CHAPTER 2 Solutions of ODEs

I Introduction

- Any system of ODEs (any order) can be written as a system of first order ODEs

$$\frac{d\underline{Y}}{dx} = f(x, \underline{Y}) \qquad \text{where } f \text{ is a vector}$$

$$\Leftrightarrow \quad \frac{dY_i}{dx} = f_i(x, \underline{Y}) \qquad \text{for } i = 1..N$$

Note: even an eigenvalue problem can be cast into this form by setting $\lambda = Y_j$

$$\frac{dY_j}{dx} = 0$$

- There are typically two kinds of problems
  - + initial value problems (by suitable change of variable, the system can be cast into a form where $Y_i(0) = y_i$ are known $\forall i$)
    - → all boundary conditions are at one point
  - + two point boundary value problems
    - → given an interval $[a,b]$, there is at least one boundary condition at each boundary.

- In the first case, the algorithm must be able to predict qualitatively the behavior of $\underline{Y}$ as $x \to \infty$ and attempt to minimize the error between true & numerical solution

- In the second case the algorithm must be able to find solutions that approximately satisfy both sets of boundary conditions and attempt to minimize the error between true & numerical solution.

II Initial value problems

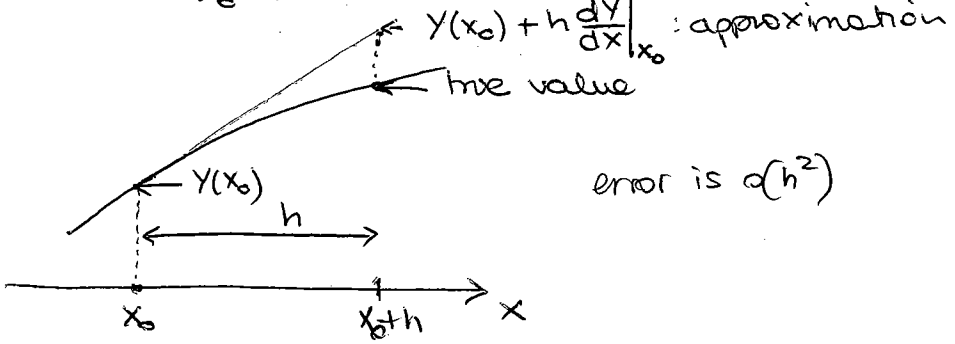
II.1 Introduction

The basis for the numerical solution of all ODEs is the Taylor expansion:

$$Y(x_0+h) = Y(x_0) + h\left.\frac{dY}{dx}\right|_{x_0} + \frac{h^2}{2}\left.\frac{d^2Y}{dx^2}\right|_{x_0} + \dots$$

$$= Y(x_0) + h\left.\frac{dY}{dx}\right|_{x_0} + o(h^2)$$

↳ to advance a solution from a point x_0 to a point x_0+h it suffices to know the derivative at x_0 -



This is the idea behind Euler's explicit algorithm:

Given a mesh x_n

Given a set of ODEs $\frac{dY}{dx} = f(x,Y)$ with IV $Y(x_1) = y_1$

then
$$\begin{cases} Y(x_1) = y_1 \\ \text{do } n = 1, nmax \\ \quad Y(x_{n+1}) = Y(x_n) + (x_{n+1}-x_n) f(x_n, \underline{Y}(x_n)) \\ \text{enddo} \end{cases}$$

Note:
- the increment to $\underline{Y}(x_n)$ depends only on the value of $\underline{Y}$ at x_n
- the algorithm would be exact if $Y(x)$ was linear
- The error commited between the true and approximate solution is, for one step, $o(h^2)$
  ⇒ the error over $\frac{1}{h}$ steps (for a change $O(1)$ in Δx) is $o(h)$.
  ⇒ Globally, Euler's method is 1st order accurate

II. 2 A better algorithm

- Euler's explicit method has poor accuracy properties
- To improve it, we may consider calculating instead

$$\frac{dY}{dx} \cong \frac{Y(x_{n+1}) - Y(x_n)}{(x_{n+1} - x_n)} \simeq \frac{1}{2}\left[f(x_n, Y(x_n)) + f(x_{n+1}, Y(x_{n+1}))\right]$$

↑ approximation of $\frac{dY}{dx}$, actually exact somewhere between x_n and x_{n+1}

↑ average of $f(x, Y)$ over the two endpoints of the iteration.

$$\Rightarrow \quad Y(x_{n+1}) = Y(x_n) + \frac{1}{2}(x_{n+1} - x_n)\left[f(x_n, Y(x_n)) + f(x_{n+1}, Y(x_{n+1}))\right]$$

it can be shown that the local error thus commited is $o(h^3)$ so the accumulation of $\frac{1}{h}$ steps yields a global error $o(h^2)$ → much better.

Problem: this is now an implicit formula for $Y(x_{n+1})$ (appears both on LHS & RHS)

- either f is a linear function, in which case it is actually possible to solve for the linear system

usually requires a matrix inversion

$$Y(x_{n+1}) - \frac{\Delta x}{2} f(x_{n+1}, Y(x_{n+1})) = Y(x_n) + \frac{\Delta x}{2} f(x_n, Y(x_n))$$

↳ Implicit formula.

- or, f is a nonlinear function, in which case this cannot be done

However, we may cheat by evaluating an approximation to $Y(x_{n+1})$ using Euler's explicit method, then refining the approximation using Euler's modified algorithm

Euler's modified Algorithm

- $Y(x_1) = y_1$
- do $n = 1, nmax$

$$k_1 = (x_{n+1} - x_n) f(x_n, Y(x_n))$$

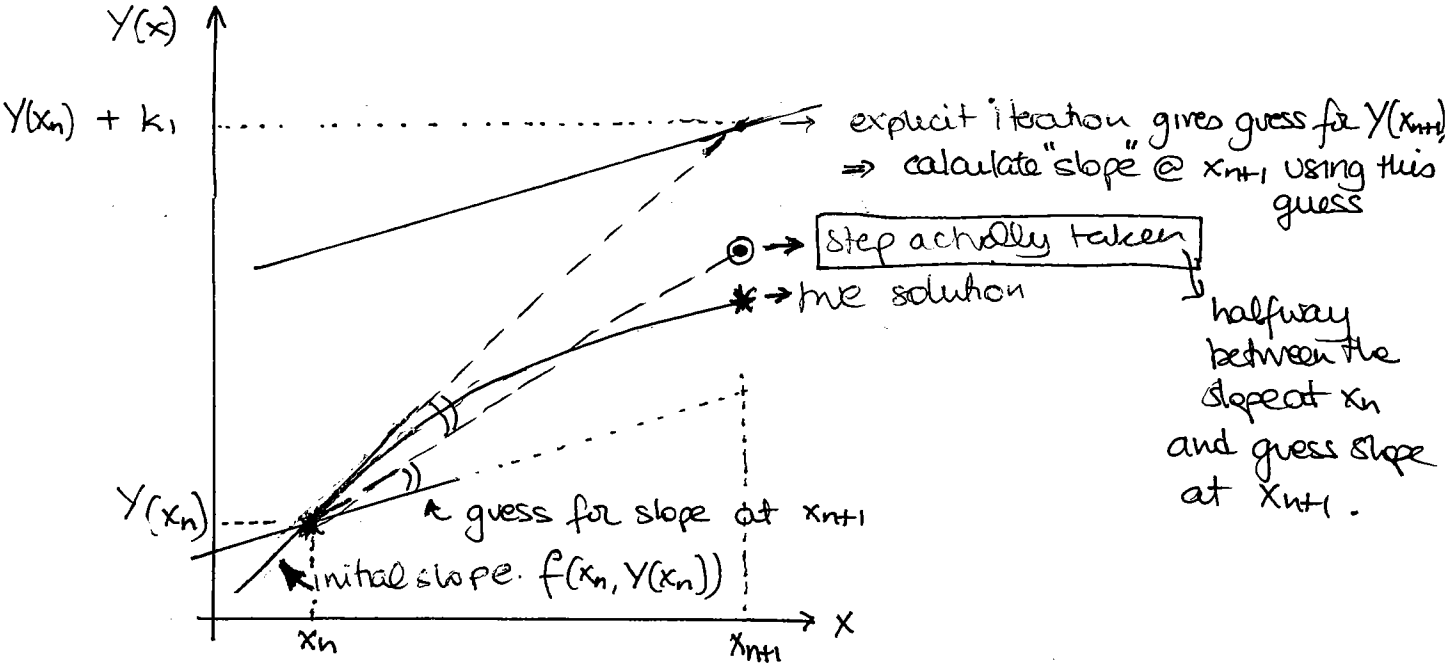
$Y(x_n) + k_1$ approximates $Y(x_{n+1})$

$$k_2 = (x_{n+1} - x_n) f(x_{n+1}, Y(x_n) + k_1)$$

$$Y(x_{n+1}) = Y(x_n) + \frac{1}{2} k_1 + \frac{1}{2} k_2$$

an approximation to $(x_{n+1} - x_n) f(x_{n+1}, Y(x_{n+1}))$

enddo



Note:
- Euler's Modified algorithm is a special case of a 2nd-order Runge Kutta Method.
- Runge Kutta methods are based on the idea that the true derivative can be approximated by a weighted average of the function $F(x, Y)$ at selected points in the interval $[x_n, x_{n+1}]$

(cf Mean Value Theorem)

II.3 Runge Kutta algorithms

① Let's try to generalize the method to consider algorithms of the kind

$$y_{n+1} = y_n + a k_1 + b k_2 \qquad (\text{here } y_n = Y(x_n))$$

where
$$\begin{cases} k_1 = \Delta x\, f(x_n, y_n) & \Delta x = x_{n+1} - x_n \\ k_2 = \Delta x\, f(x_n + \alpha\Delta x,\ y_n + \beta k_1) \end{cases}$$

first step is always the Euler explicit step

↑ second step is a "derivative" taken somewhere in interval $[x_n, x_{n+1}]$

→ how do we choose the coefficients a, b, α, β to minimize the error?

let's write, in general:

$$y_{n+1} = y_n + \Delta x \frac{dy}{dx} + \frac{\Delta x^2}{2}\frac{d^2y}{dx^2} + o(\Delta x^3)$$

$$= y_n + \Delta x f(x_n, y_n) + \frac{\Delta x^2}{2}\frac{d}{dx}\left(f(x_n, y_n)\right)$$

$$= y_n + \Delta x f(x_n, y_n) + \frac{\Delta x^2}{2}\left(\frac{\partial f}{\partial x} + f(x_n, y_n)\frac{\partial f}{\partial y}\right)_{(x_n, y_n)}$$

$$= y_n + \Delta x f(x_n, y_n) + \frac{\Delta x^2}{2}\left(\frac{\partial f}{\partial x} + f\frac{\partial f}{\partial y}\right)_{(x_n, y_n)}$$

and here
$$= y_n + a\Delta x f(x_n, y_n) + b\Delta x f(x_n + \alpha\Delta x,\ y_n + \beta\Delta x f(x_n, y_n))$$

$$= y_n + a\Delta x f(x_n, y_n) + b\Delta x f(x_n, y_n)$$
$$+ b\Delta x\left[\alpha\Delta x \frac{\partial f}{\partial x} + \beta\Delta x f(x_n, y_n)\frac{\partial f}{\partial y}\right]_{(x_n, y_n)}$$

$$= y_n + (a+b)\Delta x f(x_n, y_n)$$
$$+ b\alpha\Delta x^2 \frac{\partial f}{\partial x} + b\beta\Delta x^2 f(x_n, y_n)\frac{\partial f}{\partial y}$$

⇒ These two are equivalent and the order of the algorithm is $o(\Delta x^3)$ provided

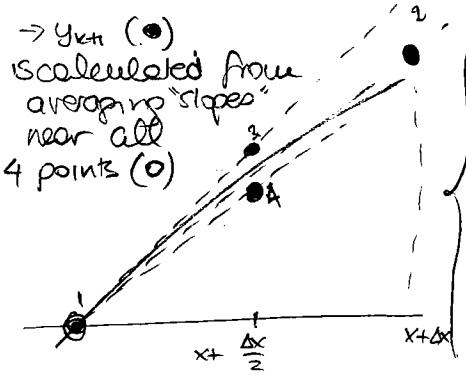
$$\boxed{a + b = 1 \qquad b\alpha = \frac{1}{2} \qquad b\beta = \frac{1}{2}}$$

$\Rightarrow$ 3 equations for 4 unknown $\Rightarrow$ there is a wide family of RK methods which have local error $O(\Delta x^3)$ (global error $O(\Delta x^2)$);

- $a = \frac{1}{2}$ $b = \frac{1}{2}$ $\alpha = 1$ $\beta = 1$ is a possibility
  $\Rightarrow$ Euler's modified algorithm
- $a = 0$ $b = 1$ $\alpha = \frac{1}{2}$ $\beta = \frac{1}{2}$
  $\Rightarrow$ the midpoint algorithm
- $a = \frac{1}{4}$ $b = \frac{3}{4}$ $\alpha = \frac{2}{3}$ $\beta = \frac{2}{3}$
  $\Rightarrow$ Heun's algorithm

If we generalize this to 4 steps, we obtain the fourth-order RK methods family.

The most famous is given by



$$
\begin{cases}
k_1 = \Delta x\, f(x_n, y_n) \\
k_2 = \Delta x\, f\left(x_n + \frac{1}{2}\Delta x,\ y_n + \frac{1}{2}k_1\right) \\
k_3 = \Delta x\, f\left(x_n + \frac{1}{2}\Delta x,\ y_n + \frac{1}{2}k_2\right) \\
k_4 = \Delta x\, f\left(x_n + \Delta x,\ y_n + k_3\right)
\end{cases}
$$

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

Note: • This method requires four evaluations of f per step. However, its convergence (accuracy) properties are far better than for simple-step Euler method taken in $\frac{\Delta x}{4}$ intervals

• The global error is now $O(\Delta x^4)$. (locally $O(\Delta x^5)$)

II.4 Stepsize control

- Adaptative stepsizes are desirable if we want to optimize calculation
  → put lots of "steps" when function varies quickly, v. few steps if it doesn't

- To do this we must design an automated way of choosing stepsize

① A "slow & dirty way"

Idea: At each step, perform 2 calculations
- 1 long step Δx
- 2 half steps $\frac{\Delta x}{2}$

→ compare the results; is err $< \epsilon_{min}$ then at next step, double Δx; is err $> \epsilon_{max}$ then at next step half Δx

Problem: this actually requires 11 evaluations of the function f per "step" Δx.
↳ CPU intensive!

② A "quick & nifty way"

Idea (From Fehlberg): (see Handout).

Compute a fifth order RK algorithm that takes 6 steps for which a recombination of the terms also yields a fourth order RK solution.

→ then an estimate of the error is given by

$$E(\Delta x) = \left| \frac{y_{n+1}^{(4)} - y_{n+1}^{(5)}}{\Delta x} \right| = \text{difference between 4th order estimate \& fifth order estimate}$$

If $E(\Delta x)$ is too large then choose another, smaller Δx and repeat.

Note: Cash & Karp have refined the Fehlberg coefficients (see Numerical Recipes).