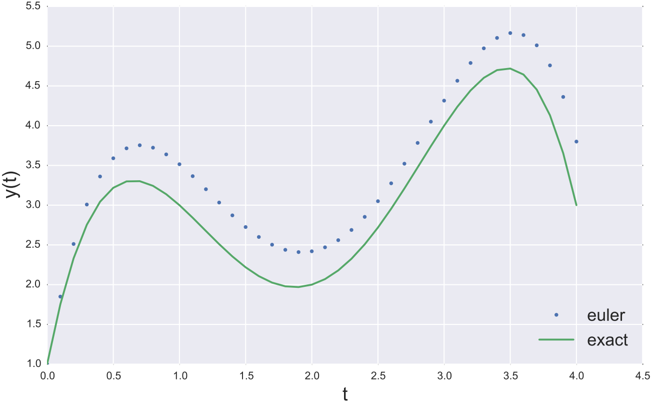
- Consider, with $h = 0.01$

$$f(t, y) = -2t^3 + 12t^2 - 20t + 8.5; \quad y(0) = 1$$



Euler Example

- With $h = 0.1$



Lessons

- This particular example was really a quadrature problem ($f(t, y) = f(t)$ alone)
- What Euler did was equivalent to a simple "rectangle" rule in integration.

$$I_n = y_n - y_0 = h \sum_{i=0}^{n-1} f(t_i)$$

- Many ODE algorithms map on to integration algorithms when we consider the special case of $f(y, t) = f(t)$, in the ODE $y' = f(t, y)$.
- As we will show shortly, this is an $\mathcal{O}(h)$ method.
- Need to develop language/results to analyze the numerical method

Error Analysis

- Usual suspects are (i) rounding error and (ii) truncation error
- Rounding error is practically not as important for ODEs as truncation error. Hence we focus primarily on truncation error (also called discretization error)
- Two metrics to analyze truncation error of a numerical method:
  - Local Error: Error incurred over a single step
  - Global Error: Cumulative error over all the previous steps
- Analogy
  - LE : GE :: semester-GPA : cumulative-GPA
  - LE : GE :: income : wealth

Global Error

- Let y_n be the numerical solution computed at t_n
- $Y(t)$ represent the exact solution to the IVP passing through the initial condition (t_0, y_0)
- The global error is given by:

$$\epsilon_n^G = |y_n - Y(t_n)|$$

Local Error

- It is the error incurred over *one step* of the method

$$\epsilon_n^L = |y_n - U_{n-1}(t_i)|$$

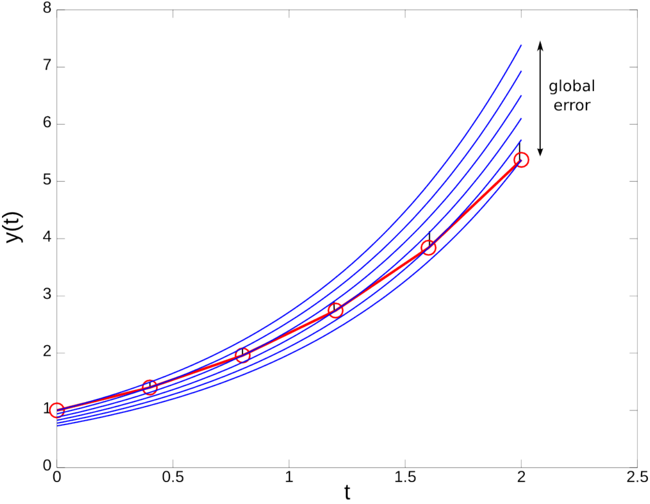
  where $U_{n-1}(t)$ is the *exact* solution of the ODE passing through the previous point (t_{n-1}, y_{n-1})
- That is, $U_{n-1}(t)$ is an exact solution to a perturbed initial condition such that it passes through (t_{n-1}, y_{n-1}).
- These definitions are more easily understood by considering an example. Let us visualize them by considering the IVP:

$$y' = \lambda y, \quad y(0) = 1$$

  with $\lambda = 1$.

Example with Euler's Method

- Euler's method with $h = 0.4$, and $y_0 = 1$.



Error Analysis

- Generally, we care only about the global error, but only local error can be readily estimated and controlled
- Thus, we care about the relationship between local and global error
- The global error is not simply the sum of the local errors.
- It may be larger or smaller, depending on whether the solutions of the ODE are converging or diverging in the domain of interest.
- We expect both the errors to decrease as the step-size is decreased. We would also like to know how quickly the error vanishes.
- Let us consider the local error of Euler's method for a simple scale ODE $y' = f(t, y)$.

Forward Euler: Error

- Taylor expansion of $y(t)$ around t_{n-1}

$$y_n = y_{n-1} + hy'(t_{n-1}) + \frac{h^2}{2}y''(\xi)$$
$$y_n = y_{n-1} + hf_{n-1} + \mathcal{O}(h^2)$$

- Thus, the local truncation error of Euler's method is second order

$$y_n = y_{n-1} + hf_{n-1} + \mathcal{O}(h^2)$$

- It can be shown that the global error for Euler's method is $\epsilon_G = \mathcal{O}(h)$
- Euler's method is perfect, if the exact solution to the IVP $Y(t)$ linear. Hence, it is called a first order method.

Error Analysis

- This pattern is true for higher (n^{th}) order 1-step methods
  If
  $$\epsilon_L \sim \mathcal{O}(h^{n+1}) \quad \implies \quad \epsilon_G \sim \mathcal{O}(h^n).$$
- We can derive higher order methods based on the Taylor series, by expanding of y_i around (t_{i-1}, y_{i-1}).

$$y_i = y_{i-1} + h \underbrace{y'_{i-1}}_{f_{i-1}} + \frac{h^2}{2!} y''_{i-1} + \ldots + \frac{h^n}{n!} y^{(n)}_{i-1} + \mathcal{O}(h^{n+1})$$

- Recall, by chain rule

$$y''(t) = \frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$$

$$y''(t) = \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y}$$

Higher Order Taylor Series Methods

- A second order method would require us to evaluate and specify the derivatives

$$\frac{\partial f}{\partial t} \text{ and } \frac{\partial f}{\partial y}$$

- Higher order derivatives of y are even more complicated to evaluate. For example

$$f''(t,y) = \frac{\partial}{\partial t}\left(\frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y}\right) + f\frac{\partial}{\partial y}\left(\frac{\partial f}{\partial t} + f\frac{\partial f}{\partial y}\right)$$

- Thus, this is not a popular route. Instead, in Runge - Kutta methods we achieve the performance of higher order Taylor formulas, while avoiding the computation of higher order derivatives.

Simple RK Methods

- Euler is the simplest example of an RK method, which we will consider more generally later
- We consider three more RK methods, which have intuitive appeal, before generalizing: backward Euler, the midpoint and Heun's methods
- **Backward Euler**: implicit method
- **Midpoint Method**: multistage method
- **Heun's Method**: predictor-corrector
- Backward Euler is $\mathcal{O}(h)$, while the remaining two are explicit $\mathcal{O}(h^2)$ methods.

Backward Euler

- Algorithm

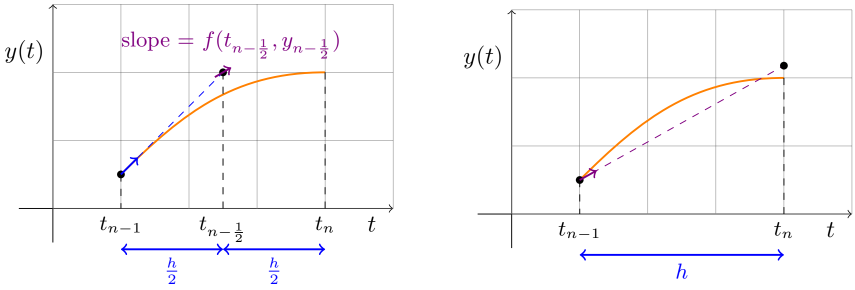
$$y_n = y_{n-1} + hf_n$$

- Implicit method
- Need to solve a potentially nonlinear equation at each time step to get y_n
- $\mathcal{O}(h)$ convergent, like forward Euler
- Better stability

Midpoint Method

Sometimes called “modified Euler”

$$\begin{aligned} y_{n-1/2} &= y_{n-1} + \frac{h}{2} f(t_{n-1}, y_{n-1}) \\ y_n &= y_{n-1} + h f(t_{n-1/2}, y_{n-1/2}) \end{aligned}$$



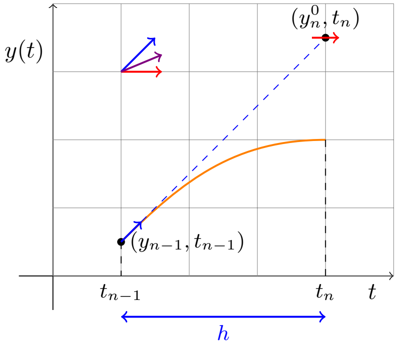
Heun's Method

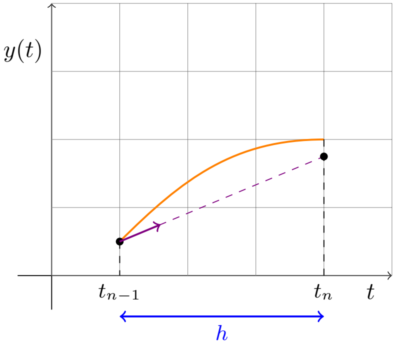
Predictor:

$$y_n^0 = y_{n-1} + hf(t_{n-1}, y_{n-1})$$

Corrector:

$$y_n = y_{n-1} + h\frac{f(t_{n-1}, y_{n-1}) + f(t_n, y_n^0)}{2}$$





It is possible to apply the corrector repeatedly.

Example Problem

Solve the nonlinear second-order equation for a pendulum,

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0,$$

where $g = 9.8$ m/s^2, and $l = 1$ meter, using the mid-point method:

$$\begin{aligned} y_{n-1/2} &= y_{n-1} + \frac{h}{2} f(t_{n-1}, y_{n-1}) \\ y_n &= y_{n-1} + h f(t_{n-1/2}, y_{n-1/2}) \end{aligned}$$

The initial conditions are $\theta(0) = \pi/3$, and $\theta'(0) = 0$.

Use $\Delta t = 0.05$, and find $\theta(t)$, for $0 \le t \le 3.0$.

Step 1: Write as First Order Equations

- Set $v = d\theta/dt$, and hence we have:

$$\begin{aligned} \frac{d\theta}{dt} &= v \\ \frac{dv}{dt} &= -\frac{g}{l}\sin\theta) \end{aligned}$$

- Now we can think of:

$$y = \begin{bmatrix} \theta \\ v \end{bmatrix},$$

and

$$\frac{d}{dt}\begin{bmatrix} \theta \\ v \end{bmatrix} = \begin{bmatrix} v \\ -(g/l)\sin\theta \end{bmatrix}$$

Step 2: Write Down Function Code

We write down stuff in standard form:

```
%
% Define the function yp = f(t, y);
% For this function "t" is not really required!
%
function yp = func(t, y)

  yp = zeros(size(y));

  g = 9.8; % m/s^2
  l = 1;   % meters

  theta = y(1); % I made these definitions only for clarity
  v     = y(2); % can directly use y(1), y(2) below.

  yp(1) = v;           % dtheta/dt
  yp(2) = -g/l * sin(theta);  % dv/dt

end
```

Step 3: Write Down Method Code

We write down one step of midpoint method:

```
%
% One Step of Mid-point Method
%
function ynew = midpoint(f, yold, h, t)

  ymid = yold + h/2 * f(t, yold);
  ynew = yold + h * f(t + h/2, ymid);

end
```

We pass the name of the function (via a function handle).

Finally, we write the "driver" routine.

Step 4: Driver Code

Here we set up the problem, to use the midpoint method.

```
% Slice up time, and preallocate arrays
t0   = 0.; Tmax = 3.0; dt   = 0.05;
npts = Tmax/dt + 1;

t     = zeros(npts, 1);
theta = zeros(npts, 1);
v     = zeros(npts, 1);

% Initial conditions
t(1) = 0.; theta(1) = pi/3; v(1) = 0.;

% Time-Stepping
for i = 2:npts

  yold = [theta(i-1); v(i-1)];
  ynew = midpoint(@func, yold, dt, t(i-1)); % function handle here!

  t(i)     = t(i-1) + dt;
  theta(i) = ynew(1);
  v(i)     = ynew(2);

end
```

Step 5: Plotting Etc.

This can be added to the driver code.

```
plot(t, theta, 'go-'); hold on;
plot(t, v, 'b--');
legend('theta','v')
xlabel('t')
ylabel('theta, v')
hold off;
```