UW–Madison Math/CS 714

Methods of Computational Mathematics I

Initial value problems I

Instructor: Yue Sun (yue.sun@wisc.edu)

September 30, 2025

Integration of ODE Initial Value Problems

We consider problems of the form

$$u'(t) = f(u,t), \quad u(t_0) = \eta.$$

Here $u(t) \in \mathbb{R}^s$ and $f : \mathbb{R} \times \mathbb{R}^s \to \mathbb{R}^s$.

Writing this system out in full, we have

$$u'(t) = \begin{bmatrix} u_1'(t) \\ u_2'(t) \\ \vdots \\ u_s'(t) \end{bmatrix} = \begin{bmatrix} f_1(u,t) \\ f_2(u,t) \\ \vdots \\ f_s(u,t) \end{bmatrix} = f(u(t),t).$$

This is a system of s coupled ODEs for the variables $u_1, u_2, \ldots, u_s$.

ODE IVPs

"Initial value problem" implies that we know $u(t_0)$, *i.e.* $u(t_0) = \eta \in \mathbb{R}^s$ is the initial condition.

The order of an ODE is the highest-order derivative that appears.

Hence $u'(t) = f(u, t)$ is a first-order ODE system.

Higher order ODEs can be rewritten as first-order ODE systems. Hence we focus on developing methods for first-order ODEs.

ODE IVPs

See chapter 5, sections 5.1 to 5.7, for background on this section, which cover

- Existence and uniqueness of solutions
- Lipschitz constant
- One-step methods (forward/backward Euler)
- Truncation error
- Runge–Kutta methods

These concepts were covered in the ODE introduction—see video under the Kaltura tab on Canvas.

Example: integrating the Rössler ODE system

Assume $t_0 = 0$ and define $t_n = nk$ where k is a grid spacing. We use k for a spacing in time, to distinguish it from h in space.

Let U^n be the numerical approximation of $u(t_n)$, and set $U^0 = \eta$. A finite-difference approximation of $u'(t) = f(u(t), t)$ gives

$$\frac{U^{n+1} - U^n}{k} = f(U^n, t_n)$$

which becomes

$$U^{n+1} = U^n + kf(U^n, t_n),$$

which is the forward Euler method. It is an explicit method, since U^{n+1} can be directly computed from U^n.

Example: integrating the Rössler ODE system

An alternative approach is to note from the fundamental theorem of calculus that

$$u(t_{n+1}) - u(t_n) = \int_{t_n}^{t_{n+1}} u'(t)dt = \int_{t_n}^{t_{n+1}} f(u(t), t)dt.$$

Approximating the integral using the trapezoid rule gives the trapezoid method,

$$U^{n+1} - U^n = \frac{k\left[f(U^n, t_n) + f(U^{n+1}, t_{n+1})\right]}{2}$$

The trapezoid method is an implicit method, since in general, root-finding must be used to find U_{n+1}.

Example: integrating the Rössler ODE system

The Rössler attractor is a chaotic ODE system with three coupled equations

$$x' = -y - z,$$
$$y' = x + ay,$$
$$z' = b + z(x - c).$$

Consider solving this system using the trapezoid method, with $u(t) = (x(t), y(t), z(t))$. To evaluate U^{n+1}, we must find a root of the equation

$$F(U) = 2(U^n - U) + k\left[f(U^n, t_n) + f(U, t_{n+1})\right].$$

Example: integrating the Rössler ODE system

In component form, writing $U = (X, Y, Z)$ and $U^n = (X^n, Y^n, Z^n)$,

$$F(U) = \begin{pmatrix} 2(X^n - X) + k\left[-Y^n - Z^n - Y - Z\right] \\ 2(Y^n - Y) + k\left[X^n + aY^n + X + aY\right] \\ 2(Z^n - Z) + k\left[2b + Z^n(X^n - c) + Z(X - c)\right] \end{pmatrix},$$

and the Jacobian is

$$J_F(U) = \begin{pmatrix} -2 & -k & -k \\ k & -2 + ak & 0 \\ kZ & 0 & -2 + k(X - c) \end{pmatrix}$$

The program *rossler_trap.py* solves the Rössler system, using Newton's method to perform the root-finding at each step.

Multistep methods

Both the Euler method and the trapezoid method are examples of one-step methods: the value of U^{n+1} is determined solely from U^n.

More generally we can look at multistep methods that incorporate information from previous steps. An r-step multistep method has the form

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = k \sum_{j=0}^{r} \beta_j f(U^{n+j}, t_{n+j}),$$

where $\alpha_r \neq 0$. If $\beta_r = 0$ then the method is explicit; otherwise it is implicit.

Multistep method example

The general form is

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = k \sum_{j=0}^{r} \beta_j f(U^{n+j}, t_{n+j}).$$

The trapezoid method is consistent with this form when $r = 1$ and

$$(\alpha_0, \alpha_1) = (1, -1), \qquad (\beta_0, \beta_1) = (\tfrac{1}{2}, \tfrac{1}{2}).$$

This gives

$$U^{n+1} - U^n = \frac{k\left[f(U^n, t_n) + f(U^{n+1}, t_{n+1})\right]}{2}$$

as before.

Families of multistep methods

Several different families of multistep methods can be derived using different approaches.

One approach is to start from the integral relation

$$u(t_{n+r}) = u(t_{n+r-1}) + \int_{t_{n+r-1}}^{t_{n+r}} u'(t)dt = u(t_{n+r-1}) + \int_{t_{n+r-1}}^{t_{n+r}} f(u(t), t)dt$$

Then

$$U^{n+r} = U^{n+r-1} + \int_{t_{n+r-1}}^{t_{n+r}} p(t)dt$$

where $p(t)$ is a polynomial interpolant of f using several prior values of U^{n+j}. (See derivation.)

Adams–Bashforth methods

The first few explicit Adams–Bashforth methods are

$$
\begin{aligned}
&\text{1 step:} \quad U^{n+1} = U^n + kf(U^n),\\
&\text{2 step:} \quad U^{n+2} = U^{n+1} + \frac{k}{2}(-f(U^n) + 3f(U^{n+1})),\\
&\text{3 step:} \quad U^{n+3} = U^{n+2} + \frac{k}{12}(5f(U^n) - 16f(U^{n+1}) + 23f(U^{n+2})).
\end{aligned}
$$

Adams–Moulton methods

The first few implicit Adams–Moulton methods are

$$
\begin{aligned}
\text{1 step:}\quad & U^{n+1} = U^n + \frac{k}{2}(f(U^n) + f(U^{n+1})),\\
\text{2 step:}\quad & U^{n+2} = U^{n+1} + \frac{k}{12}(-f(U^n) + 8f(U^{n+1}) + 5f(U^{n+2})),\\
\text{3 step:}\quad & U^{n+3} = U^{n+2} + \frac{k}{24}(f(U^n) - 5f(U^{n+1})\\
& \qquad\qquad + 19f(U^{n+2}) + 9f(U^{n+3})).
\end{aligned}
$$

Local truncation error

The local truncation error is given by

$$\tau(t_{n+r}) = \frac{1}{k}\left(\sum_{j=0}^{r} \alpha_j u(t_{n+j}) - k\sum_{j=0}^{r} \beta_j u'(t_{n+j})\right).$$

Using Taylor series

$$u(t_{n+j}) = u(t_n) + jku'(t_n) + \frac{(jk)^2}{2}u''(t_n) + \dots,$$
$$u'(t_{n+j}) = u'(t_n) + jku''(t_n) + \frac{(jk)^2}{2}u'''(t_n) + \dots.$$

Local truncation error

Hence

$$\tau(t_{n+r}) = \frac{1}{k}\left(\sum_{j=0}^{r}\alpha_j\right)u(t_n) + \left(\sum_{j=0}^{r}(j\alpha_j - \beta_j)\right)u'(t_n)$$
$$+ k\left(\sum_{j=0}^{r}\left(\frac{j^2}{2}\alpha_j - j\beta_j\right)\right)u''(t_n) + \dots$$

The method is consistent if $\tau \to 0$ and $k \to 0$, which occurs when

$$\sum_{j=0}^{r}\alpha_j = 0, \qquad \sum_{j=0}^{r} j\alpha_j = \sum_{j=0}^{r}\beta_j.$$

If the first $p+1$ terms in $\tau(t_{n+r})$ vanish, then the method is pth order accurate.

Characteristic polynomials

The characteristic polynomials of a multistep method are

$$\rho(\zeta) = \sum_{j=0}^{r} \alpha_j \zeta^j, \qquad \sigma(\zeta) = \sum_{j=0}^{r} \beta_j \zeta^j$$

The consistency conditions are equivalent to

$$\rho(1) = 0, \qquad \rho'(1) = \sigma(1)$$

We will see that the characteristic polynomials are also useful for determining stability of a method.

Multistep versus one-step methods

Multistep methods are attractive because they can achieve higher-order accuracy for limited extra work.

To measure the work required for a method, we can count the number of times that f will be evaluated.

For a large ODE system, evaluating f may become very expensive, and therefore the total f evaluations becomes a good proxy for the total workload.

Multistep versus one-step methods

Compare two methods:

- Classical fourth-order Runge–Kutta method – requires four intermediate stages to step from U^n to U^{n+1}. Four total f evaluations per timestep.
- Four-step Adams–Bashforth method – computing U^{n+4} requires evaluating $f(U^{n+3})$, and reusing $f(U^{n+2})$, $f(U^{n+1})$, and $f(U^n)$. One total f evaluation per timestep.

This is not an exact comparison, since it depends on the step size required to achieve a certain accuracy, but it suggests that multistep methods have some inherent advantages.

Multistep versus one-step methods

Multistep methods have some disadvantages:

- Starting values – for an r-step method, the values of $U^0, U^1, \ldots, U^{r-1}$ are required to begin. Can use an exact solution, or use a one-step method to begin with.
- Even timestep spacing – the derived formulae rely on the timestep size k being equal. By contrast, one-step methods allow the timestep to be adaptively chosen.
- Stability – Multistep methods introduce additional stability considerations that are not present for one-step methods.

Digression: Runge–Kutta methods

Runge–Kutta (RK) methods are another type of one-step discretization, a very popular choice.

Aim to achieve higher order accuracy by combining evaluations of f (*i.e.* estimates of y') at several points in $[t_k, t_{k+1}]$.

RK methods all fit within a general framework, which can be described in terms of Butcher tableaus.

We will first consider two RK examples: two evaluations of f and four evaluations of f.

Digression: Runge–Kutta methods

The family of Runge–Kutta methods with two intermediate evaluations is defined by

$$y_{k+1} = y_k + h(ak_1 + bk_2),$$

where $k_1 = f(t_k, y_k)$, $k_2 = f(t_k + \alpha h, y_k + \beta h k_1)$.

The Euler method is a member of this family, with $a = 1$ and $b = 0$.

By careful analysis of the truncation error,[1] it can be shown that we can choose a, b, α, β to obtain a second-order method.

[1] See order condition notes.

Digression: Runge–Kutta methods

[*order2.py*] Three such examples are:

- The modified Euler method ($a = 0$, $b = 1$, $\alpha = \beta = 1/2$):

$$y_{k+1} = y_k + hf\left(t_k + \frac{1}{2}h, y_k + \frac{1}{2}hf(t_k, y_k)\right).$$

- The improved Euler method (or Heun's method) ($a = b = 1/2$, $\alpha = \beta = 1$):

$$y_{k+1} = y_k + \frac{1}{2}h[f(t_k, y_k) + f(t_k + h, y_k + hf(t_k, y_k))].$$

- Ralston's method ($a = 1/4$, $b = 3/4$, $\alpha = 2/3$, $\beta = 2/3$)

$$y_{k+1} = y_k + \frac{1}{4}h[f(t_k, y_k) + 3f(t_k + \tfrac{2h}{3}, y_k + \tfrac{2h}{3}f(t_k, y_k))].$$

Digression: Runge–Kutta methods

The most famous Runge–Kutta method is the "classical fourth-order method", RK4:

$$y_{k+1} = y_k + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$\begin{aligned}
k_1 &= f(t_k, y_k), \\
k_2 &= f(t_k + h/2, y_k + hk_1/2), \\
k_3 &= f(t_k + h/2, y_k + hk_2/2), \\
k_4 &= f(t_k + h, y_k + hk_3).
\end{aligned}$$

Analysis of the truncation error in this case (which gets quite messy!) gives $T_k = O(h^4)$.

Digression: Butcher tableau

Summarizes an $s+1$ stage Runge–Kutta method using a triangular grid of coefficients

$$
\begin{array}{c|ccccc}
\alpha_0 & & & & & \\
\alpha_1 & \beta_{1,0} & & & & \\
\vdots & \vdots & & & & \\
\alpha_s & \beta_{s,0} & \beta_{s,1} & \cdots & \beta_{s,s-1} & \\
\hline
 & \gamma_0 & \gamma_1 & \cdots & \gamma_{s-1} & \gamma_s
\end{array}
$$

The ith intermediate step is

$$f\left(t_k + \alpha_i h, y_k + h \sum_{j=0}^{i-1} \beta_{i,j} k_j\right).$$

The $(k+1)$th answer for y is

$$y_{k+1} = y_k + h \sum_{j=0}^{s} \gamma_j k_j.$$

Digression: Butcher tableau

Forward Euler:

$$
\begin{aligned}
k_1 &= f(t_k, y_k) \\
y_{k+1} &= y_k + hk_1
\end{aligned}
$$

$$
\begin{array}{c|c}
0 & \\
\hline
 & 1
\end{array}
$$

Two-step methods:

$$
\begin{aligned}
k_1 &= f(t_k, y_k) \\
k_2 &= f(t_k + \alpha h, y_k + \beta h k_1) \\
y_{k+1} &= y_k + h(ak_1 + bk_2)
\end{aligned}
$$

$$
\begin{array}{c|cc}
0 & & \\
\alpha & \beta & \\
\hline
 & a & b
\end{array}
$$

Digression: Butcher tableau

RK4:

$$
\begin{aligned}
k_1 &= f(t_k, y_k) \\
k_2 &= f(t_k + h/2, y_k + hk_1/2) \\
k_3 &= f(t_k + h/2, y_k + hk_2/2) \\
k_4 &= f(t_k + h, y_k + hk_3) \\
y_{k+1} &= y_k + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

$$
\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$