

ODE's

1. Numerical solution of ODE's
2. Classical schemes for solving ODE's

ODE's

- The order of an ODE is the highest derivative that appears in the equation
- An ODE is linear if it is a linear function of the dependent variable and its derivatives

- First order, linear:

$$\frac{dy}{dx} = y + x^2$$

- non-linear:

$$\frac{dy}{dx} + y^3 + x^2 = c$$

- Second order, linear:

$$\frac{d^2 y}{dx^2} - y + x^2 = 5$$

- non-linear:

$$\left(\frac{d^2 y}{dx^2} \right)^2 - \frac{dy}{dx} + x^2 = 0$$

Reducing the order of ODE's

- If you have an ODE of second order or higher, you can always write it as a set of coupled, first order equations

- Given
$$\frac{d^2 y}{dx^2} + p(x) \frac{dy}{dx} + q(x)y = g(x)$$

- We can define $v(x)$ so the set

$$\frac{dv}{dx} + p(x)v + q(x)y = g(x)$$

$$v(x) = \frac{dy}{dx}$$

is equivalent to the original 2nd order equation

- This is an essential trick for numerical solution of ODE's

Example: harmonic oscillator

second order equation : $\frac{d^2 y}{dt^2} = -ky$

first order system : $\frac{dv}{dt} = -ky$

$$v(t) = \frac{dy}{dt}$$

initial conditions : $y(t = 0) = y_0, \quad v(t = 0) = v_0$

Example: gravitational force

second order system : $\frac{d^2 x}{dt^2} = -kx/(x^2 + y^2)^{3/2}$

$$\frac{d^2 y}{dt^2} = -ky/(x^2 + y^2)^{3/2}$$

x, y : position of object orbiting around mass at the origin

k : constant that depends on the mass of the object at origin

first order system : $\frac{dv_x}{dt} = -kx/(x^2 + y^2)^{3/2}$

$$\frac{dv_y}{dt} = -ky/(x^2 + y^2)^{3/2}$$

$$v_x(t) = \frac{dx}{dt}$$

$$v_y(t) = \frac{dy}{dt}$$

Numerical solution of ODE's

- A first order non-linear ODE can be written in the form

$$\frac{dy}{dt} = f(t, y) \qquad y(t_0) = y_0$$

where the independent variable t increases from an initial value t_0 . The analogous set of coupled ODE's can be written in vector form:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) \qquad \mathbf{y}(t_0) = \mathbf{y}_0$$

- These are called *initial value problems*. Given the initial conditions \mathbf{y}_0 we want to integrate this system forward in time to determine the value of $\mathbf{y}(t)$ at later times.

Finite Differences for ODE's

- We will concentrate on *finite difference methods*
- These methods divide the interval of time t over which we wish to integrate the equations into a discrete set of values $\{t_n\}$, where $n=0, 1, \dots, N$
- The intervals or timesteps between the various values t_n are denoted as $\{h_n\}$, where $h_n = t_n - t_{n-1}$

Numerical schemes for ODE's

- A scalar ODE algorithm can be written as

$$y_{n+1} = y_n + h_n F \left(h_n, \{y_i\}, \left\{ \frac{dy}{dt} \Big|_i \right\}, \dots \right)$$

where $i=0, 1, \dots, n+1$ indicates the various discrete steps at which the solution has been calculated. The numerical schemes that advances y_n is embodied in the function F

- The most useful numerical schemes depend on at most a few previous values, i.e. $i=n, n-1, n-2, \dots$

Accuracy: truncation errors

Different kinds of errors can appear during the numerical integration of ODE's:

Global truncation error.

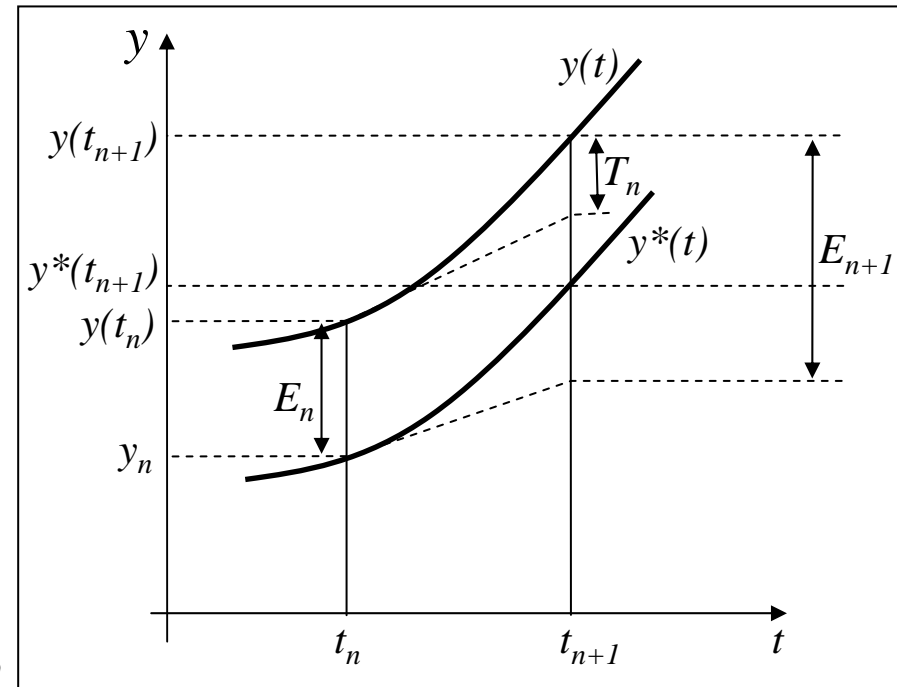
$$E_n = y(t_n) - y_n$$

difference between exact solution at t_n based on original initial conditions and current numerical approximation

Local truncation error.

$$T_{n+1} = y(t_{n+1}) - y(t_n) - h_n F = E_{n+1} - E_n$$

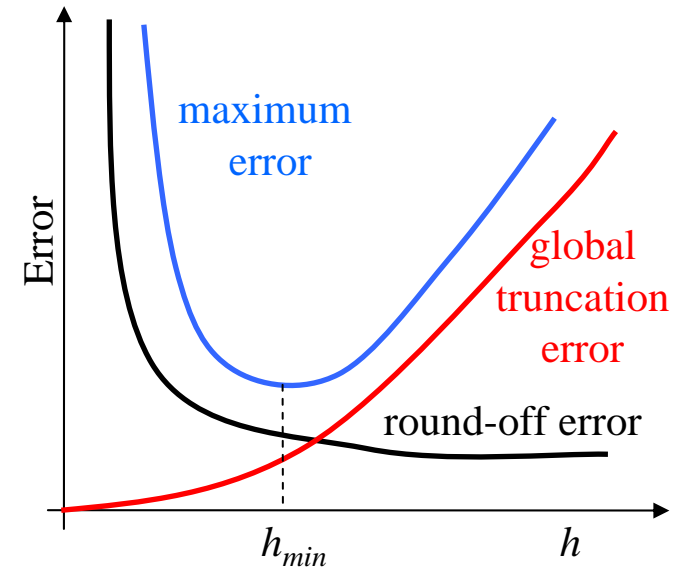
estimate of amount by which the finite difference approximation deviates from the analytic solution $y(t)$



$y(t)$: exact solution using original i.c.
 $y^*(t)$: exact solution using y_n as i.c.

Accuracy: round-off errors

- Errors are also introduced by machine-dependent numerical round-off.
- The global truncation error decreases as h decreases, but the maximum round-off error increases
- Round-off can contribute significantly to the total error when very small timesteps are used
- There is an optimal step size h_{min} where truncation and round-off errors are comparable



Order of accuracy

- The order of accuracy of a method is defined by expressing the measure of error in powers of the stepsize h
- If the global truncation error behaves as $O(h^p)$ for $p > 0$, the method is said to be order p^{th}
- In general a p^{th} order method has a local truncation error which scales as $O(h^{p+1})$
- We generally use high order methods because they converge faster and are more accurate

Convergence and Stability

- Stability and convergence are both concerned with the behavior of the computed solution in the limit as the stepsize goes to zero
- A method *converges* if
$$y_n \rightarrow y(t_n) \quad t \in [t_0, b]$$
as $h \rightarrow 0$ and $y(t_0) \rightarrow y_0$
- A method is said to be *unstable* if an error introduced at some stage in the calculation becomes unbounded, i.e. the overall global error continually increases instead of decreases
- A convergent method is stable. Stability does not require convergence, but convergence requires stability

Explicit and implicit methods

- Explicit methods: the value of y_{n+1} depends only on values at previous steps ($n, n-1, \dots$)
- Implicit schemes: the value of y_{n+1} depends also on values at step $n+1$

Explicit vs implicit schemes

Explicit schemes

- Easy to code
- Easy to apply boundary conditions
- Easy to vectorize / parallelize
- Easy to maintain / upgrade
- No limitation on the order of accuracy
- Step size limited by stability constraint

Implicit schemes

- Can use arbitrary step size
- Order of accuracy for stable schemes limited to 2
- Large overhead in solving large systems of equations

Situations where implicit schemes pay off

- Stiff system of equations / physical stiffness:

relevant or physical step size \gg step size required for stability

Caution

- If step size too large and non-linear system:

⇒ Chaotic solutions possible

⇒ Need to conduct convergence study

- Possible solution: step size adaptivity / sensing in time

Classical methods of solution

- One-step methods
- Linear multi-step methods
- Extrapolation methods
- Leap-frog integration methods
- Implicit methods for stiff ODE's

One-step methods

- Any one-step method can be written as

$$y_{n+1} = y_n + h F(t_n, y_n, h)$$

- The derivative approximation F can be evaluated either from a Taylor series expansion about t_n

$$y_{n+1} = y_n + h \frac{dy_n}{dt} + \frac{h^2}{2!} \frac{d^2 y_n}{dt^2} + \dots$$

or from approximations of the function F in the integral form

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} F(t, y(t)) dt$$

- In the latter case, approximations to F typically take the form of polynomials or exponentials

Euler method

- The simplest one-step method is the explicit, first order Taylor algorithm or Euler method:

$$y_{n+1} = y_n + h f(t_n, y_n)$$

- In this method, we truncate the Taylor series after the first term $\sim O(h)$
- Taylor series algorithms for higher orders can be written analogously from the Taylor series expansion. Typically the higher derivatives are found from analytical differentiation of the original ODE's and substituting into the Taylor series

Runge-Kutta methods

- RK methods are efficient, easily programmed, one-step algorithms that generally give higher order accuracy than Taylor series methods
- Their gains come from evaluating the function $f(t,y)$ at more than one point in the neighborhood of (t_n, y_n) instead of evaluating higher derivatives
- The function F is expressed as a weighted average of first derivatives obtained numerically at points in the region
 $[t_n, t_{n+1}]$

Runge-Kutta schemes

- The generic form for an N-stage Runge-Kutta scheme is

$$k_i = f \left(t_n + h a_i, y_n + h \sum_{j=1}^N b_{ij} k_j \right)$$

where $i=1,2,\dots, N$ labels the stage, and

$$y_{n+1} = y_n + h \sum_{i=1}^N c_i k_i$$

- Explicit schemes are obtained when all k values used are calculated at an earlier step, i.e. when $b_{ij}=0$ for all $j \geq i$

Modified Euler method

- Explicit
- 2nd order accurate
- Two-stage method with $a_2=1/2$, $b_{21}=1/2$, $c_2=1$ and all other constants zero:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + h/2, y_n + hk_1/2)$$

$$y_{n+1} = y_n + hk_2$$

- The Euler method is used to estimate the value of y at the half step $[t_n, t_{n+1}]$. Then the average estimate of the derivative is a step-centered quantity. No extra storage is required.

Improved Euler method

- Explicit
- 2nd order accurate
- Two-stage method with $a_2=1$, $b_{21}=1$, $c_2=1/2$ and all other constants zero:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + h, y_n + h k_1)$$

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

- This method takes the average of the old derivative and the first-order estimate of the new derivative at the end of the step
- This method requires extra storage for k_2

Classical Runge-Kutta scheme

- Explicit
- 4th order accurate
- Four-stage method:

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + h/2, y_n + hk_1/2)$$

$$k_3 = f(t_n + h/2, y_n + hk_2/2)$$

$$k_4 = f(t_n + h, y_n + hk_3)$$

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

- This is the most commonly used one-step method
- Requires extra storage for k_i

Low storage Runge-Kutta scheme

- Explicit
- N-stage scheme:

$$y_{n+i} = y_n + \alpha_i h f(y_{n+i-1})$$

$$\alpha_i = \frac{1}{N + 1 - i}$$

where $i=1, 2, \dots, N$

- N^{th} order accurate for linear ODE's
- Low storage (only need to store previous stage)

Linear multi-step methods

- A N-step multi-step method can be written in general form:

$$y_{n+i} = h \beta_i f_{n+i} + \sum_{j=0}^{i-1} (h \beta_j - \alpha_j y_{n+j})$$

for $i=1, \dots, N$

- We typically need to know the set of past quantities

$$(t_n, y_n), (t_{n-1}, y_{n-1}), \dots$$

at equally spaced intervals in h to compute y_{n+1} . Thus, results must be stored for several steps back.

Adams-Bashford method

- 4th order accurate
- Explicit scheme:

$$y_{n+1} = y_n + h f_n$$

$$y_{n+2} = y_{n+1} + h/2(3f_{n+1} - f_n)$$

$$y_{n+3} = y_{n+2} + h/12(23f_{n+2} - 16f_{n+1} + 5f_n)$$

$$y_{n+4} = y_{n+3} + h/24(55f_{n+3} - 59f_{n+2} + 37f_{n+1} - 9f_n)$$

- The N=1 method is the same as the first-order Euler method

Adams-Moulton method

- 4th order accurate
- Implicit scheme:

$$y_{n+1} = y_n + h/2(f_{n+1} + f_n)$$

$$y_{n+2} = y_{n+1} + h/12(f_{n+2} + 8f_{n+1} - f_n)$$

$$y_{n+3} = y_{n+2} + h/24(9f_{n+3} + 19f_{n+2} - 5f_{n+1} + f_n)$$

$$y_{n+4} = y_{n+3} + h/720(251f_{n+4} + 646f_{n+3} - 264f_{n+2} - 106f_{n+1} - 19f_n)$$

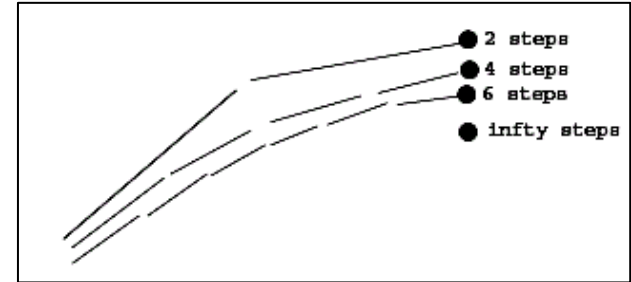
- The N=1 method is called the *trapezoidal method* because it is a trapezoidal quadrature formula

Predictor-corrector methods

- The Adams-Bashford and Adams-Moulton methods are *predictor-corrector* methods because they use a lower order method to predict the answers until enough timesteps have accumulated to carry out the full timestep procedure
- In practice, predictor-corrector methods compare favorable to Runge-Kutta methods, they can be made more accurate with equal computational effort
- Runge-Kutta methods are *self-starting* because they only require data at one time level to begin the integration. Predictor-corrector methods often use a one-step method to accumulate enough values from previous times to proceed
- In Runge-Kutta methods it is easier to vary the timestep (adaptive steps)

Extrapolation methods

- The idea of the Euler-Romberg method is to integrate the equation over the interval h many times using Euler's method with $h, h/2, h/4, \dots$



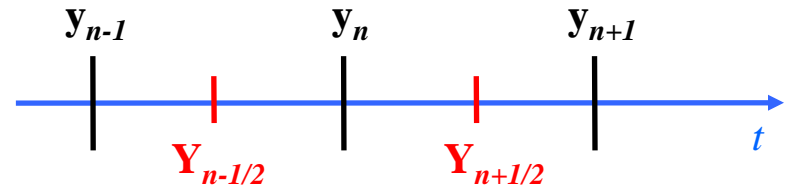
- Then an interpolation is made to $h=0$ with the increasingly accurate integrations with smaller and smaller h values
- This algorithm is self-starting and rivals the best predictor-corrector schemes for efficiency and accuracy
- The choice of step size is fairly arbitrary because the method successively halves the step size until the required accuracy is achieved at each step

Leap-frog integration methods

- In this method, two sets of staggered variables are used:

\mathbf{y}_n at $t_{n-1}, t_n, t_{n+1}, \dots$

\mathbf{Y}_n at $t_{n-1/2}, t_{n+1/2}, \dots$



- The system of equations is:

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{Y}, t)$$

$$\frac{d\mathbf{Y}}{dt} = f(\mathbf{y}, t)$$

- Because the derivative of \mathbf{y}_n depends on the dependent variables $\mathbf{Y}_{n+1/2}$ and vice versa, the centered derivative for each of the variables can always be evaluated explicitly using the most recently updated values for the other set of variables

Leap-frog scheme

- The second order explicit leap-frog algorithm is:

$$\mathbf{Y}_{n+1/2} = \mathbf{Y}_{n-1/2} + h F(\mathbf{y}_n, t_n)$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h f(\mathbf{Y}_{n+1/2}, t_{n+1/2})$$

$$t_{n+1/2} = t_n + h/2$$

- This algorithm is reversible, thus long integrations can be traced back to their initial conditions. This property is particularly useful for systems of ODE's that exhibit time reversibility
- One of the major uses of this algorithm is in particle dynamics where the positions $\mathbf{x} \equiv \mathbf{y}$ depend on the velocities $\mathbf{v} \equiv \mathbf{Y}$ and the velocities depend on a force that is only a function of the instantaneous positions

Implicit leap-frog algorithms

- The leap-frog algorithm can be generalized to include implicit dependence of the variables (more stable algorithms)
- Algorithm based on modified Euler method:

$$\mathbf{Y}_{n+1/2} = \mathbf{Y}_{n-1/2} + h F[\mathbf{y}_n, (\mathbf{Y}_{n-1/2} + \mathbf{Y}_{n+1/2})/2, t_n]$$
$$\mathbf{y}_{n+1} = \mathbf{y}_n + h f[\mathbf{Y}_{n+1/2}, (\mathbf{y}_n + \mathbf{y}_{n+1})/2, t_{n+1/2}]$$

- Algorithm based on improved Euler method:

$$\mathbf{Y}_{n+1/2} - h/2 F(\mathbf{y}_n, \mathbf{Y}_{n+1/2}, t_n) = \mathbf{Y}_{n-1/2} + h/2 F(\mathbf{y}_n, \mathbf{Y}_{n-1/2}, t_n)$$
$$\mathbf{y}_{n+1} - h/2 f(\mathbf{Y}_{n+1/2}, \mathbf{y}_{n+1}, t_{n+1/2}) = \mathbf{y}_n + h/2 f(\mathbf{Y}_{n+1/2}, \mathbf{y}_n, t_{n+1/2})$$

- As long as the implicit dependence on $\mathbf{Y}_{n+1/2}$ and $\mathbf{y}_{n+1/2}$ can be inverted analytically, the algorithms are reversible. If the implicit equations must be iterated, reversibility is lost

Schemes for stiff ODE's

- Stiffness is related to the presence of a wide range of time scales affecting the system dynamics
- Mathematically, a system is stiff when its Jacobian matrix has eigenvalues whose magnitudes differ by a large ratio
- This means that at least two independent homogeneous solutions vary with time at greatly disparate rates
- Stiff problems make us seek methods that do not restrict the step size for stability reasons, and which can treat widely disparate scales in some reasonable manner

Stiff ODE's

- When there are two or more very different “time-scales” in the problem:
 - You can't integrate with the big time steps, because you will miss the rapid changes
 - You can't integrate with the small time steps, because you may miss the slow changes
- Example: $y' = -1000y + 99e^{-x}$

solution: $y = e^{-x} - e^{-1000x}$

the second term dominates very rapidly when $x < 0$
However, for $x > 0$, the first term dominates

Backward differentiation formulas

- These are linear multi-step methods:

$$y_{n+i} = h \beta_i f_{n+i} - \sum_{j=0}^{N-1} \alpha_j y_{n+j}$$

with $\alpha_0 \neq 0$ and $\beta_i \neq 0$

- Coefficients for backward differentiation methods:

i	β_i	α_0	α_1	α_2	α_3
1	1	-1	1		
2	2/3	1/3	-4/3	1	
3	6/11	-2/11	9/11	-	1

18/11

Backward differentiation methods

- These are the most common methods for solving stiff ODE's
- The order of accuracy of these methods is equal to the number of steps N
- The first order method is the backward Euler method

Exponential methods

- Implicit methods for stiff ODE's are derived by *curve-fitting*: an interpolating function is adopted with free parameters determined by requiring the interpolant to satisfy certain conditions on the approximate solution and its derivatives

- Example: choose the two parameter polynomial function

$$I(t) = A + B t$$

- as the interpolant and require that $I(t)$ satisfy

$$I(0) = y_n, \quad I'(0) = f_n, \quad I(h) = y_{n+1}$$

- on the interval $[0, h] = [t_n, t_{n+1}]$. This results in Euler's method:

$$y_{n+1} = y_n + h f_n$$

- Other constraints and interpolants result in other previously described explicit and implicit schemes

Exponential methods

- The basis of this approach is that exact solutions of stiff ODE's behave like decaying exponential functions
- Exponentials are poorly described by polynomials when the step size is larger than the characteristic decay rate
- Thus, using exponential approximations should allow considerably longer timesteps

Exponential methods

- Consider the three-parameter exponential interpolant:

$$I(t) = A + B e^{Zt}$$

for which A , B and Z must be determined.

- The constraints (same as previous example) determine A and B in terms of Z :

$$A = y_n - \frac{f_n}{Z}, \quad B = \frac{f_n}{Z}$$

- Thus, the scheme is:

$$y_{n+1} = y_n + h f_n \left[\frac{e^{Zh} - 1}{Zh} \right]$$

Exponential methods

- There are a number of ways to choose the parameter Z :

explicit : $Z = f'_n / f_n$

explicit : $Z = \frac{1}{h_{n-1}} \ln \left(\frac{f_n}{f_n - 1} \right)$

implicit : $Z = \frac{1}{h} \ln \left(\frac{f_{n+1}}{f_n} \right)$

$$f' = \frac{d^2 y}{dt^2}$$

implicit : $Z = \frac{1}{2} \left[\frac{f'_n}{f_n} + \frac{f'_{n+1}}{f_{n+1}} \right]$

- It has been shown that exponential methods can be at least comparable in speed and accuracy to the backward differentiation methods for stiff ODE's

Variable step sizes

- One of the “big ideas” in numerical ODE solvers (and in computational sciences as a whole) is variable step sizes
- Not ALL steps need to be the same size. The step size taken should depend on the local behavior of the function

- Taking a previous example: $y = e^{-x} - e^{-1000x}$

At $x \gg 1$ the function changes very slowly \Rightarrow large steps

For $0 < x < 1$, it changes very rapidly \Rightarrow smaller steps

- The trick is understanding when to change the step size

Adaptive step sizes

- The fundamental ingredient for an adaptive step size method is a way to estimate the error of the numerical approximation
- There are two basic ways of checking the local error:
 - Use two methods with different orders over the same interval
 - Use two step sizes over the same interval
- The first method allows you to use two integrators with different order on the same step, and you estimate the error by

$$E_{local} \approx \left| y_n (high) - y_n (low) \right|$$

- Usually, the low order method will have less accuracy than the high order method

High / low order step control

- The basic idea is to solve the same problem using a high order and a low order method to estimate the error
- The most common example combines a 4th and a 5th order Runge-Kutta routine (known as Runge-Kutta Fehlberg):
 - Make a trial calculation using the 4th order method
 - Make a trial calculation using the 5th order method
 - If the difference is small, take a step using the 5th order trial result
 - Based on the error between the two methods and a prescribed tolerance, set the step-size for the next step:

$$h_{new} = h_{old} \left| \frac{\delta_{target}}{\delta_{measured}} \right|^{1/5}$$

- If the difference between the two methods is large, repeat the steps with a smaller step-size

Error estimation

- In some cases, you can't use a high order method for a step
- But you can always take two sets of steps over the same interval. The first step can use a step-size h , and the second a step-size $h/2$
- Again, we form the approximate error by considering the difference of the two methods

$$E_{local} \approx \left| y_n(h) - y_n(h/2) \right|$$

- The assumption we are making is that the small step size is a better estimate than the big step size. If the method converges, this is very likely

Using error estimates

- After you have calculated the local error, you then use it to make decisions about the local step. You need to set an error tolerance for the integrators
- The algorithm works something like this:
 - Calculate the local error
 - If the local error is larger than the error tolerance, decrease the step size and repeat the step
 - If the local error is MUCH smaller than the error tolerance, accept the step, but increase the step size for the next iteration
 - If the local error is acceptable, then accept the step and continue

Using error estimates

- You always use the highest order method for the actual step since it is more accurate than the lower order method
- Variable step sizes will not work well with all methods
- In predictor-corrector methods, for example, it is difficult to change the step size since they rely on the information from the previous steps

Global conservation measures

- The idea is to find global quantities to monitor during the integration which should be conserved. In some simulations, total energy, total angular momentum, total mass, etc. can be tracked and checked for conservation. If these quantities vary much from their initial values, something went wrong
- In reality, global conservation measures really aren't the best way to control the error in ODE integration. However, they provide an independent check of the accuracy of the results
- We could look at (say) the energy change at each time-step and see if it goes beyond a preset limit, and adjust the step size accordingly. But, this may not be practical ...

Local step size criteria

- One of the most common ways to set the step size is to use local step size criteria. Basically, we find some local stability criteria in the equations which should not be violated, and then set the step-size to be based on this value
- The best examples of this are in fluid mechanics, where the step size is driven by the Courant-Fredrerichs-Levy (CFL) condition. The assumption is that information should propagate between cells at a rate which does not exceed the time the fastest fluid element can cross the smallest grid cell
- Similar criteria are used in other fields

ODE solvers

Design criteria:

- Consistency
 - Approximation \rightarrow ODE for $h \rightarrow 0$
- Accuracy
 - Magnitude of local errors
- Stability
 - Long-term effect of local and round-off errors
- Efficiency
 - CPU and storage vs accuracy

ODE solvers: consistency

- Approximation \rightarrow ODE for $h \rightarrow 0$
- Example: forward Euler

$$y_{n+1} = y_n + h f_n$$

Taylor series

$$y_{n+1} = y_n + h y'_n + \frac{h^2}{2} y''_n + \frac{h^3}{6} y'''_n + \dots$$

So,

$$\lim_{h \rightarrow 0} \frac{y_{n+1} - y_n}{h} = y'_n = f$$

- In general consistency is not difficult to prove

ODE solvers: accuracy

- Local error of approximate vs exact solution
- Proven via Taylor-series expansion
- Example: forward Euler

$$y_{n+1} = y_n + h f_n$$

but

$$y_{n+1} = y_n + h y'_n + \frac{h^2}{2} y''_n + \frac{h^3}{6} y'''_n + \dots$$

insert original ODE for y'

$$y_{n+1} = y_n + h f_n + \frac{h^2}{2} f'_n + \frac{h^3}{6} f''_n + \dots$$

leading term of error $\sim O(h^2) \Rightarrow$ first-order accurate

- Taylor series expansion assumes smooth y , which is not always the case !

ODE solvers: stability

Long term effects of errors:

- Approximation errors
- Round-off errors

Proven via eigenvalue analysis

Example: $y' = -\alpha y$

Exact solution: $y = y_0 e^{-\alpha t}$

Suppose: $y_n = y_0 g^n$

Stable iff: $|g| < 1$

ODE solvers: stability

Examples:

- Forward Euler: $y_{n+1} = y_n - \alpha h y_n$

$$g = 1 - h\alpha$$

$|g| < 1 \Rightarrow$ stable for $h < 2/\alpha$

- Backward Euler: $y_{n+1} = y_n - \alpha h y_{n+1}$

$$g = \frac{1}{1 + h\alpha} < 1$$

$|g| < 1 \Rightarrow$ stable for any h (unconditionally stable)

ODE solvers: stability

- Stability analysis is equivalent for other ODE's
- Sometimes it is difficult to prove stability for non-linear systems of equations

ODE solvers: efficiency of high-order schemes

- Assume cumulative errors (worst case scenario)
⇒ local error:

$$err \leq \frac{1}{p!} h^{p+1} \max \left| \frac{d^{p+1} y}{dt^{p+1}} \right|_{t_0}^{t_1} = c_1 h^{p+1}$$

accumulation error:

$$E \leq \frac{(t_1 - t_0)}{h} c_1 h^{p+1} = c_2 h^p$$

total work:

$$W = c_3 \frac{(t_1 - t_0)}{h}$$

work x error:

$$W \times E = c_4 (t_1 - t_0)^2 h^{p-1}$$

- Implies:
 - Need at least $p=1$ (second order) to converge work-wise
 - Should use $p>1$ (third order or higher)

Remarks

- ODE solvers are an essential ingredient for solving time-dependent PDEs which appear very often in computational sciences

Study Questions

1. What is the difference between explicit and implicit methods ? when would you use each method ?
2. Show that the explicit forward Euler method is first order accurate
3. What is a self starting scheme ? are the Runge-Kutta methods self starting ? and the predictor-corrector schemes ?
4. What is a stiff ODE ? what is the main difficulty for solving stiff ODE's ?
5. Describe the concept of adaptive step sizes. What is the fundamental ingredient for adaptive step size schemes ?