PH5720

In this unit you will learn the determination of the numerical solution of differential equations using different approaches. To reduce the computation cost and achieve accurate numerical solution with small error, it is always advisable to compute the differential equation with larger stability region. You will study to solve the first order differential equations. There are several examples in the slide. You can write small program and see the numerical solution of the differential equation. The unit will focus on the following subsection.

- Ordinary Differential Equations: Initial Value Problems
- Ordinary Differential Equations: Boundary value Problems
- Partial Differential Equations

Numerical methods for ODEs

There are many different methods for solving ODEs, most of which are of one of following types

Euler - trapezoidal predictor-corrector

Taylor series

Runge-Kutta

Extrpolation

Multistep

Multivalue

We briefly consider each of these types of methods

Euler - trapezoidal predictor-corrector:

Euler - trapezoidal predictor-corrector method

Higher-order accuracy can be achieved by averaging Euler and backward Euler methods to obtained implicit trapezoidal predictor-corrector method.

Euler methods

$$x_{k+1} = x_k + h_k f(t_k, x_k)$$

backward Euler methods

$$x_{k+1} = x_k + h_k f(t_{k+1}, x_{k+1})$$

$$x_{k+1} = x_k + h_k \left(f(t_k, x_k) + f(t_{k+1}, x_{k+1}) \right) \frac{1}{2}$$

Stability condition

Accuracy

Euler - trapezoidal predictor-corrector method

Higher-order accuracy can be achieved by averaging Euler and backward Euler methods to obtained implicit trapezoidal predictor-corrector method.

$$x_{k+1} = x_k + h_k\big(f(t_k, x_k) + f(t_{k+1}, x_{k+1})\big)\,\frac{1}{2}$$

To determine its stability and accuracy, we apply it to scalar ODE $x' = \lambda x$, obtaining

$$x_{k+1} = x_k + h\,(\lambda x_k + \lambda x_{k+1})\,\frac{1}{2}$$

$$\left(1 - h\frac{\lambda}{2}\right)x_{k+1} = \left(1 + h\frac{\lambda}{2}\right)x_k$$

$$x_k = \left(\frac{1 + h\frac{\lambda}{2}}{1 - h\frac{\lambda}{2}}\right)^k x_0$$

Euler - trapezoidal predictor-corrector method

Higher-order accuracy can be achieved by averaging Euler and backward Euler methods to obtained implicit trapezoidal predictor-corrector method.

$$x_{k+1} = x_k + h_k\big(f(t_k, x_k) + f(t_{k+1}, x_{k+1})\big)\,\frac{1}{2}$$

To determine its stability and accuracy, we apply it to scalar ODE $x' = \lambda x$, obtaining

$$x_k = \left(\frac{1 + h\frac{\lambda}{2}}{1 - h\frac{\lambda}{2}}\right)^k x_0$$

method is stable if

$$\left|\frac{1 + h\frac{\lambda}{2}}{1 - h\frac{\lambda}{2}}\right| < 1$$

Euler - trapezoidal predictor-corrector method

Trapezoidal predictor-corrector method is stable if

$$\left|\frac{1+h\frac{\lambda}{2}}{1-h\frac{\lambda}{2}}\right| < 1$$

Which hold for any $h > 0$ when $Re(\lambda) < 0$

Thus trapezoidal predictor-corrector method is unconditionally stable.
Its growth factor

$$\frac{1+h\frac{\lambda}{2}}{1-h\frac{\lambda}{2}} = \left(1+h\frac{\lambda}{2}\right)\left(1+h\frac{\lambda}{2}+\left(h\frac{\lambda}{2}\right)^2+\left(h\frac{\lambda}{2}\right)^3+\cdots\cdots\right)$$

$$\frac{1+h\frac{\lambda}{2}}{1-h\frac{\lambda}{2}} = 1+h\lambda+\frac{(h\lambda)^2}{2}+\frac{(h\lambda)^3}{4}+\cdots$$

Agrees with expression of $e^{h\lambda}$ through terms of order h^2, so trapezoidal predictor-corrector method is **second order accurate**.

Euler - trapezoidal predictor-corrector method

Trapezoidal predictor-corrector method : **implicit method**.
unconditionally stable.

Not all implicit methods have this properties.

Implicit methods : generally have **larger stability regions**.
allowable step size is not always unlimited.

Implicitness alone is **not sufficient to guarantee stability**.

Stiff differential equations:

The computed solution is incredibly sensitive to the initial value, as each tiny perturbation results in a wildly different solution.

Stiff differential equations:

Asymptotically stable solutions **converge with time** and this convergence has the favorable property of **damping errors** in a numerical Solution.

But if the convergence of **solutions is too rapid**, then difficulties of a different type may arise.

Such an ODE is said to be **stiff**.

Stiff ODE corresponds to physical process whose components have **highly disparate time scales** or a process whose **time scale is very short** compared to the **interval** over which it is being studied.

System of ODEs $x' = f(t, x)$ is stiff if eigenvalues of its Jacobian matrix J_f differ greatly in magnitude.

Stiff differential equations:

System of ODEs $x' = f(t, x)$ is stiff if eigenvalues of its Jacobian matrix J_f differ greatly in magnitude.

There may be eigenvalues with

large negative real parts, corresponding to strongly damped, components of the solution,
or
relatively large imaginary parts, corresponding to rapidly oscillating components of the solution.

Some numerical methods are **inefficient for stiff ODEs** because rapidly varying component of solution forces very **small step sizes to maintain stability**.

Stiff differential equations:

Stability restriction depends on the **rapidly varying** component of the solution.
Accuracy restriction depends on the **slowly varying** component.

So the **step size may be much more severely restricted by stability than by the required accuracy**.

For example, **Euler's method is extremely inefficient** for solving a stiff equation because of severe stability limitation on step size.

Backward Euler method is suitable for stiff ODEs because of its unconditional stability.

Stiff ODEs need not be difficult to solve numerically provided a **suitable method**, generally implicit, is chosen.

Stiff differential equations: Example

Consider scalar ODE

$$x' = -100\,x + 100\,t + 101 \qquad x(0) = 1$$

General solution is $x(t) = 1 + t + ce^{-100\,t}$,
Particular solution satisfying initial condition is $x(t) = 1 + t$ (i.e. c = 0)

Since solution is linear, Euler method is theoretically exact for the problem.

However, to illustrate the effect of truncation or rounding errors, let us perturb the initial value slightly.

Stiff differential equations: Example

Consider scalar ODE

$$x' = -100\,x + 100\,t + 101 \qquad x(0) = 1$$

$x(t) = 1 + t + ce^{-100\,t}$, and $x(t) = 1 + t$ (i.e. c = 0)

However, to illustrate the effect of **truncation or rounding errors**, let us **perturb** the initial value **slightly**.

With step size $h = 0.1$, first few steps for given initial values are

t	0.0	0.1	0.2	0.3	0.4
Exact solution	1.00	1.10	1.20	1.30	1.40
Euler solution	0.99	1.19	0.39	8.59	−64.2
Euler solution	1.01	1.01	2.01	−5.99	67.0

Stiff differential equations: Example

Computed solution is incredibly **sensitive to initial value**, as each tiny perturbation results in **widely different solution**.

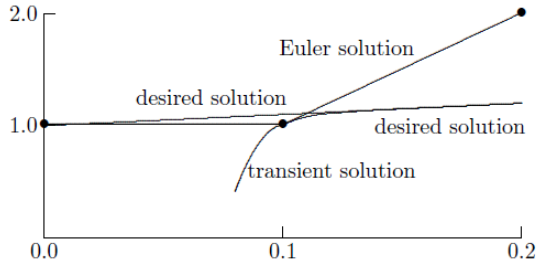
Any point deviating from desired particular solution, even by only small amount, lies on different solution, for which $c \neq 0$ and therefore rapid transient of general solution is present.

t	0.0	0.1	0.2	0.3	0.4
Exact solution	1.00	1.10	1.20	1.30	1.40
Euler solution	0.99	1.19	0.39	8.59	−64.2
Euler solution	1.01	1.01	2.01	−5.99	67.0

Stiff differential equations: Example

Euler's method bases its **projection on derivative** at current point, and resulting large value causes the numerical solution to **diverge radically from the desired solution**.

Jacobian for this ODE is $J_f = -100$, so the **stability condition** for Euler's method requires a step size $h < 0.02$, which we are **violating**.



Stiff differential equations: Example

Backward Euler method has no trouble solving this problem and is extremely **insensitive to the initial value**.

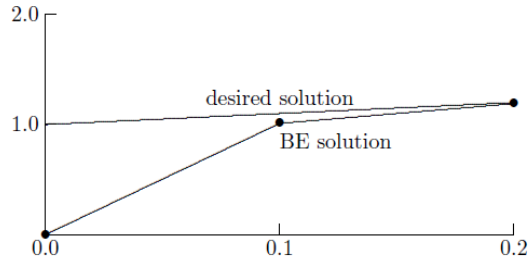
Even with a **very large perturbation** in the initial value, by using the derivative at the next point rather than the current point, the t**ransient is quickly damped out** and the backward Euler solution converges to the desired solution after only a few steps.

This behaviour is **consistent with the unconditional stability** of the backward Euler method for a stable solutions.

t	0.0	0.1	0.2	0.3	0.4
Exact solution	1.00	1.10	1.20	1.30	1.40
BE solution	0.00	1.01	1.19	1.30	1.40
BE solution	2.00	1.19	1.21	1.30	1.40

Stiff differential equations: Example

Backward Euler method has no trouble solving this problem and is extremely **insensitive to the initial value**.



t	0.0	0.1	0.2	0.3	0.4
Exact solution	1.00	1.10	1.20	1.30	1.40
BE solution	0.00	1.01	1.19	1.30	1.40
BE solution	2.00	1.19	1.21	1.30	1.40

Taylor series methods :

Taylor series methods :

Euler's method can be derived from a **Taylor series expansion**.

By **retaining more terms** in the Taylor series, we can generate higher-order single-step methods.

For example, retaining one additional term in the Taylor series

$$x(t+h) = x(t) + hx'(t) + \frac{h^2}{2}x''(t) + \frac{h^3}{6}x'''(t) + \cdots$$

Gives second order method

$$x_{k+1} = x_k + h_k x'_k + \frac{h_k^2}{2}x''_k$$

Taylor series methods :

By retaining more terms in the Taylor series, we can generate higher-order single-step methods.

This approach requires the computation of **higher derivatives** of x, which can be obtained by differentiating $x' = f(t,x)$ using the chain rule, e.g.,

$$x'' = f_t(t,x) + f_x(t,x)x'$$

$$x'' = f_t(t,x) + f_x(t,x)f(t,x)$$

where the subscripts indicate **partial derivatives** with respect to the given variable.

As the order increases, expressions for the derivatives rapidly become too complicated to be practical to compute, so Taylor series methods of **higher order** have **not often been used** in practice.

Taylor series methods : Example

Solve the nonlinear scalar ODE

$x' = 3t + x^2$ using the second-order Taylor series method

$x(0) = 1$ $\quad x'' = f_t(t,x) + f_x(t,x)f(t,x)$

$t_{k+1} = t_k + h$ $\quad x'' = 3 + 2x(3t + x^2)$

$t_0 = 0 \quad t_1 = 0.05 \quad h = 0.05$

$x' = 1$ $\quad x'' = 5$

$$x_{k+1} = x_k + h_k x'_k + \frac{h_k^2}{2} x''_k$$

$$x_1 = x_0 + h_0 x'_0 + \frac{h_0^2}{2} x''_0 = 1 + 0.05 + 0.00625 = 1.05625$$

$t_1 = 0.05 \quad t_2 = 0.1 \quad h = 0.05$

$x'_1 = 1.26566$ $\quad x''_1 = 5.6737$

$$x_2 = x_1 + h_0 x'_1 + \frac{h_0^2}{2} x''_1 = 1.05625 + 0.063 + 0.00709 = 1.12665$$

Taylor series methods : Example

Solve the nonlinear scalar ODE

$x'(t) = t + x$ using the second-order Taylor series method

$x(0) = 1 \; and \; h = 0.1$

Solve the nonlinear scalar ODE

$x'(t) = -2tx^2$ using the second-order Taylor series method

$x(0) = 1 \; and \; h = 0.25$

Solve the nonlinear scalar ODE

$x'(t) = 2t + 3e^x$ using the second-order Taylor series method

$x(0) = 0 \;\; and \; h = 0.1$

Runge-Kutta methods:

Runge-Kutta methods:

Runge-Kutta methods are **single-step methods** that are similar in motivation to Taylor series methods but do **not involve explicit computation of higher derivatives**.

Instead, Runge-Kutta methods simulate effect of higher derivatives by **evaluating f several times** between t_k and t_{k+1}

Simplest example is second order Heun's method

$$x_{k+1} = x_k + \frac{h_k}{2}(k_1 + k_2)$$

$$k_1 = f(t_k, x_k)$$

$$k_2 = f(t_k + h_k, x_k + h_k k_1)$$

Runge-Kutta methods:

Heun's method is analogous to implicit trapezoid method, but remains explicit by using Euler prediction $x_k + h_k k_1$ instead of $x(t_{k+1})$ in evaluating f at t_{k+1}.

Best known Runge-Kutta methods is classical fourth-order scheme

$$x_{k+1} = x_k + \frac{h_k}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(t_k, x_k)$$
$$k_2 = f\left(t_k + \frac{h_k}{2}, x_k + \frac{h_k}{2}k_1\right)$$
$$k_3 = f\left(t_k + \frac{h_k}{2}, x_k + \frac{h_k}{2}k_2\right)$$
$$k_4 = f(t_k + h_k, x_k + h_k k_3)$$

Runge-Kutta methods:

To proceed to time t_{k+1}, Runge-Kutta methods **require no history** of the solution prior to time t_k, which makes them self-starting at the beginning of the integration, and also makes it **easy to change the step size** during the integration.

These features also make Runge-Kutta methods **relatively easy to program**, which accounts in part for their popularity.

Unfortunaltely, classical Runge-Kutta methods **provide no error estimate** on which to base the choice of step size.

Fehlberg devised embedded Runge-Kutta methods that uses six function evaluations per step to produce 5^{th} – order and 4^{th} – order estimates of solution difference provides **estimate for local error**.

Runge-Kutta methods:

Another embedded Runge-Kutta methods is due to Dorman and Prince .

This approach has led to automatic Runge-Kutta solvers that are effective for many problems but **relatively inefficient for stiff problems** or when very high accuracy is required.

It is possible, however, to define implicit Runge-Kutta methods with superior stability properties that are suitable for **solving stiff ODEs**.

Runge-Kutta methods: Example

Solve the nonlinear scalar ODE

$x'(t) = -2tx^2$ using the fourth-order Runge-Kutta method

$x(0) = 1 \; and \; h = 0.25$

Solve the nonlinear scalar ODE

$x'(t) = \frac{x}{2} - \frac{t}{2}$ using the fourth-order Runge-Kutta method

$x(0) = 1 \; and \; h = 0.25$

Solve the nonlinear scalar ODE

$x'(t) = x - t^2$ using the fourth-order Runge-Kutta method

$x(0) = 1 \; and \; h = 0.1$

Extrapolation methods:

Extrapolation methods:

Extrapolation methods are based on the use of a **single-step method** to integrate the ODE over a given interval, $t_k \leq t \leq t_{k+1}$, using **several different step sizes** h_i, and yielding results denoted by $X(h_i)$.

This gives a discrete approximation to a function $X(h)$, where $X(0) = x(t_{k+1})$.

Interpolating polynomial or **rational function** $\hat{X}(h)$ is **fit** to these data, and $\hat{X}(0)$ is then taken as the approximation to $X(0)$.

Extrapolation methods are **capable of achieving very high accuracy**, but they tend to be much **less efficient** and **less flexible** than other methods for ODEs, so they are not often used unless extremely high accuracy is required and cost is not a significant factor.

Multistep methods:

Multistep methods:

Multistep methods use information at more than one previous point to estimate the solution at the next point.

Linear multistep methods have the form

$$x_{k+1} = \sum_{i=1}^{m} \alpha_i x_{k+1-i} + h \sum_{i=0}^{m} \beta_i f(t_{k+1-i}, x_{k+1-i})$$

Parameters α_i and β_i are determined by polynomial interpolation.

If $\beta_0 = 0$, method is explicit, but if $\beta_0 \neq 0$, method is implicit.

Implicit methods are usually more accurate and stable than explicit methods, but require starting guess for x_{k+1}.

Multistep methods:

Starting guess is conveniently supplied by an explicit method, so the explicit and implicit methods are used as a predictor-corrector pair.

One could use the corrector repeatedly (i.e., fixed-point iteration) until some convergence tolerance is met, but it may not be worth the expense.

So, a fixed number of corrector steps, often only one, may be used instead, giving a PECE (predict, evaluate, correct, evaluate) scheme.

Although it has no effect on the value of x_{k+1}, the second evaluation of f in a PECE scheme yields an improved value of x'_{k+1} for future use.

Multistep methods:

Alternatively, the nonlinear equation for x_{k+1} given by an implicit multistep method can be solved by Newton's method or other similar iterative method.

Again with a good starting guess supplied by the solution at the previous step or by an explicit multistep method.

Newton's method or a close variant of it is essential when using an implicit multistep method designed for stiff ODEs, as fixed-point iteration will fails to converge for reasonable step sizes.

`

Multistep methods: Example

Simplest second-order accurate explicit two-step method is

$$x_{k+1} = x_k + \frac{h}{2}(3x'_k - x'_{k-1})$$

Simplest second-order accurate implicit method is trapezoid method

$$x_{k+1} = x_k + \frac{h}{2}(x'_{k+1} + x'_k)$$

One of the most popular pairs of multistep methods is explicit fourth-order Adams-Bashforth predictor.

$$x_{k+1} = x_k + \frac{h}{24}(55x'_k - 59x'_{k-1} + 37x'_{k-2} - 9x'_{k-3})$$

And implicit fourth-order Adams-Moulton corrector.

$$x_{k+1} = x_k + \frac{h}{24}(9x'_{k+1} + 19x'_k - 5x'_{k-1} + x'_{k-2})$$

Multistep methods: Example

Backward differentiation formulas form another important family of implicit multistep methods.

Backward differentiation formulas methods, typified by popular formula

$$x_{k+1} = \frac{1}{11}(18x_k - 9x_{k-1} + 2x_{k-2}) + \frac{6h}{11}x'_{k+1}$$

are effective for solving stiff ODEs.

Multistep Adams methods:

Stability and accuracy of some Adams methods are summarized below,

Stability threshold indicates left endpoint of stability interval for scalar ODE

Error constant indicates coefficient of h^{p+1} term in local truncation error, where p is order of the method

Explicit methods		
order	Stability threshold	Error constant
1	-2	1/2
2	-1	5/12
3	-6/11	3/8
4	-3/10	251/720

Implicit methods		
order	Stability threshold	Error constant
1	-∞	-1/2
2	-∞	-1/12
3	-6	-1/24
4	-3	-19/720

Implicit methods are both more stable and more accurate than corresponding explicit methods of same order

Properties of multistep methods:

They are not self-starting, because several previous solution values of x_k are needed initially.

Changing step size is complicated, since the interpolation formulas are most conveniently based on equally spaced intervals for several consecutive points.

A good local error estimate can be determined from the difference between the predictor and the corrector.

They are relatively complicated to program.

Being based on interpolation, they can efficiently provide solution values at output points other than the integration points.

Properties of multistep methods:

Implicit methods have a much greater region of stability than explicit methods but must be iterated to convergence to enjoy this benefit fully

> PECE scheme is actually explicit, albeit in a somewhat complicated way.

Although implicit methods are more stable than explicit methods, they are still not necessarily unconditionally stable.

> No multistep method of greater than second order is unconditionally stable, even if it is implicit.

Properly designed implicit multistep method can be very effective for solving stiff ODEs.`

Multivalue methods:

Multivalue methods:

With multistep methods it is difficult to change step size, since past history of the solution is most easily maintained at equally spaced intervals.

Like multistep methods, multivalue methods are also based on polynomial interpolation, but they avoid some of the implementation difficulties associated with multistep methods.

One of the key ideas motivating multivalue methods is the observation that the interpolating polynomial itself can be evaluated at any point, not just at equally spaced intervals.

Equal spacing associated with multistep methods is simply an artifact of the representation as a linear combination of successive solution and derivative values with fixed weights.

Multivalue methods:

Another key idea in implementing multivalue methods is choosing the representation of the interpolating polynomial so that its parameters are essentially the values of the solution and one or more of its derivatives at a given point t_k.

This approach is analogous to using a Taylor, rather than Lagrange, representation of the polynomial.

The solution is advanced in time by a simple transformation of this representation from one point to the next, and it is easy to change step size.

Multivalue methods are mathematically equivalent to multistep methods, but are more convenient and flexible to implement, so most modern software implementations are based on them.

Multivalue methods: Example

Consider four-value method for solving scalar ODE

$$x' = f(t, x)$$

Instead of representing the interpolating polynomial by its value at four different points, we represent it by its value and first three derivatives at a single point t_k.

$$x_k = \begin{bmatrix} x_k \\ hx'_k \\ \frac{h^2}{2}x''_k \\ \frac{h^3}{6}x'''_k \end{bmatrix}$$

Multivalue methods: Example

By differentiating Taylor series

$$x(t_k + h) = x(t_k) + hx' + \frac{h^2}{2}x'' + \frac{h^3}{6}x'''$$

three times, we see that the corresponding values at the next point

$$t_{k+1} = t_k + h$$

are given approximately by the transformation.

$$\hat{x}_{k+1} = Bx_k$$

where matrix B is given by

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multivalue methods: Example

We have not yet used the differential equation, however, so we add a correction term to the foregoing prediction to obtain the final value

$$x_{k+1} = \hat{x}_{k+1} + \alpha r$$

where r is fixed 4-vector and

$$\alpha = h(f(t_{k+1}, x_{k+1}) - \hat{x}'_{k+1})$$

For consistency, i.e., for the ODE to be satisfied, we must have $r_2 = 1$;

Remaining three components of r can be chosen in various ways, resulting in different methods, analogous to the different choices of parameters in multistep methods.

Multivalue methods: Example

For example, four-value method with

$$r = \begin{bmatrix} \frac{3}{8} & 1 & \frac{3}{8} & \frac{1}{6} \end{bmatrix}^T$$

is equivalent to the implicit 4th-order Adams-Moulton method given earlier.

Multivalue methods:

It is easy to change the step size with a multivalue method: we need merely rescale the components of x_k to reflect the new step size.

It is also easy to change the order of the method simply by changing the components of r.

These two capabilities, combined with sophisticated tests and strategies for deciding when to change order and step size, have led to the development of very powerful and efficient software packages for solving ODEs based on variable order/variable-step methods.

Such routines are analogous to adaptive quadrature routines.

Multivalue methods:

Such routines automatically adapt to a given problem, varying the order and step size of the integration method as necessary to meet the user-supplied error tolerance efficiently.

Such routines often have options for solving either stiff or non-stiff problems, and some even detect stiffness automatically and select an appropriate method accordingly.

The ability to change order easily also obviates the need for special starting procedures. With a variable-order/variable-step method, one can simply start with a first-order method, which requires no additional starting values, and let the automatic order/step-size selection procedure increase the order as needed for the required accuracy.