Lecture Notes on Scientific Computing

Author: Windson
Course: Scientific Computing

Department of Applied Mathematics
National Chung Hsing University

145 Xingda Road, South District, Taichung City 40227, Taiwan (R.O.C.)
e-mail: info AT windson.cc

License
This document is licensed under the Creative Commons Attribution 4.0 International License (CC BY-SA 4.0).

March 2, 2026

Contents

1 Polynomial Interpolation **1**

1.1 Lagrange Basis 1
1.2 Example 2
1.3 Uniqueness 2
1.4 Partition of Unity and Affine Invariance 3
1.5 Nodal Polynomial Form 3
1.6 Interpolation Error 4
1.7 Derivatives of the Interpolant 4
1.8 Runge's Phenomenon 5

2 Numerical Integration **6**

2.1 Left and Right Rectangle Rules 6
2.2 Midpoint Rule 7
2.3 Trapezoidal Rule 7
2.4 Simpson's Rule from Lagrange Interpolation 8
2.5 Numerical Experiment 9

3 Initial-Value Problems **9**

3.1 Euler Method 10
3.2 Midpoint Method 10
3.3 Heun's Method 10
3.4 Kutta's Third-Order Method 11
3.5 Classical RK4 11

4 Error, Consistency, and Stability **12**

4.1 Euler Residual 12
4.2 Euler Error Equation 12
4.3 Stability and Amplification Factors 13

5 Nonlinear Simple Pendulum **14**

5.1 Physical Model 14
5.2 Small-Angle Solution 14
5.3 Large-Angle Period 15
5.4 First-Order System 16
5.5 RK4 Pendulum Simulation 16
5.6 Exact Trajectory and Convergence 17
5.7 Period Extraction by Zero Crossings 17
5.8 Symplectic Euler 18

6 Higher-Order Runge–Kutta Methods **18**

7 Practical Checklist **19**

Scientific computing develops numerical methods for problems whose exact solutions are unavailable, inconvenient, or too expensive to evaluate. The central workflow is:

1. formulate a mathematical model;
2. replace derivatives, integrals, or functions by computable approximations;
3. derive an error estimate;
4. study convergence and stability; and
5. validate the method with computation.

This note collects the interpolation, numerical integration, initial-value problem, Runge–Kutta, stability, and nonlinear-pendulum material from the course.

1 Polynomial Interpolation

Suppose that a function is known at distinct nodes

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n), \qquad y_i = f(x_i).$$

We seek a polynomial P_n of degree at most n satisfying

$$P_n(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

1.1 Lagrange Basis

For each node x_i, construct a polynomial that vanishes at every other node. Its unnormalized numerator is

$$\prod_{\substack{j=0 \\ j\neq i}}^{n} (x - x_j).$$

At $x = x_i$, this numerator has the value

$$\prod_{\substack{j=0 \\ j\neq i}}^{n} (x_i - x_j).$$

Dividing by this value normalizes the polynomial to one:

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

Indeed, if $k \neq i$, one factor is $x_k - x_k = 0$, so $L_i(x_k) = 0$. If $k = i$, every numerator and denominator factor is equal, so $L_i(x_i) = 1$. Therefore,

$$L_i(x_j) = \delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Taking a linear combination with weights y_i gives the Lagrange interpolating polynomial:

$$P_n(x) = \sum_{i=0}^{n} y_i L_i(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

At a node x_k,

$$P_n(x_k) = \sum_{i=0}^{n} y_i L_i(x_k) = \sum_{i=0}^{n} y_i \delta_{ik} = y_k,$$

so the polynomial interpolates every data point.

1.2 Example

For $(x_0, y_0) = (1, 2)$, $(x_1, y_1) = (2, 4)$, and $(x_2, y_2) = (4, 8)$, the basis function associated with $x_1 = 2$ is

$$\begin{aligned} L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \\ &= \frac{(x - 1)(x - 4)}{(2 - 1)(2 - 4)} \\ &= -\frac{1}{2}(x - 1)(x - 4). \end{aligned}$$

At $x = 3$,

$$L_0(3) = -\frac{1}{3}, \qquad L_1(3) = 1, \qquad L_2(3) = \frac{1}{3}.$$

Consequently,

$$P_2(3) = 2\left(-\frac{1}{3}\right) + 4(1) + 8\left(\frac{1}{3}\right) = 6.$$

1.3 Uniqueness

Suppose P_n and Q_n are two polynomials of degree at most n that interpolate the same $n + 1$ data points. Let

$$R(x) = P_n(x) - Q_n(x).$$

Then $R(x_i) = 0$ for every i, so R has at least $n+1$ distinct roots. A nonzero polynomial of degree at most n can have at most n roots. Hence R is the zero polynomial and

$$P_n(x) = Q_n(x).$$

The interpolating polynomial is therefore unique.

1.4 Partition of Unity and Affine Invariance

Apply interpolation to the constant function $f(x) = 1$. Since $f(x_i) = 1$,

$$1 = P_n(x) = \sum_{i=0}^{n} L_i(x).$$

Thus the basis functions form a partition of unity: their weights always sum to one. In particular, interpolation preserves constants and affine combinations.

If the independent variable is changed by an affine map $x = at + b$, the same construction in the t-coordinate produces the correspondingly transformed interpolating polynomial. Translation and scaling change the coordinates, but not the interpolation structure.

1.5 Nodal Polynomial Form

Define the nodal polynomial

$$Q(x) = \prod_{j=0}^{n} (x - x_j).$$

Removing the factor corresponding to x_i gives

$$\frac{Q(x)}{x - x_i} = \prod_{\substack{j=0 \\ j \neq i}}^{n} (x - x_j).$$

Differentiating the product and evaluating at x_i, every term vanishes except the term in which the factor $(x - x_i)$ is differentiated:

$$Q'(x_i) = \prod_{\substack{j=0 \\ j \neq i}}^{n} (x_i - x_j).$$

Therefore,

$$L_i(x) = \frac{Q(x)}{(x - x_i)Q'(x_i)},$$

and the interpolant can also be written as

$$P_n(x) = \sum_{i=0}^{n} \frac{Q(x)}{(x - x_i)Q'(x_i)} f(x_i).$$

1.6 Interpolation Error

Assume $f \in C^{n+1}[a, b]$, with all nodes and the evaluation point x in the interval. Define

$$E(x) = f(x) - P_n(x).$$

At the interpolation nodes, $E(x_i) = 0$. For a point x not equal to a node, choose a coefficient $c(x)$ so that

$$c(x) = \frac{f(x) - P_n(x)}{Q(x)}.$$

Now define an auxiliary function of z:

$$F(z) = f(z) - P_n(z) - c(x)Q(z).$$

By construction, $F(x_i) = 0$ for $i = 0, \ldots, n$, and $F(x) = 0$. Thus F has $n + 2$ distinct roots. Repeated use of Rolle's theorem gives a point ξ between the nodes and x such that

$$F^{(n+1)}(\xi) = 0.$$

Since $P_n^{(n+1)} = 0$ and $Q^{(n+1)} = (n + 1)!$,

$$0 = f^{(n+1)}(\xi) - c(x)(n + 1)!.$$

Therefore,

$$c(x) = \frac{f^{(n+1)}(\xi)}{(n + 1)!},$$

and the error formula is

$$\boxed{f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n + 1)!} \prod_{i=0}^{n} (x - x_i)}.$$

If f is a polynomial of degree $k \leq n$, then $f^{(n+1)} = 0$. Hence the interpolation is exact: $E(x) = 0$ for every x.

1.7 Derivatives of the Interpolant

Differentiate the interpolant term by term:

$$P_n'(x) = \sum_{i=0}^{n} f(x_i) L_i'(x).$$

Starting with

$$Q(x) = (x - x_i) Q'(x_i) L_i(x),$$

differentiate once:

$$Q'(x) = Q'(x_i) L_i(x) + (x - x_i) Q'(x_i) L_i'(x).$$

At another node x_j, $j \neq i$, $L_i(x_j) = 0$, so

$$L_i'(x_j) = \frac{Q'(x_j)}{(x_j - x_i) Q'(x_i)}.$$

At $x = x_i$, differentiate the identity once more:

$$Q''(x) = 2Q'(x_i) L_i'(x) + (x - x_i) Q'(x_i) L_i''(x).$$

Evaluating at x_i gives

$$L_i'(x_i) = \frac{Q''(x_i)}{2Q'(x_i)}.$$

The derivative error follows by differentiating the interpolation error. Writing the intermediate point as $\xi = \xi(x)$,

$$f'(x) - P_n'(x) = \frac{1}{(n+1)!} \left[Q'(x) f^{(n+1)}(\xi(x)) + Q(x) f^{(n+2)}(\xi(x)) \xi'(x) \right].$$

At a node x_i, the second term disappears because $Q(x_i) = 0$. Hence

$$f'(x_i) - P_n'(x_i) = \frac{f^{(n+1)}(\xi_i)}{(n+1)!} \prod_{\substack{k=0 \\ k \neq i}}^{n} (x_i - x_k).$$

1.8 Runge's Phenomenon

Increasing the number of nodes does not always improve interpolation. For

$$f(x) = \frac{1}{1 + 25x^2}, \qquad -1 \leq x \leq 1,$$

use equally spaced nodes $x_i = -1 + 2i/n$. The unique polynomial is

$$P_n(x) = \sum_{i=0}^{n} f(x_i) \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

For large n, the product factors become badly conditioned near the endpoints and the interpolant can oscillate strongly there. This is Runge's phenomenon. More nodes are not sufficient by themselves; node placement, such as Chebyshev nodes, is also important.

2 Numerical Integration

Let

$$I = \int_a^b f(x)\,dx,$$

and divide $[a, b]$ into n equal subintervals. Set

$$h = \Delta x = \frac{b-a}{n}, \qquad x_i = a + ih.$$

2.1 Left and Right Rectangle Rules

The left-point approximation is

$$I_L = h \sum_{i=1}^{n} f(x_{i-1}),$$

and the right-point approximation is

$$I_R = h \sum_{i=1}^{n} f(x_i).$$

For the right rule, write the error as a sum of local errors:

$$\begin{aligned} E_R &= I - I_R \\ &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} [f(x) - f(x_i)]\,dx. \end{aligned}$$

Taylor's theorem about the right endpoint gives

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{f''(\xi_i)}{2}(x - x_i)^2.$$

Therefore,

$$E_R = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} \left[f'(x_i)(x - x_i) + \frac{f''(\xi_i)}{2}(x - x_i)^2 \right] dx.$$

If $M_1 = \max_{[a,b]} |f'(x)|$, the leading term satisfies

$$\left| \int_{x_{i-1}}^{x_i} f'(x_i)(x - x_i)\,dx \right| \le \frac{M_1 h^2}{2}.$$

Summing n intervals,

$$|E_R| \lesssim n\frac{M_1 h^2}{2} = \frac{M_1(b-a)^2}{2n}.$$

The same argument about the left endpoint gives

$$|E_L| \lesssim \frac{M_1(b-a)^2}{2n}.$$

Thus both endpoint rules are first-order methods: their global error is $O(h) = O(n^{-1})$.

2.2 Midpoint Rule

Let $m_i = (x_{i-1} + x_i)/2$. Approximate f about the midpoint:

$$f(x) = f(m_i) + f'(m_i)(x - m_i) + \frac{f''(\xi_i)}{2}(x - m_i)^2.$$

The local midpoint error is

$$E_{M,i} = \int_{x_{i-1}}^{x_i} [f(x) - f(m_i)]\, dx.$$

The linear term cancels by symmetry:

$$\int_{x_{i-1}}^{x_i} (x - m_i)\, dx = 0.$$

For the quadratic term, use $t = x - m_i$, with limits $-h/2$ and $h/2$:

$$\int_{-h/2}^{h/2} \frac{f''(\xi_i)}{2} t^2\, dt = \frac{f''(\xi_i)}{2}\frac{h^3}{12} = \frac{f''(\xi_i)h^3}{24}.$$

If $M_2 = \max|f''|$, then

$$|E_M| \le n\frac{M_2 h^3}{24} = \frac{M_2(b-a)^3}{24n^2}.$$

The midpoint rule is second-order.

2.3 Trapezoidal Rule

On $[x_{i-1}, x_i]$, the trapezoidal approximation is

$$T_i = \frac{h}{2}[f(x_{i-1}) + f(x_i)].$$

Summing gives

$$T_n = h\left[\frac{f(x_0)}{2} + \sum_{i=1}^{n-1} f(x_i) + \frac{f(x_n)}{2}\right].$$

The local trapezoidal error is

$$E_{T,i} = -\frac{h^3}{12} f''(\xi_i),$$

so

$$|E_T| \leq \frac{M_2 (b-a)^3}{12 n^2}.$$

The trapezoidal rule is also second-order.

2.4 Simpson's Rule from Lagrange Interpolation

Use three consecutive points x_{i-1}, x_i, and x_{i+1}, where the spacing is h. The quadratic Lagrange interpolant is

$$P_2(x) = L_0(x) f(x_{i-1}) + L_1(x) f(x_i) + L_2(x) f(x_{i+1}),$$

where

$$\begin{aligned} L_0(x) &= \frac{(x - x_i)(x - x_{i+1})}{2h^2}, \\ L_1(x) &= -\frac{(x - x_{i-1})(x - x_{i+1})}{h^2}, \\ L_2(x) &= \frac{(x - x_{i-1})(x - x_i)}{2h^2}. \end{aligned}$$

Set $x = x_i + ht$, so $dx = h\,dt$ and $t \in [-1, 1]$. Then

$$\int_{x_{i-1}}^{x_{i+1}} L_0(x)\,dx = \frac{h}{3},$$

because

$$\frac{h}{2} \int_{-1}^{1} (t^2 - t)\,dt = \frac{h}{3}.$$

Similarly,

$$\int_{x_{i-1}}^{x_{i+1}} L_1(x)\,dx = \frac{4h}{3},$$

because

$$-h \int_{-1}^{1} (t^2 - 1)\,dt = \frac{4h}{3},$$

and

$$\int_{x_{i-1}}^{x_{i+1}} L_2(x)\,dx = \frac{h}{3}.$$

Therefore,

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\,dx \approx \frac{h}{3}[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})].$$

Adding the $n/2$ panels, with n even, gives the composite Simpson rule:

$$S_n = \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 4f(x_{n-1}) + f(x_n)\right].$$

Taylor expansion about the midpoint shows that the odd powers cancel. The first remaining neglected term is proportional to $h^5 f^{(4)}$, so the global error is $O(h^4)$.

2.5 Numerical Experiment

For a smooth test function such as $f(x) = e^{-x^2}$, compute a high-precision reference value and compare the five rules for $N = 10, 20, 40, 80, 160$. If E_N denotes the error, the observed order is estimated by

$$p \approx \frac{\log(|E_N|/|E_{2N}|)}{\log 2}.$$

The expected orders are one for left/right rules, two for midpoint and trapezoidal rules, and four for Simpson's rule. Symmetry can occasionally produce a higher observed order for a particular function, but the general theoretical guarantees remain those listed above.

3 Initial-Value Problems

Consider

$$u'(t) = f(t, u(t)), \qquad u(0) = u_0.$$

Integrating from t to $t + h$ and applying the fundamental theorem of calculus gives the exact identity

$$u(t + h) = u(t) + \int_t^{t+h} f(s, u(s))\,ds.$$

Every one-step method below approximates this integral in a different way.

3.1 Euler Method

Approximate the integrand by its left endpoint:

$$f(s, u(s)) \approx f(t, u(t)).$$

Then

$$\int_t^{t+h} f(s, u(s))\, ds \approx hf(t, u(t)),$$

and, with $t_n = nh$ and $v_n \approx u(t_n)$,

$$\boxed{v_{n+1} = v_n + hf(t_n, v_n)}.$$

Writing $K_1 = f(t_n, v_n)$ gives $v_{n+1} = v_n + hK_1$.

3.2 Midpoint Method

Approximate the integral with the slope at the midpoint:

$$u(t+h) \approx u(t) + hf\left(t + \frac{h}{2}, u\left(t + \frac{h}{2}\right)\right).$$

The midpoint value is unknown, so predict it by a half Euler step:

$$v^{(1)} = v_n + \frac{h}{2} f(t_n, v_n).$$

Thus

$$\begin{aligned} K_1 &= f(t_n, v_n), \\ K_2 &= f\left(t_n + \frac{h}{2}, v_n + \frac{h}{2} K_1\right), \\ v_{n+1} &= v_n + hK_2. \end{aligned}$$

3.3 Heun's Method

Apply the trapezoidal rule to the IVP integral:

$$u(t+h) \approx u(t) + \frac{h}{2}[f(t, u(t)) + f(t+h, u(t+h))].$$

Predict the unknown endpoint by Euler:

$$v^{(1)} = v_n + hf(t_n, v_n).$$

Then

$$
\begin{aligned}
K_1 &= f(t_n, v_n), \\
K_2 &= f(t_n + h, v_n + hK_1), \\
v_{n+1} &= v_n + \frac{h}{2}(K_1 + K_2).
\end{aligned}
$$

Both midpoint and Heun methods have local error $O(h^3)$ and global error $O(h^2)$.

3.4 Kutta's Third-Order Method

One third-order method from the course is

$$
\begin{aligned}
K_1 &= f(t_n, v_n), \\
K_2 &= f(t_n + h, v_n + hK_1), \\
K_3 &= f\left(t_n + \frac{h}{2}, v_n + \frac{h}{2}\frac{K_1 + K_2}{2}\right), \\
v_{n+1} &= v_n + h\frac{K_1 + K_2 + 4K_3}{6}.
\end{aligned}
$$

The weights are chosen so that the Taylor expansion agrees through terms of order three.

3.5 Classical RK4

Use two midpoint estimates, one based on K_1 and one based on K_2:

$$
\begin{aligned}
K_1 &= f(t_n, v_n), \\
K_2 &= f\left(t_n + \frac{h}{2}, v_n + \frac{h}{2}K_1\right), \\
K_3 &= f\left(t_n + \frac{h}{2}, v_n + \frac{h}{2}K_2\right), \\
K_4 &= f(t_n + h, v_n + hK_3).
\end{aligned}
$$

The update is

$$
\boxed{v_{n+1} = v_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)}.
$$

The four slopes approximate the integral with a Simpson-like weighted average. Taylor matching gives local truncation error $O(h^5)$ and global error $O(h^4)$.

4 Error, Consistency, and Stability

4.1 Euler Residual

Let $u_n = u(t_n)$ be the exact solution and v_n the numerical solution. The residual of Euler's scheme is

$$r_n = \frac{u_{n+1} - u_n}{h} - f(t_n, u_n).$$

Taylor's theorem gives

$$u_{n+1} = u_n + hu'(t_n) + \frac{h^2}{2}u''(\xi_n),$$

so, because $u' = f(t, u)$,

$$r_n = \frac{h}{2}u''(\xi_n) = O(h).$$

Since $r_n \to 0$ as $h \to 0$, the method is consistent and first-order accurate as an approximation to the differential equation.

4.2 Euler Error Equation

Let $e_n = u_n - v_n$. Subtract the numerical update from the exact equation with residual:

$$\frac{u_{n+1} - u_n}{h} = f(t_n, u_n) + r_n, \qquad \frac{v_{n+1} - v_n}{h} = f(t_n, v_n).$$

Then

$$e_{n+1} = e_n + h[f(t_n, u_n) - f(t_n, v_n)] + hr_n.$$

By the mean-value theorem,

$$f(t_n, u_n) - f(t_n, v_n) = f_u(t_n, \eta_n)e_n,$$

where η_n lies between u_n and v_n. Hence

$$e_{n+1} = [1 + hf_u(t_n, \eta_n)]e_n + hr_n.$$

Assume $|u''(t)| \le M$ and $|f_u(t, u)| \le L$. Then

$$|e_{n+1}| \le (1 + hL)|e_n| + \frac{h^2 M}{2}.$$

Iterating from $e_0 = 0$,

$$|e_{n+1}| \le \frac{h^2 M}{2} \sum_{k=0}^{n} (1+hL)^k = \frac{hM}{2L}[(1+hL)^{n+1} - 1].$$

Since $1 + hL \le e^{hL}$ and $T = (n+1)h$,

$$|e_{n+1}| \le \frac{hM}{2L}(e^{LT} - 1).$$

For fixed T, this tends to zero linearly with h.

4.3 Stability and Amplification Factors

For the test equation

$$u' = \lambda u,$$

Euler gives

$$v_{n+1} = (1 + h\lambda)v_n.$$

The amplification factor is

$$R(z) = 1 + z, \qquad z = h\lambda.$$

Absolute stability requires

$$|R(z)| = |1 + h\lambda| \le 1.$$

For $\lambda < 0$, this becomes

$$-1 \le 1 + h\lambda \le 1,$$

so

$$0 \le h \le \frac{2}{|\lambda|}.$$

For example, $u' = -5u$ requires $h \le 0.4$ for stable explicit Euler integration. For a general Runge–Kutta method, the stability condition is $|R(h\lambda)| \le 1$; for a system, $h\lambda_i$ must lie in the method's stability region for every relevant Jacobian eigenvalue λ_i.

5 Nonlinear Simple Pendulum

5.1 Physical Model

Consider a point mass m attached to a massless, inextensible string of length L in a uniform gravitational field g. Let $\theta(t)$ be the angle measured from the downward vertical. The arc length is

$$s = L\theta.$$

The gravitational force has tangential component $-mg\sin\theta$. Newton's second law in the tangential direction gives

$$-mg\sin\theta = m\frac{d^2s}{dt^2}.$$

Since $s = L\theta$,

$$\frac{d^2s}{dt^2} = L\frac{d^2\theta}{dt^2}.$$

Substituting and cancelling m yields

$$\boxed{\theta'' + \frac{g}{L}\sin\theta = 0}.$$

The initial-value problem is

$$\theta(0) = \theta_0, \qquad \omega(0) = \theta'(0) = \omega_0.$$

5.2 Small-Angle Solution

For small angles measured in radians,

$$\sin\theta \approx \theta.$$

The nonlinear equation becomes

$$\theta'' + \frac{g}{L}\theta = 0.$$

Let $\omega_* = \sqrt{g/L}$. The general solution is

$$\theta(t) = A\cos(\omega_* t) + B\sin(\omega_* t).$$

At $t = 0$,

$$\theta(0) = A = \theta_0.$$

Differentiating,

$$\theta'(t) = -A\omega_* \sin(\omega_* t) + B\omega_* \cos(\omega_* t),$$

so

$$\theta'(0) = B\omega_* = \omega_0, \qquad B = \omega_0 \sqrt{\frac{L}{g}}.$$

Therefore,

$$\boxed{\theta(t) = \theta_0 \cos\left(\sqrt{\frac{g}{L}}t\right) + \omega_0 \sqrt{\frac{L}{g}} \sin\left(\sqrt{\frac{g}{L}}t\right)}.$$

Writing this as $C\cos(\omega_* t - \phi)$ gives

$$C = \sqrt{\theta_0^2 + \omega_0^2 \frac{L}{g}}, \qquad \tan\phi = \frac{\omega_0}{\theta_0}\sqrt{\frac{L}{g}}.$$

The period satisfies $\omega_* T = 2\pi$, hence

$$\boxed{T_0 = 2\pi\sqrt{\frac{L}{g}}}.$$

This small-angle period is independent of amplitude and initial angular velocity.

5.3 Large-Angle Period

For large angles, retain $\sin\theta$. Conservation of energy gives, for release from rest at amplitude θ_0,

$$\frac{1}{2}mL^2\dot{\theta}^2 + mgL(1-\cos\theta) = mgL(1-\cos\theta_0).$$

Using $1 - \cos\theta = 2\sin^2(\theta/2)$,

$$\dot{\theta}^2 = \frac{4g}{L}\left[\sin^2\left(\frac{\theta_0}{2}\right) - \sin^2\left(\frac{\theta}{2}\right)\right].$$

Set $k = \sin(\theta_0/2)$ and use the substitution

$$\sin\frac{\theta}{2} = k\sin\phi.$$

After simplifying the quarter-period integral,

$$\frac{T}{4} = \sqrt{\frac{L}{g}} \int_0^{\pi/2} \frac{d\phi}{\sqrt{1 - k^2\sin^2\phi}}.$$

Thus

$$\boxed{T = 4\sqrt{\frac{L}{g}}K(k)},$$

where $K(k)$ is the complete elliptic integral of the first kind. In software such as SciPy, `ellipk` commonly takes the parameter $m = k^2$:

```
from scipy.special import ellipk
import numpy as np

def exact_period(L, g, theta0):
    m = np.sin(theta0 / 2.0) ** 2
    return 4.0 * np.sqrt(L / g) * ellipk(m)
```

5.4 First-Order System

Define angular velocity $\omega = \theta'$. Then

$$\frac{d}{dt}\begin{bmatrix}\theta \\ \omega\end{bmatrix} = \begin{bmatrix}\omega \\ -(g/L)\sin\theta\end{bmatrix} = f(t, y), \qquad y(0) = \begin{bmatrix}\theta_0 \\ \omega_0\end{bmatrix}.$$

RK4 is applied to this vector equation component by component.

5.5 RK4 Pendulum Simulation

```
import numpy as np

def pendulum_derivatives(t, y, g, L):
    theta, omega = y
    return np.array([omega, -(g / L) * np.sin(theta)])

def rk4_step(f, t, y, h, g, L):
    k1 = f(t, y, g, L)
    k2 = f(t + h / 2, y + h * k1 / 2, g, L)
    k3 = f(t + h / 2, y + h * k2 / 2, g, L)
    k4 = f(t + h, y + h * k3, g, L)
    return y + h * (k1 + 2*k2 + 2*k3 + k4) / 6

def simulate(theta0=np.pi / 4, L=1.0, g=9.81,
             t_end=10.0, h=0.05):
    times = np.arange(0.0, t_end, h)
    states = np.zeros((len(times), 2))
```

```
    states[0] = [theta0, 0.0]
    for i in range(1, len(times)):
        states[i] = rk4_step(
            pendulum_derivatives, times[i - 1], states[i - 1], h,
                g, L
        )
    return times, states
```

The trajectory can be plotted as $\theta(t)$, and the phase portrait is obtained by plotting angular velocity against angle.

5.6 Exact Trajectory and Convergence

For release from rest, the nonlinear trajectory can be expressed with Jacobi elliptic functions. With

$$k = \sin(\theta_0/2), \qquad m = k^2, \qquad u = \sqrt{g/L}\, t,$$

one representation is

$$\theta(t) = 2 \arcsin\left(k \frac{\mathrm{cn}(u, m)}{\mathrm{dn}(u, m)} \right).$$

The numerical error at the time nodes is

$$E_i = |\theta_{\mathrm{RK4}}(t_i) - \theta_{\mathrm{exact}}(t_i)|.$$

Repeat the experiment with several step sizes, for example

$$h \in \{0.1, 0.05, 0.025, 0.0125, 0.00625\}.$$

If RK4 is in its asymptotic regime, halving h reduces the global error by approximately $2^4 = 16$. A log-log graph of error versus h has slope approximately four:

$$|E(h)| \approx C h^4.$$

5.7 Period Extraction by Zero Crossings

A numerical period can be obtained from successive turning points, where $\omega(t) = 0$. If ω_n and ω_{n+1} have opposite signs, linearly interpolate the zero:

$$t_{\mathrm{root}} = t_n - \omega_n \frac{t_{n+1} - t_n}{\omega_{n+1} - \omega_n}.$$

The time between every second turning point is one full oscillation period. Comparing this value with the elliptic-integral period measures the combined integration and event-location error.

5.8 Symplectic Euler

The pendulum is conservative, so long-time energy behavior matters. Explicit RK4 is accurate over a fixed interval but may slowly distort energy over very long integrations. A semi-implicit Euler method updates velocity first and then angle:

$$\omega_{n+1} = \omega_n - h\frac{g}{L}\sin\theta_n,$$
$$\theta_{n+1} = \theta_n + h\omega_{n+1}.$$

It is only first-order accurate, but its geometric structure often prevents the systematic energy drift seen in ordinary explicit Euler.

6 Higher-Order Runge–Kutta Methods

An explicit Runge–Kutta method obtains several stage slopes and combines them with weights. Increasing the order requires satisfying increasingly many nonlinear order conditions. These conditions can be organized using rooted trees and B-series.

For explicit methods of order $p \geq 5$, the Butcher barrier implies that more than p stages are required. For example, fifth order needs at least six stages and sixth order needs at least seven stages. The additional evaluations reduce the efficiency advantage of a fixed high-order method.

Embedded methods such as RKF45 and Dormand–Prince use the same stages with two weight vectors b_i and b_i^*:

$$y_{n+1} = y_n + h\sum_i b_i K_i, \qquad \widehat{y}_{n+1} = y_n + h\sum_i b_i^* K_i.$$

The difference

$$\|y_{n+1} - \widehat{y}_{n+1}\|$$

estimates the local error. The solver can then increase h when the estimate is small and decrease h when it is large. This produces adaptive step-size control without evaluating the differential equation twice with two unrelated methods.

7 Practical Checklist

- Check that interpolation nodes are distinct.
- Use the interpolation error formula to identify smoothness and node-placement effects.
- Distinguish local truncation error from accumulated global error.
- Verify an observed convergence order with a refinement table or log-log plot.
- For ODEs, check both accuracy and stability; a small local error does not guarantee a useful long-time solution.
- For conservative dynamics, monitor energy and phase-space behavior.
- Keep angle units in radians when evaluating trigonometric functions.
- When using `scipy.special.ellipk`, pass the parameter $m = k^2$, not the modulus k itself.