Unit V

Differential Equations

Differential equations

- explicit one-step methods
  - Euler method
  - improved Euler method
  - analysis of explicit methods
- Runge-Kutta methods
- adaptivity and stiffness
- multi-step methods

Integration of ODEs

- ODE = *ordinary differential equation*
  - involves functions of a single variable and the derivatives
- *order* = the highest derivative
- we concentrate on solving
  - first order ODEs
  - systems of first order ODEs

  because...
  - a higher order ODE can be reduced to an equivalent system of first order ODEs

Integration of ODEs

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

- use the intermediate variable z(x) as shown
- first we examine methods for numerical solution of a single first order equation ....

First order ODE

- the general first order ODE is

$$\frac{dy}{dt} = f(t, y) \quad \text{OR} \quad \frac{dy}{dx} = f(x, y)$$

- f is an arbitrary function
- y = y(t) or y = y(x) according to context
  - independent variable often is t (time) to reflect ODE use in a dynamic problem

System of first order ODEs

- most general is a coupled set of N ODEs

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

- given functions f_i that define the ODE
- find function(s) $y_i(x)$ called *solutions* of the ODE
- this process is called *integrating* the ODE
- we could ask first about uniqueness ....

Uniqueness of solutions

- to get a unique solution you need some constraints on y_i
- these are called *boundary conditions*
  - algebraic relationships between the solutions
  - defined at specified discrete points
  - not valid elsewhere
  - simplest is to specify y_i values at given point(s)
  - most complex is a system of nonlinear equations relating the solution functions (*differential algebraic equations* = *DAEs*)

Boundary conditions

- the type of boundary conditions determines the numerical tactics which work best
- *initial value problem (IVP)*
  - given values $y_i(x_s)$ *for all* the y_i at the same *initial point* x_s
  - to find values of $y_i(x_f)$ at some *final point* x_f
  - values $y_{ij} = y_i(x_j)$ at discrete intermediate points x_j, $x_s < x_j < x_f$ may also be required
- *2-point boundary value problem*
  - some values given as $y_i(x_s)$
  - other values given at $y_i(x_f)$
  - a more difficult problem
- the general first order IVP is **$y' = f(x,y)$, $y(x_0)=y_0$**

Connection to definite integral

- if y is a function only of x a first order IVP is just a definite integral ....
- $y' = f(x)$, $y(x_0)=y_0$ is the same as

$$y(x) = y_0 + \int_{x_0}^{x} f(z)dz$$

- so we could apply numerical quadrature techniques to solve it
- we know that many integrals have no unique closed form analytical solution
- so we can expect *existence* of solutions to the general IVP will be more problematic
- we tackle the problem numerically ....

Use iterations to solve an IVP

- we want the solution y(x)
  - or specifically $y(x_f)$ for some value $x = x_f$ of interest
- approximate y(x) by a sequence of discrete values
  $y(x_0)$, $y(x_1)$, $y(x_2)$, ..., $y(x_f)$
  - start at the initial x-value x_0
  - increment by step h
  - arrive at the final x-value x_f
- example: solve $y' = \sin(x^2)$, $y(0)=0$
  - for a simple case like this [i.e. y=f(x) only] something like trapezoid integration works
  - the Matlab functions *trapz* and *cumtrapz* are useful to illustrate
- numerical methods for IVPs follow this kind of step-by-step method to walk to the solution

Trapezoidal scheme

- in the simple case .... we can get the next y_i value in one step

$$y_1 = y_0 + \frac{h}{2}(f(x_0) + f(x_1))$$

- but this doesn't work for the general case

$$y_1 = y_0 + \frac{h}{2}(f(x_0, y_0) + f(x_1, y_1))$$

- the unknown y_1 appears on both sides of the equation
  - we can't evaluate $f(x_1,y_1)$ because we don't know y_1 yet

Trapezoidal scheme

- write the equation as implicitly defining y_1 and solve

$$y_1 - y_0 - \frac{h}{2}(f(x_0, y_0) + f(x_1, y_1)) = 0$$

  using something like Newton's method
- repeat the process at each step to get y_2, y_3, ... values
- this is called the *trapezoidal scheme*
  - an implicit method
  - useful for some special cases (e.g. stiff equations)
  - more efficient explicit methods are available

Explicit approach

- Euler tactics:
  - approximate dy/dx by a forward difference $\Delta^1 f = \Delta y/\Delta x$
  - multiply the equation by Δx
  - you get algebraic formulas for the change y_i as x is stepped one step-size $h = \Delta x$
  - if h is 'small enough' you MAY get a good approximation to the solution
- in practice ... Euler au nature is
  - not very accurate
  - not very stable
- in theory... it is the fundamental conceptual basis for most ODE solution methods:
- **add small increments of derivatives (right hand side functions) times step-sizes to your functions**

Euler tactics by first difference

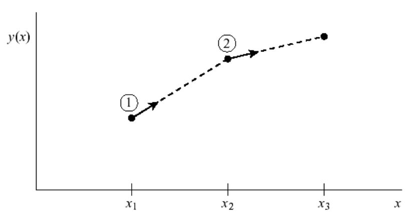
$$\begin{aligned} y' &= f(x,y) \\ \frac{y(x+h)-y(x)}{h} &\approx f(x,y) \\ y(x+h) &\approx y(x) + hf(x,y) \end{aligned}$$

- so we define the Euler step by

$$y_{i+1} = y_i + hf(x_i, y_i)$$

- we neglect the truncation errors introduced by using first differences

Euler's method



- the derivative at the starting point of each interval is extrapolated to find the next function value
- **$y_{i+1} = y_i + h\, f(x_i,y_i)$**

Euler tactics by Taylor series

- the Euler step can also be derived from a truncated Taylor series

$$\begin{aligned} y(x_0+h) &= y(x_0) + hy'(x_0) + \frac{h^2}{2}y''(\gamma) \\ &\approx y(x_0) + hy'(x_0) \\ &= y_0 + hf(x_0, y(x_0)) \\ y_1 &= y_0 + hf(x_0, y_0) \end{aligned}$$

- y'' must exist for this to work
- truncation error is $O(h^2)$ for one step so ...
- the method has $O(h)$ accuracy

Euler is unstable

- the Euler step relies on first differences so ...
- we cannot really e x p e c t stability
- truncation error
  - accuracy is $O(h)$ at a fixed point x
  - so you can improve accuracy by reducing stepsize but ...
  - ... only if you stay at the same x value
  - if x moves the solution walks away across the one-parameter set of solution curves
  - error grows with increasing x and decreases with decreasing h
- what about roundoff error?
  - also an issue
  - must be wary of making h too small

Example: Euler steps

Apply Euler steps to the ODE IVP $y' = 1+y^2$, $y(0)=1$. True solution is $y(x)=\tan(x+\pi/4)$. Use h=0.5 and solve over [0,2].

Example: Euler steps are O(h)

Apply Euler steps to the ODE IVP $y' = y$, $y(0)=1$. True solution is $y(x)=e^x$. Start with h=0.5, continuously halve the step size, and check the error at each stage.

Improving the Euler step

- Euler is O(h) because ....
- it uses $f(x_i,y_i)$ to extrapolate across the interval $[x_i,x_{i+1}]$ to get a value for $y(x_{i+1})$
  - in reality the slope is changing across that interval
  - ... and possibly rapidly
- the trapezoidal scheme gets $O(h^2)$ because ....
- it uses values at BOTH endpoints to estimate the slope but
  - it's an implicit method, so less useful in practice
- viewed in terms of quadrature
  - Euler uses the left endpoint scheme
  - trapezoidal uses the trapezoid method
- a *midpoint scheme* is a good alternative ....

Midpoint schemes

- we use the midpoint quadrature method to get the midpoint scheme for ODEs:

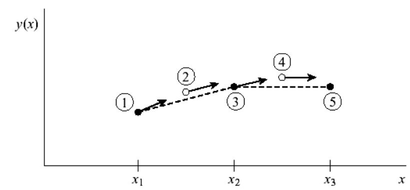
$$\begin{aligned} y_{i+1} &= f(x_i, y_i) \\ &= y_i + hf(x_{i+1/2}, y_{i+1/2}) \end{aligned}$$

  - here we have

$$\begin{aligned} x_{i+1/2} &= x_i + h/2 \\ y_{i+1/2} &\approx y(x_{i+1/2}) \end{aligned}$$

  - but we don't have an estimate for $y(x_{i+1/2})$ of course
- so how to get an explicit method out of this?

Improving the Euler step: geometric



- using f(x,y) at the left starting point (1) to ...
- find the midpoint (2)
- then use f(x,y) at the midpoint (2) to update (1) and ...
- find a better path to the end point (3) where $x_2 = x_1+h$

Improving the Euler step: algebraic

- we estimate the y-value at the midpoint by

$$y_{i+h/2} = y_i + \frac{h}{2} f(x_i, y_i)$$

  - both x and y values at the midpoint are used to update the solution
  - the slope for the real step across the interval is estimated using an Euler step and ...
  - adjusted to improve the overall performance
  - symmetry cancels first order error terms, so the method is $O(h^2)$

Improved Euler methods

- the *explicit midpoint method* is one way to improve Euler

$$y_{j+1} = y_j + hf(x_j + \frac{h}{2}, y_j + \frac{h}{2} f(x_j, y_j))$$

  - the (unknown) y value at the midpoint is estimated using an Euler step

- the *explicit trapezoid method* is another alternative

$$y_{j+1} = y_j + h\left(\frac{f(x_j, y_j) + f(x_j + h, y_j + hf(x_j, y_j))}{2}\right)$$

  - the (unknown) y value at the right endpoint is estimated using an Euler step

Example: Improved Euler method

Solve $y' = x + y^2$, $y(0) = 1$ using the explicit midpoint method with step size $h = 0.1$.

Example: Improved Euler method

Solve $y' = x - y$, $y(1) = 1$ using the explicit midpoint scheme with $h = 0.1$ [the exact solution is $y = x - 1 + e^{1-x}$]

Predictor-corrector shemes

- a *predictor-corrector* method turns an implicit scheme into an explicit one
  - the updated y-value is written as

$$y_{i+1} = f(y_{i+1}, p_1, \cdots, p_n)$$

  - $p_1, ..., p_n$ are parameters, e.g. x_i, y_j etc.
  - the explicit scheme (the *predictor*) is used to get the RHS above
  - the implicit scheme (the *corrector*) is used to improve the solution
- repeated prediction-correction steps can be used

Three ways to reduce error

1. use a higher-order method
   - more difficult to program
   - require additional differentiability conditions
2. use a smaller step size h
   - takes longer to converge
   - can lead to propagation of roundoff error
   - converges to the exact solution of the ODE in the absence of roundoff error
3. use repeated corrections
   - takes longer to calculate
   - converges to an exact solution of a discrete approximation to the ODE

- **there is no *RIGHT* way .... hence the variety of available methods**

Step sizes

- second order methods use
  - two function evaluations to take ...
  - a step of size h
- first order methods can use
  - two function evaluations to take ...
  - 2 steps of size h/2
- for small h
  - the $O(h^2)$ error of the second-order method is almost certainly smaller than ....
  - the $O(h/2)$ error of the first-order method and half-size step
- choose the step size as large as is consistent with desired accuracy

Heun's method

- *Heun's method* uses the predictor

$$\begin{aligned} k_1 &= hf(x_j, y_j) \\ y_{j+1} &= y_j + k_2 \end{aligned}$$

  with the corrector

$$k_2 = \frac{h}{4}[f(x_j, y_j) + 3f(x_j + 2h/3, y_j + 2k_1/3)]$$

- this is a weighted average of
  - 1/4 times the slope at the left endpoint and ...
  - 3/4 times the slope 2/3 of the way along the interval
- the method is $O(h^2)$

The theta method

- the *theta method* uses a weighting

$$y_{j+1} = y_j + h(\theta f(x_j, y_j) + (1-\theta) f(x_{j+1}, y_{j+1}))$$

  - Euler is θ = 1
  - trapezoidal scheme is θ = 1/2
  - *backward* (or *implicit*) *Euler* is θ = 0

$$y_{j+1} = y_j + hf(x_{j+1}, y_{j+1})$$

- all theta methods are *O*(h) except the trapezoidal scheme

Explicit one-step methods

- an *explicit one-step* method is of the form

$$y_{j+1} = y_j + h\Phi_f(x_j, y_j, h)$$

  - the *increment function* Φ_f depends on f and its derivatives
  - explicit because everything is known here and...
  - one-step because only one step is used to get the next y-value
- an *implicit one-step* method involves unknown quantities on the RHS of the step equation
  - backwards Euler $y_{j+1} = y(x_j, x_{j+1}, y_j, y_{j+1})$
  - all theta methods (except for Euler itself)
- an *explicit multi(k)-step* method uses a similar formula with k previous y values to get the next one

Two ways to assess error

- the *local truncation error* of an explicit one-step method at a point x_i is ...
  - $LTE_{i+1} = y_{i+1} - [y(x_i) + h\Phi(x_i, y(x_i), h)]$
  - the difference between $y(x_{i+1})$ and the value we would have got for y_{i+1} if we had used the exact value $y_i = y(x_i)$ for the step
- the *global error* of an explicit one-step method is ...
  - $GE_{i+1} = y(x_{i+1}) - y_{i+1}$
  - the difference between the true value and the computed value

Assessing methods

- GE is the most interesting quantity but it is most easily assessed using the LTE
- an explicit one-step method is ...

  convergent if GE → 0 as h → 0
- example: Euler's method has ...
  - an *O*(h^2) local truncation error and
  - an O(h) global error
  - the latter is what we are interested in to assess the method
  - can also show that Euler is convergent

Taylor methods

- the solution to $y' = f(x,y)$, $y(x_0) = x_0$ is ....
  - automatically differentiable
  - suppose it's also twice differentiable
  - then calculate

$$\begin{aligned} y'' &= \frac{d}{dx} y'(x) \\ &= \frac{d}{dx} f(x, y(x)) \\ &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial x} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} \\ &= \frac{\partial f}{\partial x} \cdot 1 + \frac{\partial f}{\partial y} f(x,y) \\ &= f_x + f_y f(x,y) \end{aligned}$$

Taylor methods

- the Taylor expansion about x_0 is

$$y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{h^2}{2} y''(x_0) + O(h^3)$$

- second order approximation gives

$$\begin{aligned} y(x_0 + h) &\approx y(x_0) + hy'(x_0) + \frac{h^2}{2} y''(x_0) \\ y_1 &= y_0 + hf(x_0, y_0) + \frac{h^2}{2} f'(x_0, y_0) \\ &= y_0 + hf(x_0, y_0) + \frac{h^2}{2} (f_x(x_0, y_0) + f_y(x_0, y_0) f(x_0, y_0)) \end{aligned}$$

- giving the O(h^2) Taylor scheme ...

Second order Taylor method

$$y_{j+1} = y_j + hf(x_j, y_j) + \frac{h^2}{2}\left(f_x(x_j, y_j) + f_y(x_j, y_j) f(x_j, y_j)\right)$$

- if y is differentiable enough we can get kth order Taylor schemes too
  - Euler's method is the first order Taylor scheme (slide 16)
- Taylor methods require symbolic partial derivatives
  - so not often used in practice
  - how can we make them practicable?

Runge-Kutta methods

- to avoid evaluating the derivatives required for the Taylor method ...
  - approximate these by a weighted average using only f(x,y) function evaluations
  - can combine the information from several Euler-type steps taken across the interval
  - each Euler step requires evaluation of f(x,y) only once
- *Runge-Kutta methods* of different orders are possible
  - correspond to the Taylor method being approximated
  - called R-K2, R-K3 etc

Runge-Kutta methods

- an *explicit Runge-Kutta method* of order N looks like this:

$$y_{j+1} = y_j + h(k_1 g_1 + k_2 g_2 + \cdots + k_N g_N)$$

with

$$\begin{aligned}
g_1 &= f(x_j + c_1 h, y_j) \\
g_2 &= f(x_j + c_2 h, y_j + a_{2,1} h g_1) \\
g_3 &= f(x_j + c_3 h, y_j + a_{3,1} h g_1 + a_{3,2} h g_2) \\
&\vdots \\
g_N &= f(x_j + c_N h, y_j + a_{N,1} h g_1 + a_{N,2} h g_2 + \cdots + a_{N,N-1} h g_{N-1})
\end{aligned}$$

Runge-Kutta methods

- c_1, ..., c_N are called the *RK nodes*
  - typically c_1=0 and often c_N=1
  - determines x-locations where the 'trial derivatives' are to be taken
- k_1, ..., k_N are called the *RK weights*
  - gives the linear combination of 'trial derivatives' used to estimate the average slope across the step
- the lower triangular A = ($a_{i,j}$) is called the *RK matrix*
  - determines the values used for 'trial derivatives' to be combined
- the choice of nodes, weights, and the RK matrix defines the *explicit R-K* method (ERK)
  - there are also *implicit R-K* methods for which A is not lower-triangular [less utilitarian and less common than ERK methods]
- the *R-K tableau* k|c|A displays the R-K method simply

Classical R-K2 methods

- $\begin{array}{c|c|cc} 0 & 0 & \\ 1 & \frac{1}{2} & \frac{1}{2} \end{array}$ is the *explicit midpoint method*

$$y_{j+1} = y_j + hf(x_j + \frac{h}{2}, y_j + \frac{h}{2} f(x_j, y_j))$$

- $\begin{array}{c|c|c} \frac{1}{2} & 0 & \\ \frac{1}{2} & 1 & 1 \end{array}$ is the *explicit trapezoid method*

$$y_{j+1} = y_j + h\left(\frac{f(x_j, y_j) + f(x_j + h, y_j + hf(x_j, y_j))}{2}\right)$$

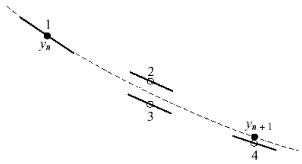
- $\begin{array}{c|c|c} \frac{1}{4} & 0 & \\ \frac{3}{4} & \frac{2}{3} & \frac{2}{3} \end{array}$ is *Heun's method*

Classical R-K4 method

- $\begin{array}{c|c|ccc} \frac{1}{6} & 0 & & & \\ \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & & \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} & \\ \frac{1}{6} & 1 & 0 & 0 & 1 \end{array}$ is the classical R-K4 method

$$\begin{aligned}
y_{j+1} &= y_j + h\left(\frac{1}{6}g_1 + \frac{1}{3}g_2 + \frac{1}{3}g_3 + \frac{1}{6}g_4\right) \\
g_1 &= f(x_j, y_j) \\
g_2 &= f\left(x_j + \frac{1}{2}h, y_j + \frac{1}{2}hg_1\right) \\
g_3 &= f\left(x_j + \frac{1}{2}h, y_j + \frac{1}{2}hg_2\right) \\
g_4 &= f(x_j + h, y_j + hg_3)
\end{aligned}$$

Finding the weights for R-K4



- the slope function f(x,y) is evaluated four times per h step
  - once at the start point (1)
  - twice at trial midpoints (2&3)
  - once at a trial endpoint (4)
- a weighted average of these slopes updates y_j to y_{j+1}

Example: RK-4 method

Use the R-K4 method to solve $y' = x^2 y$, $y(0)=1$ with h=0.05
[the exact solution is $y = \exp(x^3/3)$].

Choosing an R-K method

- R-K4 requires four evaluations of the righthand side function per h step
- superior to R-K2 provided twice as large step size gives better accuracy
  - often true, usually true, but....not always
  - remember high order does not always imply high accuracy
- **the availability of a variety of algorithms is important for solving ODE problems**

R-K methods

- almost always successful, but....
- only moderate accuracy and...
- not the most efficient generally
- R-K methods are the workhorse of ODE solving
- internal consistency can be monitored to keep track of inaccuracy, and step-size adjusted on the fly
  - *adaptive step-size* algorithms
  - local truncation error can be used to estimate the global order
  - step-size adjusted to meet specified tolerance criteria

Setting up numerical routines

- algorithm routine
  - sets up the dependent y_j's at the starting value x
  - calculates new values for the dependent y_j's at x+h
  - provides information required for quality control
- stepper routine
  - calls the algorithm routine
  - decides whether to accept the values, or....
  - reject the h step and call the algorithm with a smaller step-size
  - finds the largest step-size compatible with specified performance
- driver routine
  - starts and stops the integration
  - stores intermediate values
  - acts as user-interface

Matlab implementation

- ode23
  - simultaneous R-K2 and R-K3 methods
- ode45
  - simultaneous R-K4 and R-K5 methods
- these routines use
  - an adaptive step-size and ...
  - monitor the accuracy
- the implementation of the algorithm shares intermediate slope values
  - reduces the number of function evaluations per step

Using Matlab functions

- *[x,y] = ode45(**diffeq,xn,y0**)*
- *[x,y] = ode45(diffeq,[x0 xn],y0)*
- *[x,y] = ode45(diffeq,[x0 xn],y0,options)*
- *[x,y] = ode45(diffeq,[x0 xn],y0,options,arg1,arg2,...)*
- *diffeq* = name of m-file (string) that evaluates f(x,y)
- *[x0 xn]* = vector defining integration interval
  – default x0 = 0, in which case only xn has to be given
- *y0* = initial condition
- *options* = datastructure for adjusting control parameters

Controlling Matlab functions

- default control parameters can be adjusted with the odeset function:
  options = odeset('paramname',paramvalue, ...)
  [x,y] = ode45(diffeq,xn,y0,options)
- the standard 'paramname' list is
  – applicable to all (relevant) Matlab ode solvers
- example
  options = odeset('RelTol',1e-6,'MaxStep',0.2)
- arg1,arg2,... pass-through parameters to adjust diffeq function
  – requires options to be used, but....
  – a null matrix [ ] can be used if none of default options are to be adjusted: [x,y] = ode45(diiffeq, xn, y0,[ ],alpha, beta)

R-K4 for systems of ODEs

- each equation has a set of trial slopes $g_1,...,g_4$, but....
- each slope in general depends on x_j, and ALL the y_j values for each equation
- so all g_1's have to be evaluated before any g_2's, then...
- all g_2's have to be evaluated before any g_3's, then...
- all g_3's have to be evaluated before any k_g's, then...
- the g_4's can be found for each equation, then....
- the y_j values can ALL be incremented to the next step
- **ode45 can solve a system of ODEs too**
  – **the derivative function and initial value need to be column vectors**