

Statistics and Numerical Methods, Tsinghua University

Ordinary Differential Equations

Xuening Bai (白雪宁)

Institute for Advanced Study (IASTU) & Department of Astronomy (DoA)



清華大學

Tsinghua University

Oct. 22, 2024

What are ordinary differential equations?

Ordinary differential equation (ODE) is a differential equation containing one or more functions of **one independent variable** and the derivatives of those functions.

In most general form: $F(x, y, y', y'', \dots, y^{(n)}) = 0$

More commonly, in explicit form: $y^{(n)} = F(x, y, y', \dots, y^{(n-1)})$

Would like to solve for $y(x)$, typically over some interval $[x_1, x_2]$.

Need to supply n conditions at x_1 and/or x_2 for a unique solution.

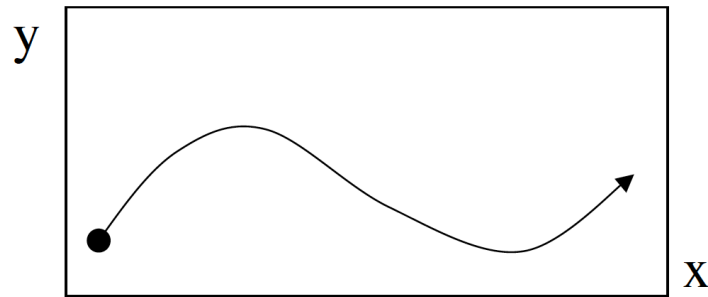
There are $n+1$ unknowns for 1 equation.

Can be generalized to vector form, where each component of y has an corresponding equation.

What is ordinary differential equation(s)?

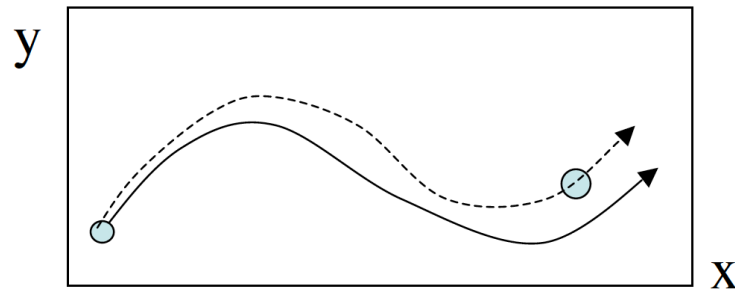
In most general form: $F(x, y, y', y'', \dots, y^{(n)}) = 0$

Initial value problem: know y and all its derivatives at the same point x_0 .



Can obtain a unique solution by integrating from that point.

Boundary value problem: know y and its derivatives at more than one point.



Must guess boundary values to start and iterate towards the solution.

Outline

- Initial value problem

 - Fundamentals

 - Runge-Kutta methods

 - Bulirsch-Stoer method

 - Multistep methods (optional)

 - Stability analysis

 - Stiff systems

 - Geometric integrators

- Boundary value problem

 - Shooting method

 - Relaxation method

Initial value problem (IVP)

In explicit form, a general IVP reads

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, \dots, y_N), \quad i = 1, \dots, N$$

where $y_i(x)$ is known at $x=x_0$.

The above is a set of 1st-order ODEs. Higher-order ODEs can all be reduced to this form by redefining variables:

$$\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x) \quad \longrightarrow \quad \begin{aligned} \frac{dy}{dx} &= z(x) \\ \frac{dz}{dx} &= r(x) - q(x)z(x) \end{aligned}$$

It thus suffices to consider 1st order ODEs.

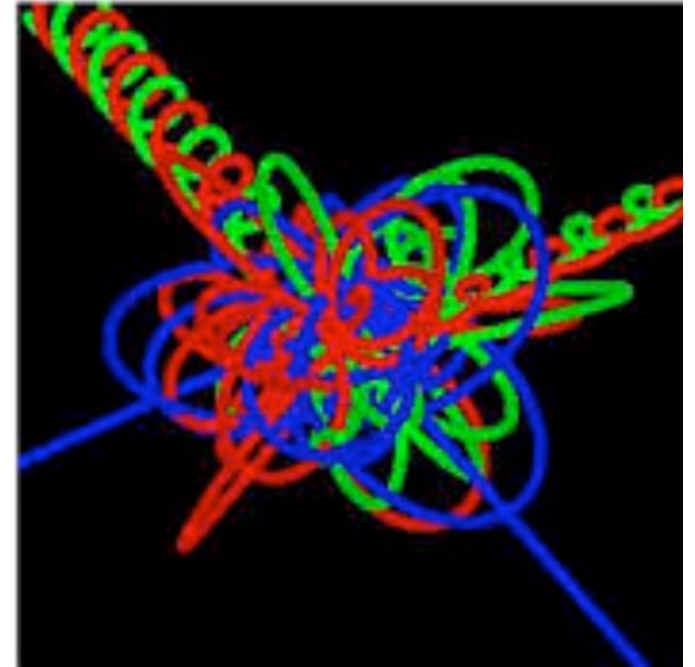
Example: orbits of celestial objects

Integrating the trajectories of stars is the most fundamental task for studying orbital dynamics:

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i$$

$$\frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i$$

$$\mathbf{F}_i = \sum_{i \neq j} \frac{Gm_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|^2}$$

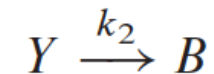
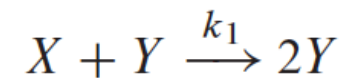
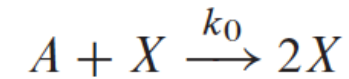


Goal is to accurately integrate the above equations from some starting time.

More tricky aspect: total energy is conserved in nature, and we don't want to break it at numerical level!

Example: kinetic equations of a chemical network

Chemical (or nuclear) reaction networks is made of a series of reactions with rate coefficients k_i :



The abundances of each species satisfy the following differential equations:

$$\frac{d[A]}{dt} = -k_0[A][X] \equiv -a_0$$

$$\frac{d[X]}{dt} = k_0[A][X] - k_1[X][Y] \equiv a_0 - a_1$$

$$\frac{d[Y]}{dt} = k_1[X][Y] - k_2[Y] \equiv a_1 - a_2$$

$$\frac{d[B]}{dt} = k_2[Y] \equiv a_2$$

Note that the rates of different reactions can vary dramatically, which can be numerically demanding.

General concepts

Consider a simple 1st-order ODE $\frac{dy}{dx} = f(x, y)$, $y(x_0) = y_0$

This equation has an accurate solution, denoted as $y(x)$.

We normally solve it in steps, commonly of fixed **step size h** , so that in the n th step, we have

$$x_n = x_{n-1} + h$$

The resulting numerical solution at this step is denoted by y_n .

The **global truncation error** is defined as $e_n \equiv y(x_n) - y_n$

This is not only related to errors in the present step, but reflects all errors accumulated from previous steps.

Euler method

Consider a simple 1st-order ODE $\frac{dy}{dx} = f(x, y)$, $y(x_0) = y_0$

At n th step, use simple finite difference to replace the derivative:

$$\frac{y_{n+1} - y_n}{h} = f(x_n, y_n) \quad \Rightarrow \quad y_{n+1} = y_n + hf(x_n, y_n)$$

This gives the **Euler method**.

Alternatively, it can be obtained by considering Taylor expansion:

$$y(x_{n+1}) = y(x_n) + hy'(x_n) + \cancel{\frac{h^2 y''(x_n)}{2}} + \dots$$

$$\Rightarrow y_{n+1} = y_n + hf(x_n, y_n)$$

Elementary methods

Another way to look at the solution: $y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(t, y(t)) dt$

By using different methods to approximate the integral, one can derive:

(forward) **Euler method**: $y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(t, y(t)) dt \approx y_n + hf(x_n, y_n)$

Backward Euler method: $y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(t, y(t)) dt \approx y_n + hf(x_{n+1}, y_{n+1})$

Trapezoid method:

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(t, y(t)) dt \approx y_n + \frac{h}{2} \left[f(x_n, y_n) + f(x_{n+1}, y_{n+1}) \right]$$

These methods are implicit (more later)

Accuracy of a scheme

Consider a simple 1st-order ODE $\frac{dy}{dx} = f(x, y)$, $y(x_0) = y_0$

Generally, numerical solution in n th step can be written as

$$y_{n+1} = y_n + h\phi(x_n, y_n, x_{n+1}, y_{n+1})$$

Let $y(x)$ be the exact solution. We can define the **local truncation error** as

$$T_{n+1} = y(x_{n+1}) - y(x_n) - h\phi(x_n, y(x_n), x_{n+1}, y(x_{n+1}))$$

A method is of **p th order**, if $T_{n+1} = O(h^{p+1})$.

In this case, the **global truncation error** is on the order $e_n = O(h^p)$

- The Euler method, backward Euler method are 1st order accurate.
- The Trapezoid method is 2nd order accurate.

Higher-order schemes

The Euler method is not recommended for practical use because:

It is very inaccurate, and that it is not very stable (see later).

Three main types of higher-order methods:

Runge-Kutta methods:

Generalization of Euler methods to higher order to match Taylor expansion.

Bulirsch-Stoer method:

Use Richardson extrapolation to extrapolate results to 0 step size.

Multistep methods:

First predict based on previous steps, then correct towards higher order.

Runge-Kutta methods

Use additional evaluations of f to add more information.

General format for an n -stage RK method is given by:

$$y_{n+1} = y_n + h \sum_{i=1}^n c_i k_i, \quad k_1 = f(x_n, y_n), \quad k_i = f\left(x_n + a_i h, y_n + h \sum_{j=1}^{i-1} b_{ij} k_j\right)$$

where coefficients a_i, b_{ij}, c_i are constants.

First choose the # of stages.

Then determine the coefficients to match the Taylor series (there are some degrees of freedom).

The Euler method is a 1-stage RK method, and is 1st order accurate.

Runge-Kutta methods

Consider 2-stage RK methods:

$$y_{n+1} = y_n + h(c_1 k_1 + c_2 k_2) , \quad k_1 = f(x_n, y_n) , \quad k_2 = f(x_n + a_2 h, y_n + h b_{21} k_1)$$

First do Taylor expansion from this formula to obtain:

$$k_2 = f(x_n, y_n) + h(a_2 f'_x + b_{21} f f'_y) + \dots$$

$$y_{n+1} = y_n + (c_1 + c_2) h f + c_2 h^2 (a_2 f'_x + b_{21} f f'_y)$$

The solution should satisfy:

$$y(x_{n+1}) = y_n + h f(x_n, y_n) + \frac{h^2}{2} \frac{d}{dx} f(x_n, y(x_n)) + \dots = y_n + h f + \frac{h^2}{2} (f'_x + f'_y f) + \dots$$

Matching the coefficients, we arrive at: $c_1 + c_2 = 1$, $c_2 a_2 = \frac{1}{2}$, $c_2 b_{21} = \frac{1}{2}$

More unknowns (4) than constraints (3): we have one degree of freedom.

Runge-Kutta methods

Consider 2-stage RK methods:

$$y_{n+1} = y_n + h(c_1 k_1 + c_2 k_2) , \quad k_1 = f(x_n, y_n) , \quad k_2 = f(x_n + a_2 h, y_n + h b_{21} k_1)$$

The coefficients should satisfy: $c_1 + c_2 = 1$, $c_2 a_2 = \frac{1}{2}$, $c_2 b_{21} = \frac{1}{2}$

Two popular choices:

Midpoint method: $c_2 = 1, c_1 = 0, a_2 = b_{21} = 1/2$

$$y_{n+1} = y_n + h f\left(x_n + \frac{h}{2}, y_n + \frac{h f(x_n, y_n)}{2}\right)$$

Modified Euler (Heun's) method: $c_1 = c_2 = 1/2, a_2 = b_{21} = 1$

$$y_{n+1} = y_n + \frac{h}{2} \left[f(x_n, y_n) + f(x_n + h, y_n + h f(x_n, y_n)) \right]$$

Runge-Kutta methods

The most popular is the **classical RK method**, with 4 stages (known as RK4):

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) ,$$

$$k_1 = f(x_n, y_n), \quad k_3 = f\left(x_n + \frac{h}{2}, y + \frac{hk_2}{2}\right),$$

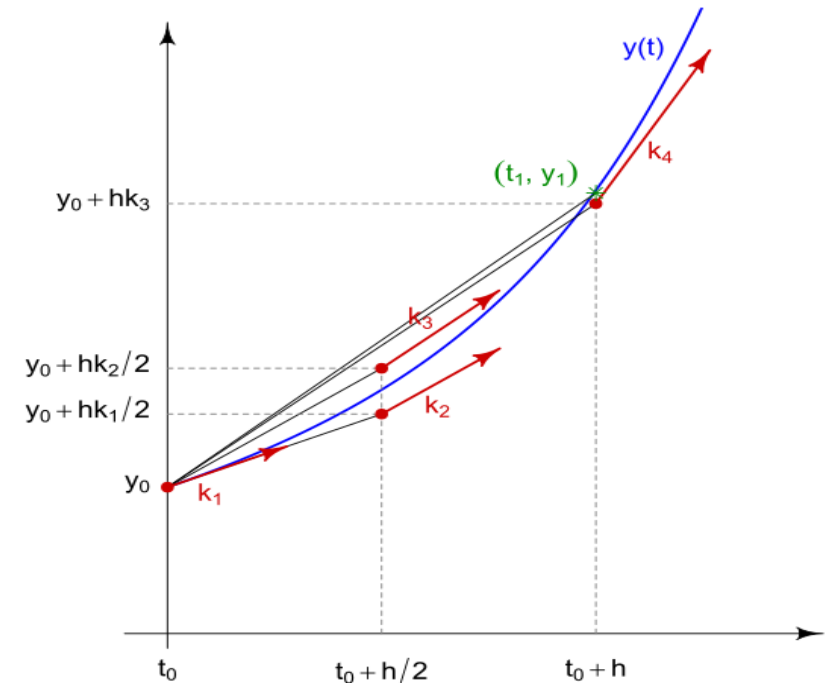
$$k_2 = f\left(x_n + \frac{h}{2}, y + \frac{hk_1}{2}\right), \quad k_4 = f(x_n + h, y + hk_3).$$

It is popular for good reasons:

It is explicit yet provide higher-order of accuracy (4th order).

It is more stable than lower-order RK methods (see later).

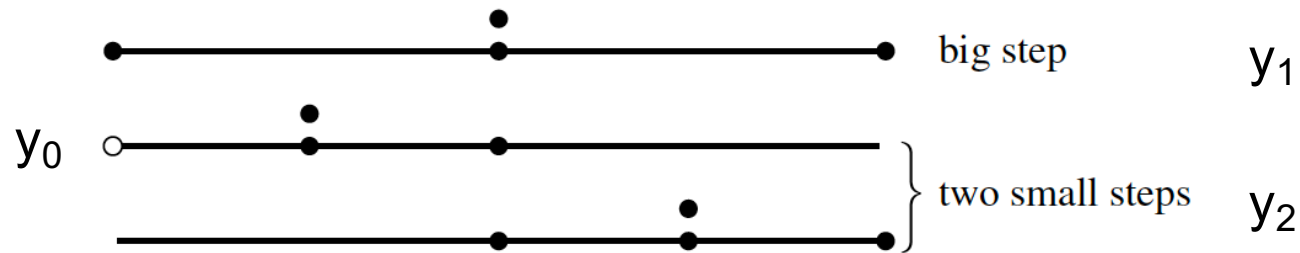
Usually, RK4 is a good starting point if you don't know what to use.



Adaptive step size

For 4th order RK4 method, local truncation error is $O(h^5)$.

General idea: **step doubling**



First use a step size of $2h$: $y(x + 2h) = y_1 + (2h)^5 A + O(h^6)$

Then two steps with step size h : $y(x + 2h) = y_2 + 2h^5 A + O(h^6)$

Therefore, we have $y(x + 2h) \approx y_2 + \frac{\Delta}{15} + O(h^6)$ where $\Delta \equiv y_2 - y_1$

Error estimate

Adaptive step size

A more efficient technique: [embedded Runge-Kutta formulas](#).

Starting from a 6-stage, 5th order RK formula:

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + c_2h, y_n + a_{21}k_1)$$

...

$$k_6 = hf(x_n + c_6h, y_n + a_{61}k_1 + \dots + a_{65}k_5)$$

$$y_{n+1} = y_n + b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4 + b_5k_5 + b_6k_6 + O(h^6)$$

*See Numerical Recipes
for coefficients.*

Embedded with a 4th-order formula

$$y_{n+1}^* = y_n + b_1^*k_1 + b_2^*k_2 + b_3^*k_3 + b_4^*k_4 + b_5^*k_5 + b_6^*k_6 + O(h^5)$$

So the local error estimate is $\Delta \equiv y_{n+1} - y_{n+1}^* = \sum_{i=1}^6 (b_i - b_i^*)k_i$

Adaptive step size

In practice, y is usually a vector. We typically set error tolerance on each component, and then take some norm:

$$\text{err} = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} \left(\frac{\Delta_i}{\text{scale}_i} \right)^2}$$

Scale is commonly made of an absolute and relative error tolerance:

$$\text{scale}_i = \text{atol}_i + |y_i| \text{rtol}_i.$$

Exactly how to choose them should be problem dependent.

[To control global error, you may also choose $\text{scale}_i = \varepsilon h^* y_i'(x)$]

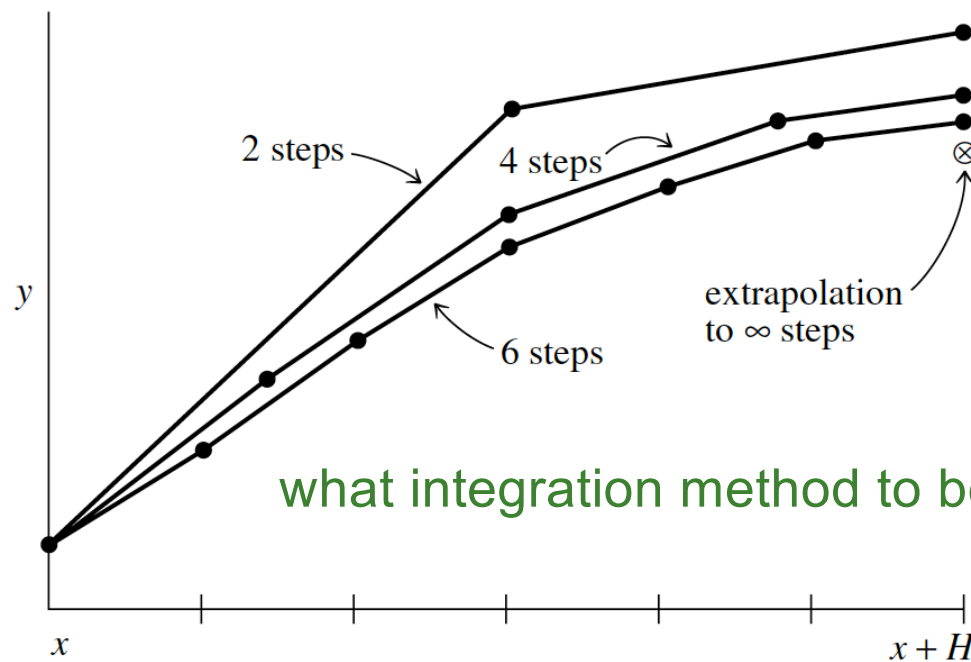
The step size should be such that $\text{err} < 1$.

If step size h_0 produces err_0 , then you should choose $h = h_0 \text{err}_0^{-1/5}$ (local error).

Bulirsch-Stoer method

Basic idea: Richardson extrapolation towards the limit of $h \rightarrow 0$.

Starting from x , integrate over a large step to $x+H$ by a sequence of n substeps each of size $h=H/n$.



what form of fitting function should be used?

what integration method to be used?

Bulirsch-Stoer method

Integration method: would like the error to contain only even powers of h .
Practically, adopt the **modified midpoint method** (2nd order accurate):

$$z_0 \equiv y(x)$$

$$z_1 = z_0 + hf(x, z_0)$$

$$z_{m+1} = z_{m-1} + 2hf(x + mh, z_m)$$



$$y_n - y(x + H) = \sum_{i=1}^{\infty} \alpha_i h^{2i}$$

$$y(x + H) \approx y_n \equiv \frac{1}{2}[z_n + z_{n-1} + hf(x + H, z_n)]$$

(non-trivial) result due to
W. Gragg

Advantage: reach higher-order by varying the # of substeps and combining results.

$$\text{For instance, reach 4}^{\text{th}} \text{ order by } y(x + H) \approx \frac{4y_n - y_{n/2}}{3}$$

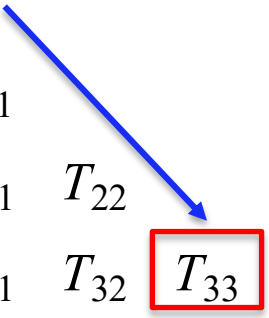
[Here, 1.5 evaluations per h to achieve 4th order, instead of 4 for RK4. BS will do even better!]

Bulirsch-Stoer method

Procedure: employs a sequence of substeps $n=2, 4, 6, \dots$ (typically up to 16), then use ~~rational function~~ or polynomial interpolation to the limit of $h \rightarrow 0$.

Use **recursive formula** (based on Neville's algorithm, see earlier lecture) to conduct polynomial interpolation:

k	n	h	y_n
1	2	$H/2$	T_{11}
2	4	$H/4$	$T_{21} \quad T_{22}$
3	6	$H/6$	$T_{31} \quad T_{32} \quad T_{33}$
...



$$T_{k,j+1} = T_{kj} + \frac{T_{kj} - T_{k-1,j}}{(n_k/n_{k-j})^2 - 1}$$

Error estimate at the k th row: $\text{err}_k = \|T_{kk} - T_{k,k-1}\| \sim H^{2k-1}$.

Can fix H , and increase k to achieve desired level of accuracy.

Alternative strategy is to fix k , and adjust H so that error is under control.

Optional: multistep methods

Basic idea: $y(x) = y_n + \int_{x_n}^x f(x', y) dx'$

Approximate $f(x,y)$ by a polynomial from previous steps, so that

$$y_{n+1} = y_n + h(\beta_0 y'_{n+1} + \beta_1 y'_n + \beta_2 y'_{n-1} + \beta_3 y'_{n-2} + \cdots)$$

If $\beta_0=0$, the method is **explicit**, otherwise implicit (see later).

Best known example is **4th-order Adams-Bashforth method**:

$$y_{n+1} = y_n + \frac{h}{24}(55y'_n - 59y'_{n-1} + 37y'_{n-2} - 9y'_{n-3})$$

Must use single-step methods to get started.

Assume fixed step size, which is not very flexible.

Optional: predictor-corrector methods

From $y_{n+1} = y_n + h(\beta_0 y'_{n+1} + \beta_1 y'_n + \beta_2 y'_{n-1} + \beta_3 y'_{n-2} + \dots)$

To avoid implicit formula, first predict y_{n+1} using an explicit Adams-Bashforth formula. For instance, in 3rd order, we have

$$y_{n+1} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2}) + O(h^4)$$

Then correct the result accordingly

$$y_{n+1} = y_n + \frac{h}{12}(5y'_{n+1} + 8y'_n - y'_{n-1}) + O(h^4)$$

This is known as [Adams-Bashforth-Moulton scheme](#).

Summary of explicit ODE solvers

- Runge-Kutta methods

For moderate accuracy requirements, RK with adaptive step size is most efficient.

- Bulirsch-Stoer method

For high-accuracy requirement and smooth problems, BS is both robust and efficient.

- Multistep methods

Historically popular, but is largely superseded by BS.

Stability analysis

Stability properties can be understood by examining local behaviors.

Consider the **model problem**: $\frac{dy}{dx} = \lambda y \quad \Rightarrow \quad y(x) = e^{\lambda x}$

If λ is negative, then y is bounded at long times.

If λ is positive, then y is unbound at long times.

If λ is imaginary, then y is oscillatory.

A method is called **A-stable** if it produces bounded solution for any λ with non-positive real part, **for any step size h** .

Conditionally stable methods produce bounded solution only for limited range of h .

It is custom to show the stability range over the complex plane λh .

Stability analysis: forward Euler method

Consider the model problem: $\frac{dy}{dx} = \lambda y$

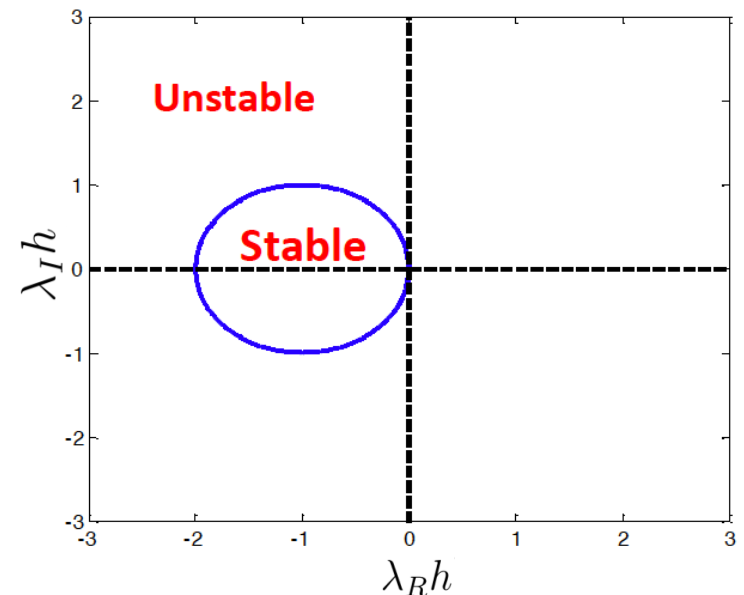
Forward Euler method yields: $y_{n+1} = y_n + h y'_n = y_n + \lambda h y_n = (1 + \lambda h) y_n$

Stable iff $|1 + \lambda h| \leq 1$

or $(1 + \lambda_R h)^2 + (\lambda_I h)^2 \leq 1$

For real λ , the maximum step size is

$$h_{\max} \leq \frac{2}{-\lambda}$$

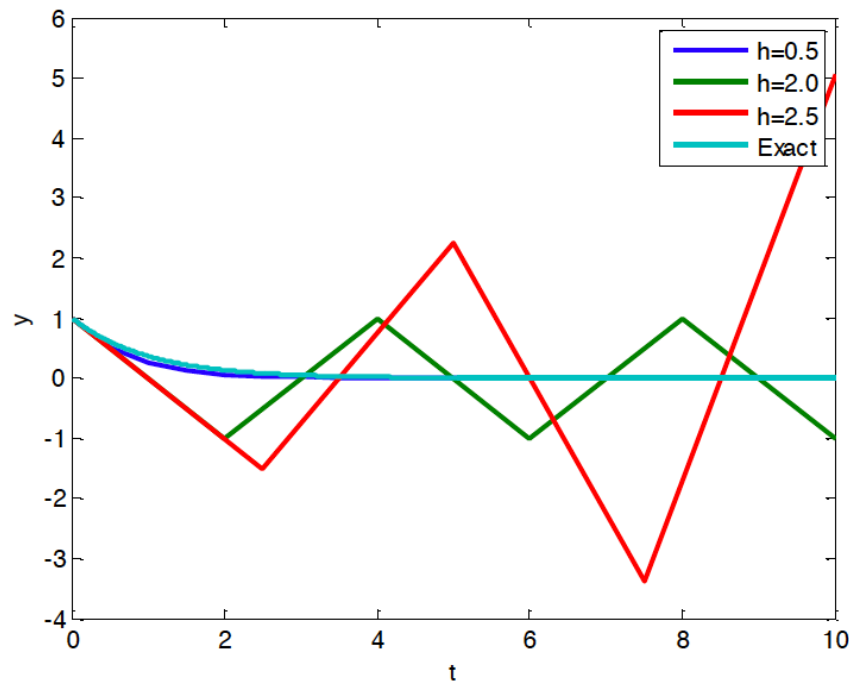


For pure imaginary λ , Forward Euler method is always (unconditionally) unstable!

Stability analysis: forward Euler method

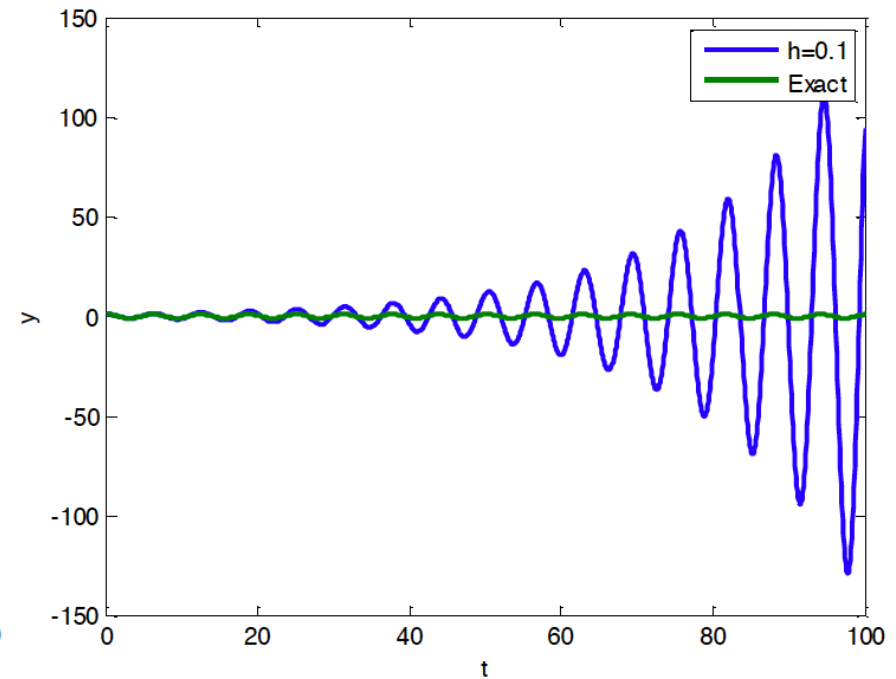
Example 1:

$$y' + y = 0; y_0 = 1$$



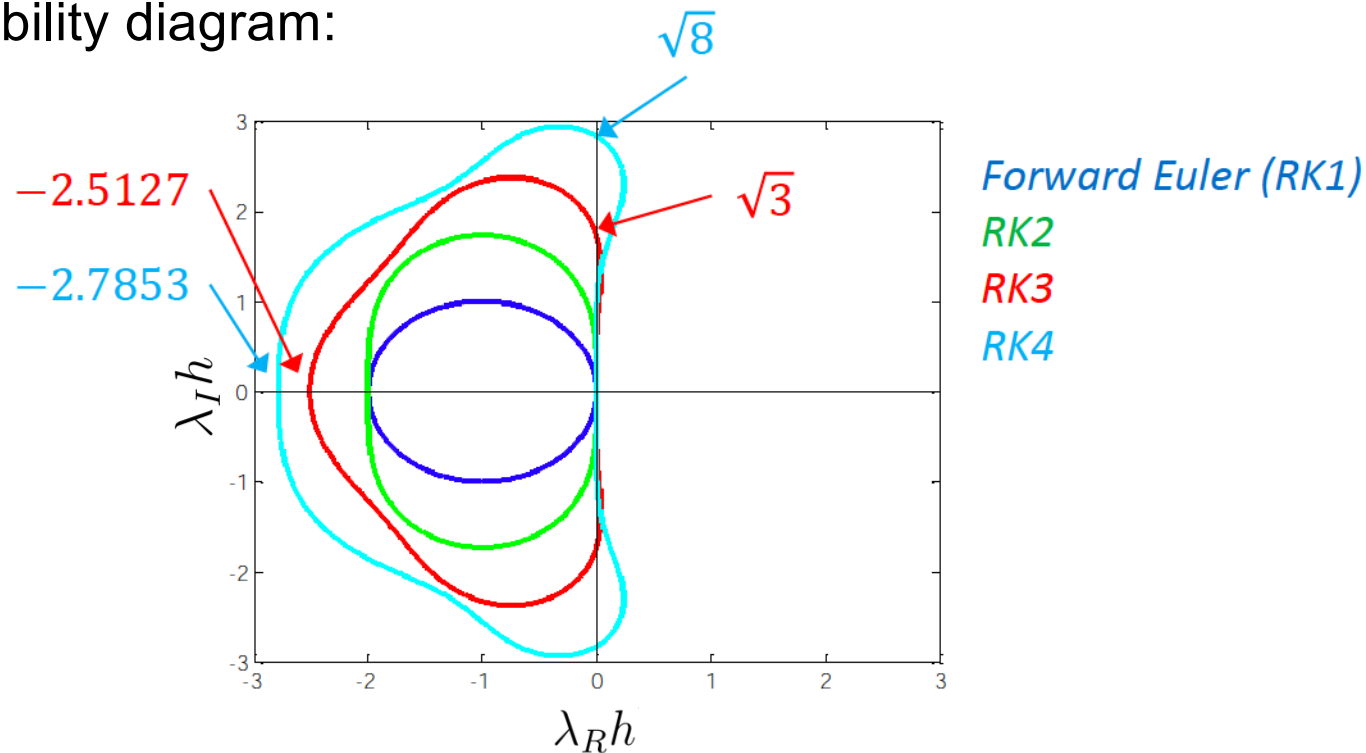
Example 2:

$$y' + iy = 0; y_0 = 1$$



Stability analysis: Runge-Kutta methods

Stability diagram:



Higher order algorithm also exhibit better stability.

RK3 and above are conditionally stable for pure imaginary λ .

Stability for system of ODEs

For a system of ODEs: $\frac{d\mathbf{y}}{dx} = \mathbf{f}(x, \mathbf{y})$

Locally, it could be approximated as $\frac{d(\mathbf{y} - \mathbf{y}_0)}{dx} \approx A(\mathbf{y} - \mathbf{y}_0)$ where $A = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right|_{\mathbf{y}_0}$

In the usual case, A can be diagonalized through a similarity transformation:

$$A = Q^T \Lambda Q \quad \text{where } Q \text{ is an orthogonal matrix}$$

One can transform variables $\mathbf{z} \equiv Q(\mathbf{y} - \mathbf{y}_0)$ and equivalently, we are solving

$$\frac{d\mathbf{z}}{dx} = \Lambda \mathbf{z}$$

Stability requires **stability criteria for h is satisfied for ALL eigenvalues.**

Stiff systems

In a system of equations, we have seen that h is constrained by the largest absolute eigenvalues, so that typically $h|\lambda|_{\max} \leq 1$.

Consider the following:

$$\begin{aligned} u' &= 998u + 1998v \\ v' &= -999u - 1999v \end{aligned} \quad \text{with} \quad \begin{aligned} u(0) &= 1 \\ v(0) &= 0 \end{aligned}$$

The solution is:

$$\begin{aligned} u &= 2e^{-x} - e^{-1000x} \\ v &= -e^{-x} + e^{-1000x} \end{aligned}$$

Suppose x denotes time, there are **two dramatically different timescales** over which the values decay.

Even the solution varies very slowly after the initial rapid decay, stability requirement still demands using tiny step size.

An ODE in such situation is said to be **stiff**.

Solving stiff ODEs

Need algorithms with much broader stability ranges, e.g., those that are **A-stable**.

The failure of explicit methods suggests we need to go **implicit**. The prototype of such methods is the **backward Euler method**:

For standard ODE $\frac{dy}{dx} = f(x, y)$, $y(x_0) = y_0$

Forward Euler: $y_{n+1} = y_n + hf(x_n, y_n)$

Both are 1st order accurate.

Backward Euler: $y_{n+1} = y_n + hf(x_{n+1}, y_{n+1})$

There is also the **Trapezoid method** (a.k.a. Crank-Nicholson):

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})]$$

2nd order accurate

Stability analysis: backward Euler method

Consider the model problem: $\frac{dy}{dx} = \lambda y$

Backward Euler method yields: $y_{n+1} = y_n + h y'_{n+1} = y_n + \lambda h y_{n+1}$

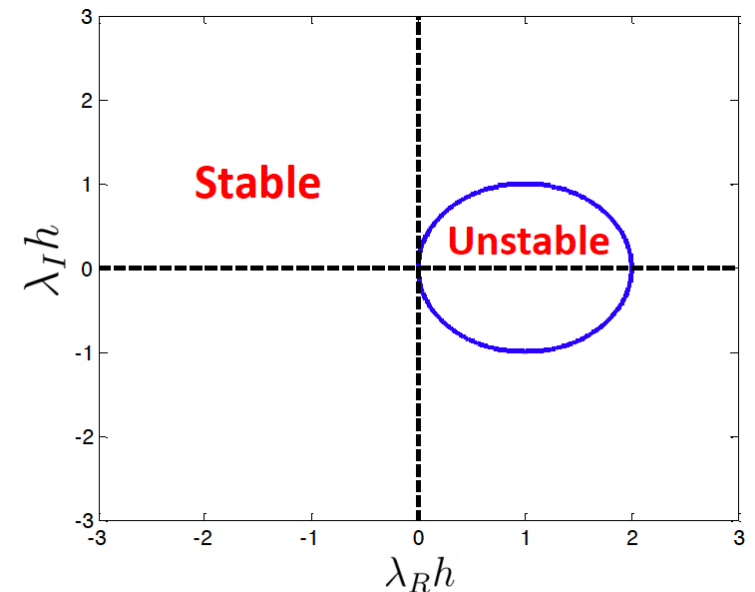
therefore, $y_{n+1} = (1 - \lambda h)^{-1} y_n$

Stable iff $\left| \frac{1}{1 - \lambda h} \right| \leq 1$

or $(1 - \lambda_R h)^2 + (\lambda_I h)^2 \geq 1$

The method is **A-stable** (unconditionally stable)!

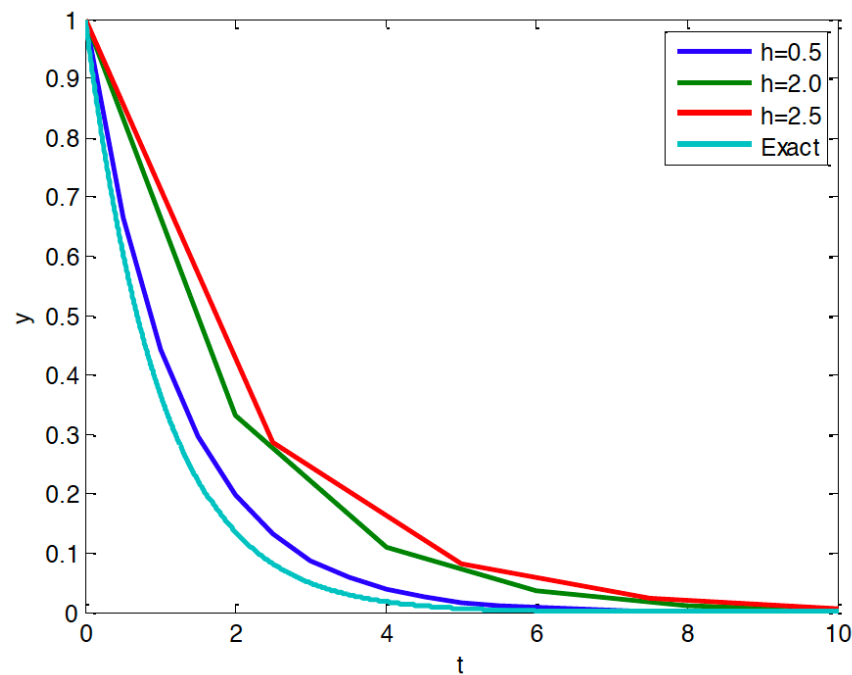
Minor issue: also stable in some regions that should grow exponentially.



Stability analysis: backward Euler method

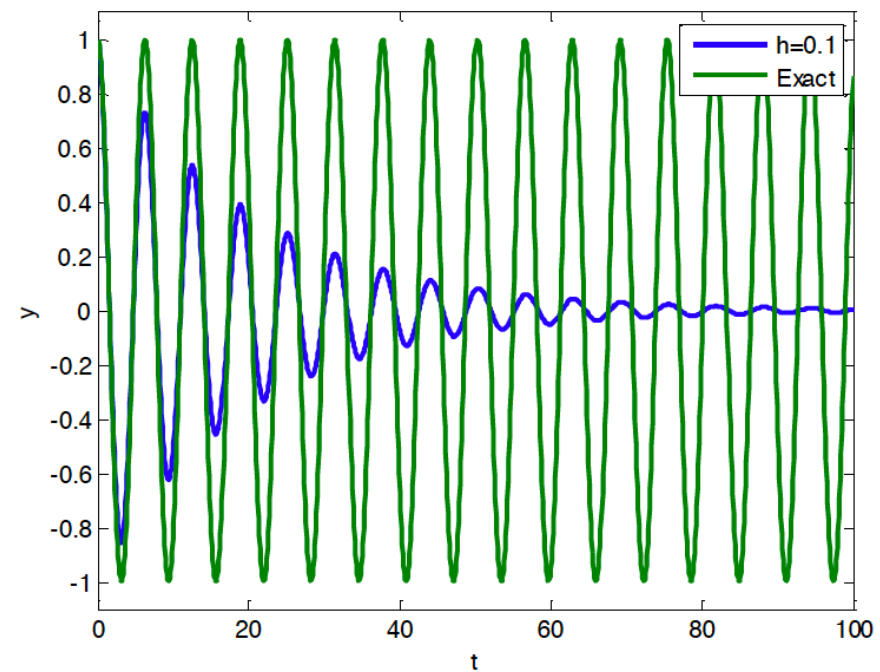
Example 1:

$$y' + y = 0; y_0 = 1$$



Example 2:

$$y' + iy = 0; y_0 = 1$$



Stability analysis: Crank-Nicholson method

Consider the model problem: $\frac{dy}{dx} = \lambda y$

Crank-Nicholson method yields: $y_{n+1} = y_n + \frac{\lambda h}{2}(y_n + y_{n+1})$

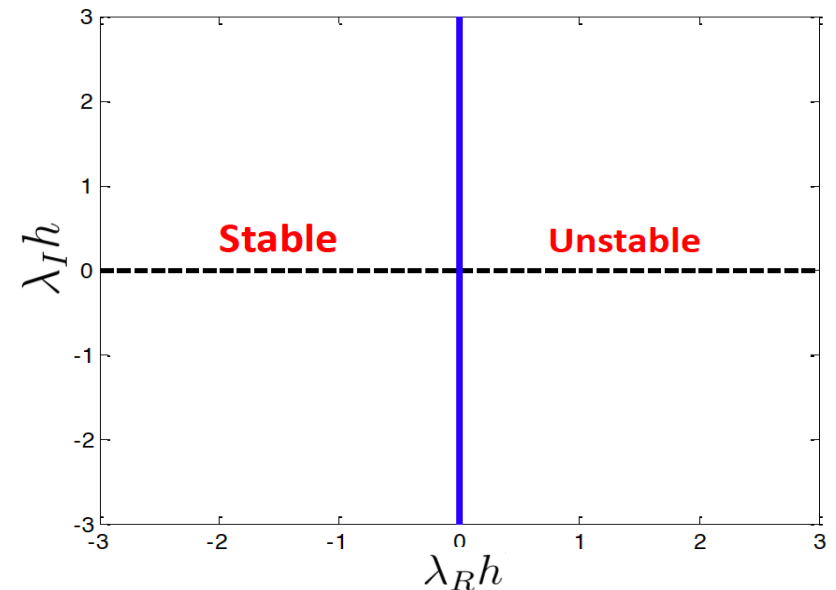
$$\text{therefore, } y_{n+1} = \frac{1 + \lambda h/2}{1 - \lambda h/2} y_n$$

$$\text{Stable iff } \left| \frac{1 + \lambda h/2}{1 - \lambda h/2} \right| \leq 1$$

$$\begin{aligned} \text{or } (1 + \lambda_R h/2)^2 + (\lambda_I h/2)^2 \\ \leq (1 - \lambda_R h/2)^2 + (\lambda_I h/2)^2 \end{aligned}$$

$$\text{or } \lambda_R \leq 0$$

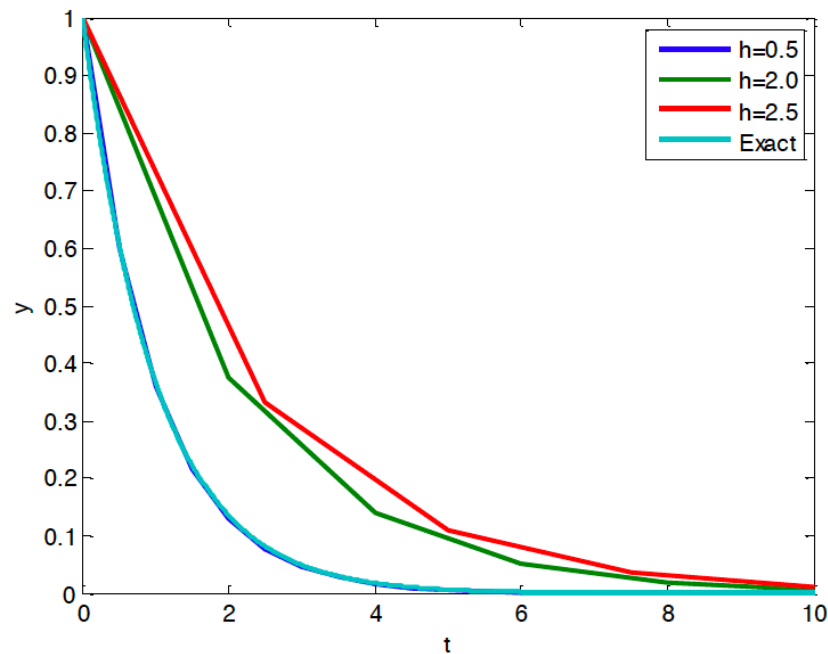
The method is also **A-stable**!



Stability analysis: Crank-Nicholson method

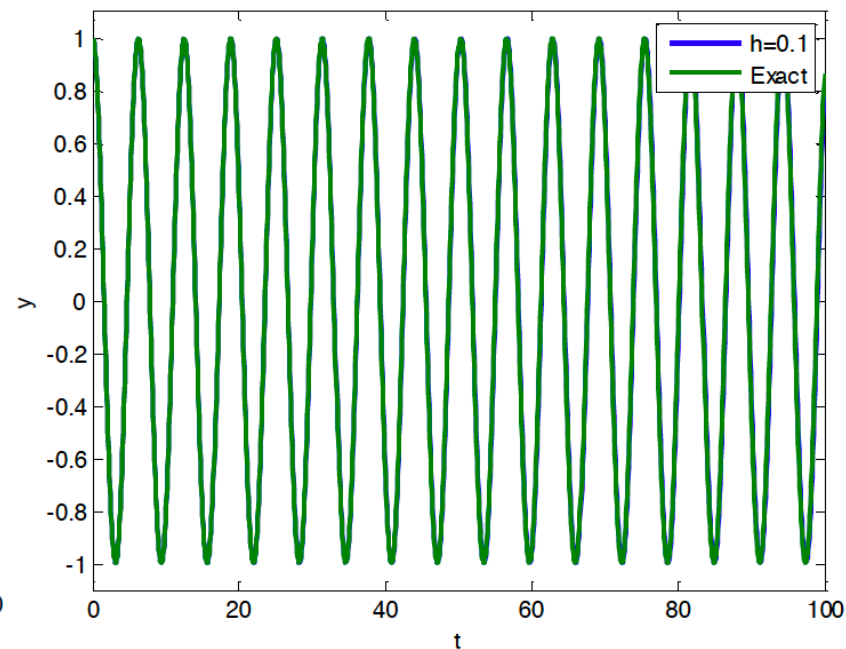
Example 1:

$$y' + y = 0; y_0 = 1$$



Example 2:

$$y' + iy = 0; y_0 = 1$$



No amplitude error!

There is small phase error.


General stiff system of ODEs

Implicit methods inevitably requires root-finding.

For backward Euler, we must solve

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x_{n+1}, \mathbf{y}_{n+1}) \text{ , which is generally nonlinear.}$$

Most commonly, we linearize the equations (i.e., Newton method)

$$\mathbf{y}_{n+1} \approx \mathbf{y}_n + h \left[\mathbf{f}(x_{n+1}, \mathbf{y}_n) + \frac{\partial \mathbf{f}}{\partial \mathbf{y}} (\mathbf{y}_{n+1} - \mathbf{y}_n) \right]$$


Jacobi matrix

The solution is $\mathbf{y}_{n+1} = \mathbf{y}_n + h \left[1 - h \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \right]^{-1} \mathbf{f}(\mathbf{y}_n)$

While there is no guarantee, this approach is usually stable.

Even higher-order

Generally speaking, implicit methods improves stability at the cost of accuracy. Do not often go beyond 2nd order.

For many applications, backward Euler and Crank-Nicholson are sufficient (caution that Crank-Nicholson may also become unstable in non-linear problems).

Some applications benefit from higher-order implicit schemes

- Kaps-Rentrop (Rosenbrock) methods
Implicit extensions of Runge-Kutta methods.
- Bader-Deuflhard method
Implicit extensions of Bulirsch-Stoer methods.
- Backward differentiation formulas (BDF)
Implicit extensions of multistep methods

Backward differentiation formulas

General form: $hf_{n+1} = \sum_{j=0}^k \alpha_{kj} y_{n+1-j}$

$$y_{n+1} - y_n = hf_{n+1} + O(h^2) \quad \text{BDF1}$$

$$y_{n+1} - \frac{4}{3}y_n + \frac{1}{3}y_{n-1} = \frac{2}{3}hf_{n+1} + O(h^3) \quad \text{BDF2}$$

$$y_{n+1} - \frac{18}{11}y_n + \frac{9}{11}y_{n-1} - \frac{2}{11}y_{n-2} = \frac{6}{11}hf_{n+1} + O(h^4) \quad \text{BDF3}$$

$$y_{n+1} - \frac{48}{25}y_n + \frac{36}{25}y_{n-1} - \frac{16}{25}y_{n-2} + \frac{3}{25}y_{n-3} = \frac{12}{25}hf_{n+1} + O(h^5) \quad \text{BDF4}$$

$$y_{n+1} - \frac{300}{137}y_n + \frac{300}{137}y_{n-1} - \frac{200}{137}y_{n-2} + \frac{75}{137}y_{n-3} - \frac{12}{137}y_{n-4} = \frac{60}{137}hf_{n+1} + O(h^6) \quad \text{BDF5}$$

BDF1,2 are A-stable, BDF3,4,5 are close to A-stable.

BDF methods are highly popular for solving stiff systems.

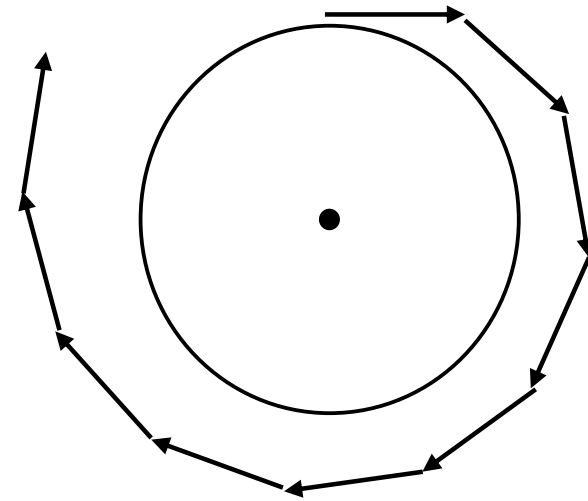
Geometric integrators

Consider simple Keplerian motion:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v} \quad , \quad \frac{d\mathbf{v}}{dt} = -\frac{GM}{r^3}\mathbf{r}$$

Using Forward Euler, we obtain

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{v}_n h \quad , \quad \mathbf{v}_{n+1} = \mathbf{v}_n + -\frac{GM}{|\mathbf{x}_n|^3}\mathbf{x}_n h$$



Clearly, the particle spirals out, without closing the orbit nor conserving energy.

The situation will improve when using higher-order integrators, but due to truncation error, there is no guarantee for closed orbit or energy conservation.

Geometric integrators can preserve certain geometric properties of the system.

Hamiltonian mechanics

Orbital motion is more fundamentally described by **Hamiltonian mechanics**:

$$H(\mathbf{p}, \mathbf{q}) = \frac{1}{2m}p^2 + \Phi(\mathbf{q})$$

Equation of motion is given by $\frac{d\mathbf{p}}{dt} = -\frac{\partial H}{\partial \mathbf{q}}, \quad \frac{d\mathbf{q}}{dt} = \frac{\partial H}{\partial \mathbf{p}}$

The flow of trajectories in a Hamiltonian system is strongly constrained: evolution from $(\mathbf{p}_t, \mathbf{q}_t)$ to $(\mathbf{p}_{t+\tau}, \mathbf{q}_{t+\tau})$ **preserves phase-space volume** (**Liouville's theorem**).

Define the Jacobian: $J_{t,\tau} \equiv \frac{\partial(\mathbf{p}_{t+\tau}, \mathbf{q}_{t+\tau})}{\partial(\mathbf{p}_t, \mathbf{q}_t)} \Rightarrow \det(J)=1.$

Another important geometric property is **time-reversibility**.

Examples:

Forward Euler: $x_{n+1} = x_n + hv_n$, $v_{n+1} = v_n - h\Phi'(x_n)$

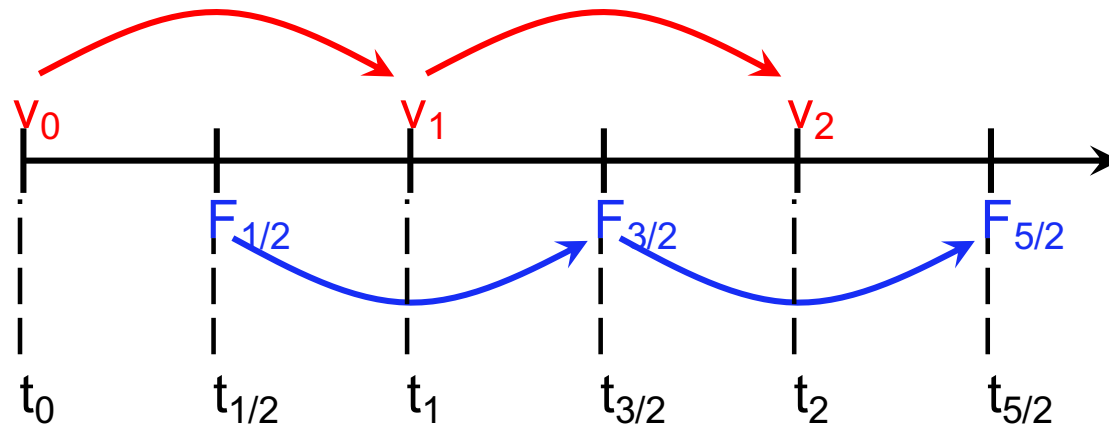
$$\Rightarrow \left| \frac{\partial(x_{n+1}, v_{n+1})}{\partial(x_n, v_n)} \right| = \begin{vmatrix} 1 & h \\ -\Phi''(x_n) & 1 \end{vmatrix} \neq 1$$

Modified Euler: $x_{n+1} = x_n + hv_n$, $v_{n+1} = v_n - h\Phi'(x_{n+1})$

$$\Rightarrow \left| \frac{\partial(x_{n+1}, v_{n+1})}{\partial(x_n, v_n)} \right| = \begin{vmatrix} 1 & h \\ -h\Phi''(x_{n+1}) & 1 - h^2\Phi''(x_{n+1}) \end{vmatrix} = 1$$

Neither of the above are **time-reversible**.

Leapfrog integrator



$$x' = x_n + \frac{h}{2}v_n, \quad v_{n+1} = v_n - h\Phi'(x'), \quad x_{n+1} = x' + \frac{h}{2}v_{n+1}$$

$$\Rightarrow \left| \frac{\partial(x_{n+1}, v_{n+1})}{\partial(x_n, v_n)} \right| = \begin{vmatrix} 1 - \frac{h^2}{2}\Phi''(x') & h - \frac{h^3}{4}\Phi''(x') \\ -h\Phi''(x_{n+1}) & 1 - \frac{h^2}{2}\Phi''(x_{n+1}) \end{vmatrix} = 1$$

It is also **time-reversible**.

Symplectic integrators

Integration scheme for [Hamiltonian systems](#), whose mathematical properties are characterized by [symplectic geometry](#).

The Hamiltonian is modified to accommodate discrete integration steps:

Modified Euler: $H(v, x) = \frac{1}{2}v^2 + \Phi(x)\delta_h(t)$

Leapfrog: $H(v, x) = \frac{1}{2}v^2 + \Phi(x)\delta_h(t - h/2)$

where $\delta_h(t) \equiv h \sum_{j=-\infty}^{\infty} \delta(t - jh)$

The difference from the original Hamiltonian is the source of truncation error.

Forward Euler, Runge-Kutta integrators, etc. are NOT symplectic integrators.

Symplectic integrators

Symplectic integrators can be constructed by splitting:

$$H_A \equiv \frac{1}{2}v^2, \quad H_B \equiv \Phi(x)$$

Drift

Kick

Modified Euler: $A_h B_h(x_0, v_0)$ or $B_h A_h(x_0, v_0)$

Leapfrog: $A_{\frac{h}{2}} B_h A_{\frac{h}{2}}(x_0, v_0)$ or $B_{\frac{h}{2}} A_h B_{\frac{h}{2}}(x_0, v_0)$

Can be generalized to higher order (e.g., 4th order Yoshida integrator).

Integration of few-body systems (e.g., solar system dynamics) often involves the [Wisdom-Holman \(1991\) integrator](#).

Analogous integrators exist for integrating particle orbits in electromagnetic field (the [Boris integrator](#)).

Outline

- Initial value problem

 - Fundamentals

 - Runge-Kutta methods

 - Bulirsch-Stoer method

 - Multistep methods (optional)

 - Stability analysis

 - Stiff systems

 - Geometric integrators

- Boundary value problem

 - Shooting method

 - Relaxation method

Two-point boundary value problem

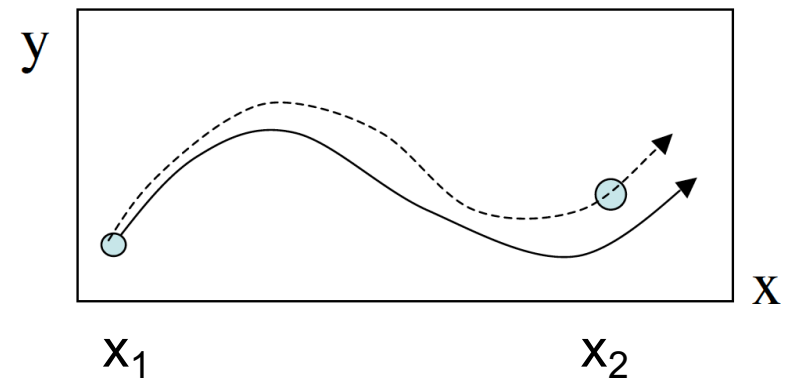
Consider a vector ODE, where components of y are known at two boundaries:

$$\frac{dy_i(x)}{dx} = g_i(x, y_1, \dots, y_N), \quad i = 1, \dots, N$$

$$B_{1j}(x_1, y_1, \dots, y_N) = 0, \quad j = 1, \dots, n_1$$

$$B_{2j}(x_2, y_1, \dots, y_N) = 0, \quad j = 1, \dots, n_2$$

where $n_1 + n_2 = N$.



Crucial difference with initial value problem is that the boundary conditions at either of the two boundaries are insufficient to determine a solution.

Iterative solutions are unavoidable.

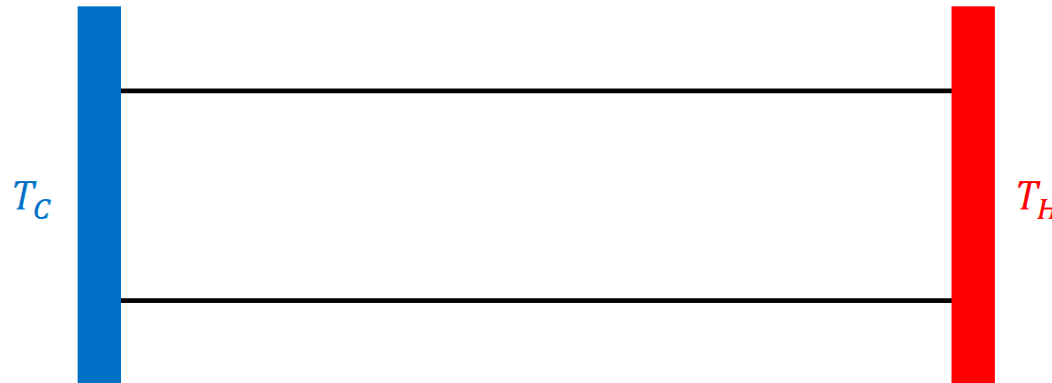
Example: equilibrium temperature distribution

In equilibrium, the equation of heat transfer on a rod reads

$$\frac{d}{dx} \left[\kappa(x) \frac{dT}{dx} \right] = 0 .$$

It should be supplied with two boundary conditions at the two ends.

$$T(x_L) = T_c , \quad T(x_R) = T_H .$$



Example: equation of stellar structure

Equations governing stellar structure derive from force balances:

$$\begin{aligned} \frac{dP}{dr} &= -\frac{GM_r}{r^2} \rho, \\ \frac{dM_r}{dr} &= 4\pi r^2 \rho. \end{aligned} \quad \Rightarrow \quad \frac{1}{r^2} \frac{d}{dr} \left(\frac{r^2}{\rho} \frac{dP}{dr} \right) = -4\pi G \rho.$$

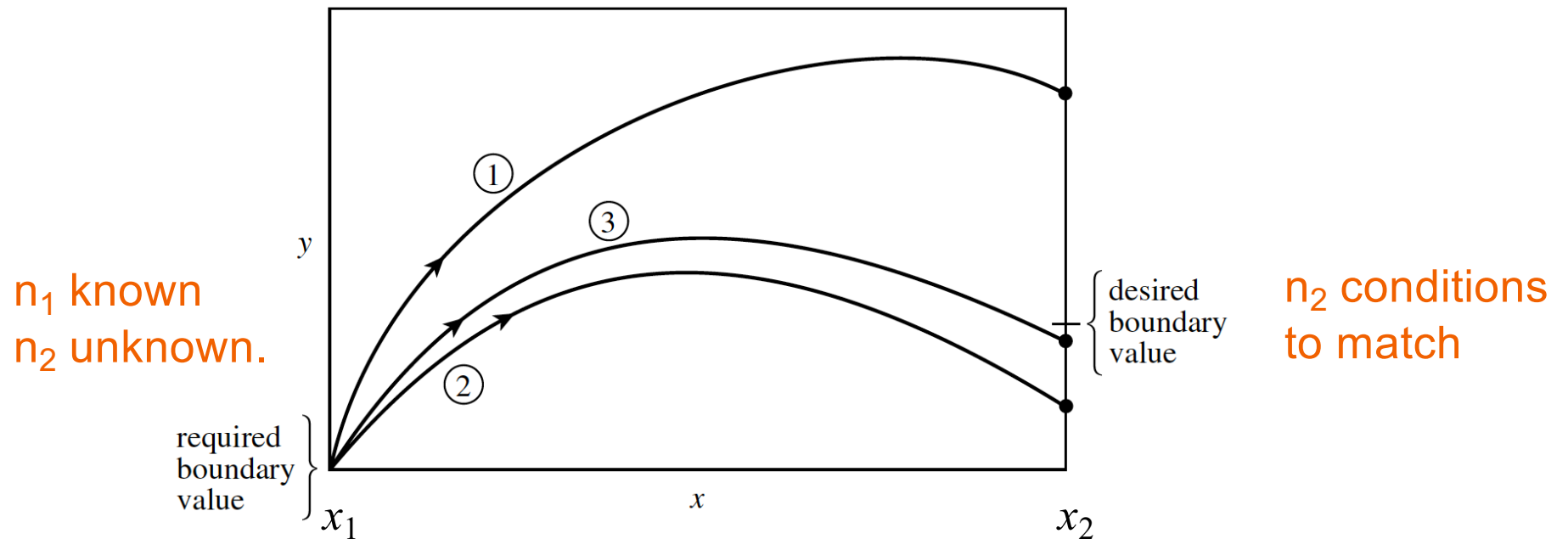
A polytropic equation of state ($P=K\rho^{(n+1)/n}$) closes the system, resulting in the so-called **Lane-Emden equation**:

$$\frac{1}{\xi^2} \frac{d}{d\xi} \left(\xi^2 \frac{d\theta}{d\xi} \right) = -\theta^n, \text{ where } \rho=\rho_0\theta^n, r=\alpha\xi$$

The **boundary conditions** are: $\theta = \theta_0$, $\frac{d\theta}{d\xi} = 0$ at $\xi = 0$ and $\theta = 0$ at $\xi = \xi_*$

Given the stellar radius r_* (we know ξ_* but not θ_0), what is its density profile?

Shooting method



Guess all variables at x_1 satisfying the n_1 boundary conditions.

Integrate to x_2 using your favorite method [as an initial value problem](#).

Usually there is a mismatch with the other n_2 boundary conditions.

Adjust the initial guesses iteratively to improve, as a root-finding problem.

Shooting method

Let the initial guesses of the unknown boundary values at x_1 be \mathbf{V} (vector with n_2 components), i.e., the x_1 boundary can be represented as

$$y_i(x_1) = y_i(x_1; V_1, \dots, V_{n_2})$$

Integrating the system from x_1 to x_2 , we can define a discrepancy vector at x_2 be \mathbf{F} (n_2 components), simplest choice being

$$F_j = B_{2j}(x_2, \mathbf{y}(x_2, \mathbf{V}))$$

We want to find \mathbf{V} that is the root of \mathbf{F} .

Generally, one can iteratively improve the estimate of \mathbf{V} by the [Newton method](#) (or Broyden's method), that is, to find $\delta\mathbf{V}$ such that $\mathbf{F} + \delta\mathbf{F} \simeq 0$.

$$\delta\mathbf{V} = -J^{-1}\delta\mathbf{F}, \quad \text{where } J_{kl} = \frac{\partial F_k}{\partial V_l}, \quad (k, l = 1, \dots, n_2)$$

Then start a new iteration using $\mathbf{V}^{\text{new}} = \mathbf{V}^{\text{old}} + \delta\mathbf{V}$

Shooting method

The Jacobian may not be analytically known.

Use finite difference approximations (most easy to implement):

$$\frac{\partial F_i}{\partial V_j} \approx \frac{F_i(V_0, \dots, V_j + \Delta V_j, \dots) - F_i(V_0, \dots, V_j, \dots)}{\Delta V_j}$$

This means n_2+1 integrations of N coupled ODEs per iteration.

For linear system, one iteration will be enough.

For non-linear system, require many (say M) iterations: much more expensive (a factor $\sim n_2 M$) than solving initial values problems. Efficient integrator needed.

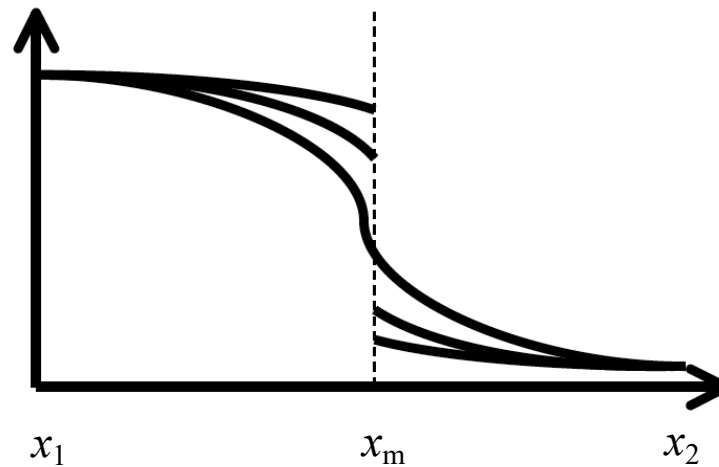
Even then, still need good “guesses”, otherwise results may not converge.

Shooting method (variant version)

Sometimes, solution at x_2 can be extremely sensitive to initial guesses at x_1 .

Instead, shoot from both ends and match the solution at interior point.

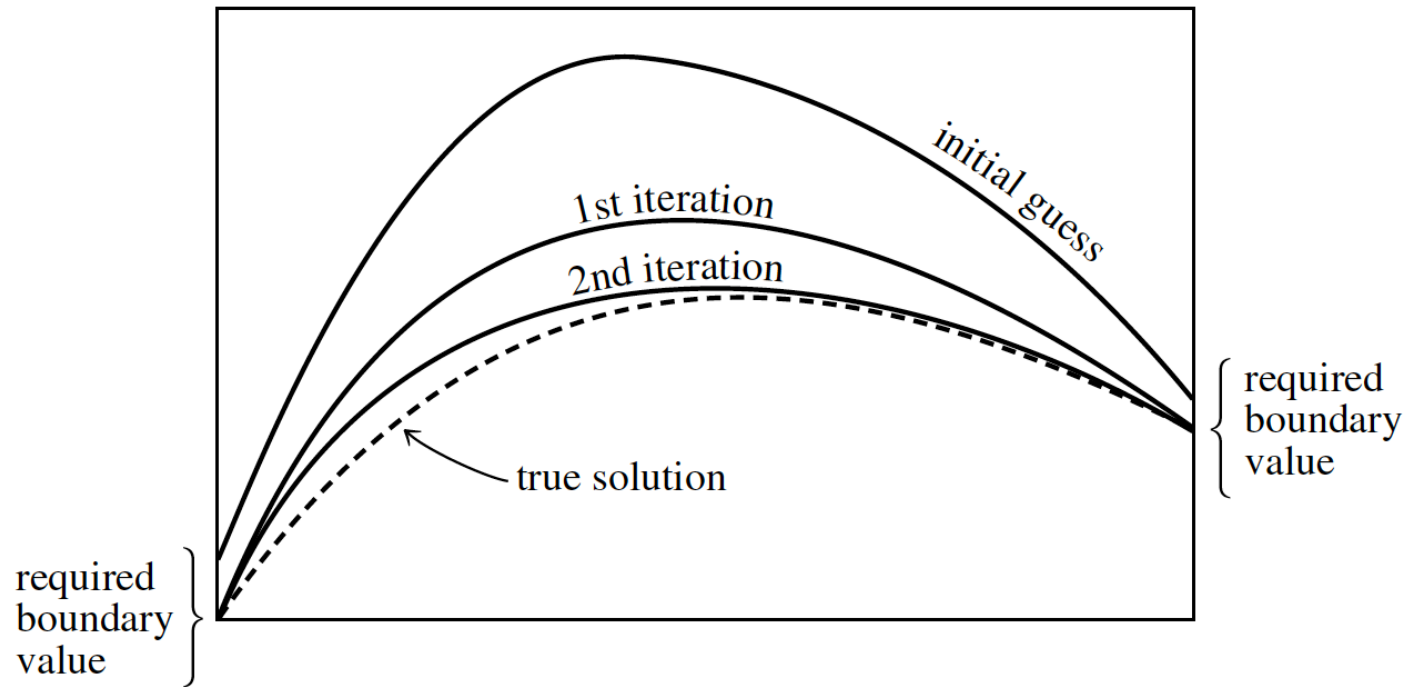
V_1 specified at x_1
(n_2 components)



V_2 specified at x_2
(n_1 components)

Require y to be continuous
($N=n_1+n_2$ conditions) at x_m .

Relaxation method



First, guess the complete solution over the entire domain.
Then iteratively improve the solution at all points at once.
Lead to large linear systems to be solved at each iteration.

Relaxation method

Consider $\frac{d\mathbf{y}(x)}{dx} = \mathbf{g}(x, \mathbf{y})$ (N coupled ODEs)

We solve this equation on M+1 grid points, denoted by x_0, \dots, x_M .

At each mesh interval $k=1, \dots, M$, convert the ODE into a finite difference form:

$$\mathbf{y}_k - \mathbf{y}_{k-1} = (x_k - x_{k-1})\mathbf{g}[(x_k + x_{k-1})/2, (\mathbf{y}_k + \mathbf{y}_{k-1})/2]$$

More schematically, we seek to solve (N equations at each midpoint):

$$\mathbf{E}_k \equiv \mathbf{y}_k - \mathbf{y}_{k-1} - (x_k - x_{k-1})\mathbf{g}[(x_k + x_{k-1})/2, (\mathbf{y}_k + \mathbf{y}_{k-1})/2] = 0$$

At x_0 there are n_1 boundary conditions and at x_M there are n_2 boundary conditions.

They are described by:

$$\mathbf{E}_0 = \mathbf{B}_1(x_0, \mathbf{y}_0) \quad (n_1 \text{ equations})$$
$$\mathbf{E}_{M+1} = \mathbf{B}_2(x_M, \mathbf{y}_M) \quad (n_2 \text{ equations})$$

Relaxation method

We are talking about a system of $M \cdot N$ equations from the ODEs, and N equations from the boundary conditions, for $(M+1) \cdot N$ variables y_{jk} .

Iteration using the Newton method by Taylor expanding the equations to 1st order. Every iteration solves a large linear system of the form:

boundary condition at x_0	X X X X X	$n_1 x N$		V	B	$(M+1)N$ rows
	X X X X X		V	B		
	X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X	$N \times 2N$	V	B		
Jacobian from the ODEs	$N \times 2N$		X X X X X X X X X X	V	B	
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X	V	B			
	X X X X X X X X X X	$N \times 2N$	V	B		
boundary condition at x_M	$n_2 x N$		X X X X X X X X X X	V	B	
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X		V	B		
	X X X X X X X X X X	V	B			

Practical issues of implementation

Need good linear algebra package: almost all work is solving the linear system.

Storage: Don't write the entire matrix, but only store elements of individual blocks.

Given the finite difference equations in our example, the matrix can be solved efficiently by Gaussian elimination followed by backward substitution specifically tuned for this system.

This is usually better than applying general sparse matrix solvers.

See Numerical recipes for detailed implementation.

Good initial guesses: otherwise may not converge.

Can use more stencils for more accurate differentiation formula, or use variable grid spacings, at the cost of more complex matrix.

Summary

■ Initial value problems

Higher-order methods: Runge-Kutta, Bulirsch-Stoer, and multistep

Stiff systems: need implicit solvers for stability, BDF is good choice.

Hamiltonian systems: symplectic integrators to preserve geometric properties.

■ Boundary value problems

Shooting method: convert to initial value problems followed by iteration to match the boundary conditions.

Relaxation method: guess the entire solution and iteratively improve towards the final solution by solving large linear systems.