

Numerical Solutions of Differential Equations (2)

- Numerical Stability ...
 - Implicit/Explicit methods
 - Backwards Euler
- Improved Euler/Heun's method
- Runge Kutta
- Adaptive step size methods
- Stiff systems
- System Dynamics Modelling using Netlogo

Agenda

- So far have analysed a simple scheme how to integrate differential equations and learned about sources of error
- Also learned that small step size may become problematic
- Today:
 - Learn about other issues that may arise – numerical stability
 - Better schemes that improve truncation error or deal better with stability issues

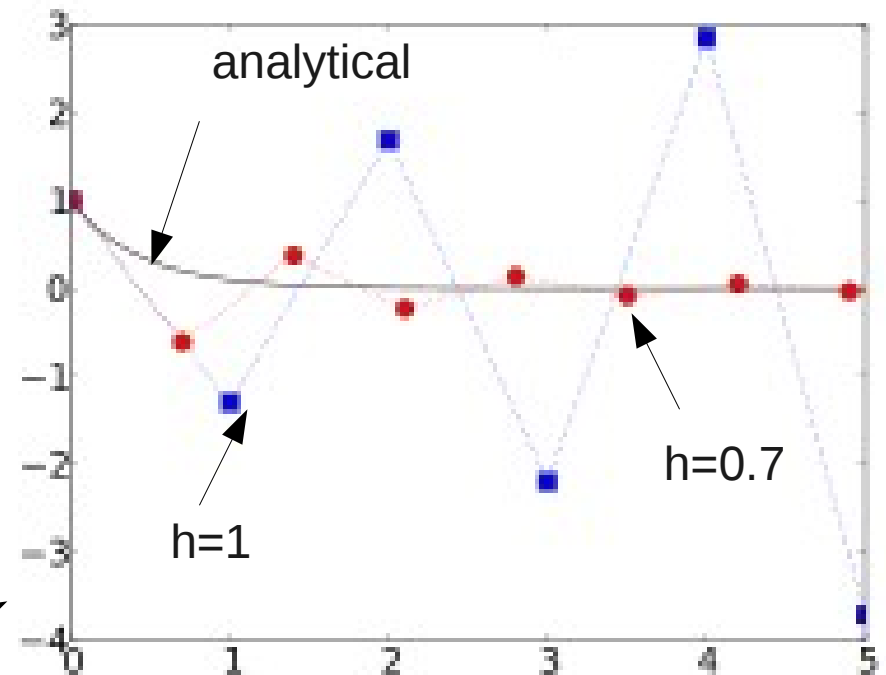
Numerical Stability of Euler Scheme

- Consider Euler scheme to solve Malthusian growth

$$dP/dt = rP, P(0) = P_0 \begin{cases} \rightarrow P(t)^{exact} = P_0 \exp(rt) \\ \rightarrow P^{Euler}(t) = (1 + hr)^{t/h} P_0 \end{cases}$$

- Suppose $r < 0$
 - For large step size h
 $1 + hr < 0 \rightarrow$ oscillations
 - If $|hr| > 1$ numerical behaviour qualitatively different from true solution!

$$dx/dt = -2.3x, x(0) = 1$$



Numerical Stability

- A method is called **numerically stable** if a small deviation from the true solution does not tend to grow as the solution is iterated
 - Let's say at some point in time numerical solution deviates from solution of the Euler method by some small amount δy due to round-off errors
 - Numerical stability: How do these errors propagate?
- Start with Euler scheme:

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

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

$$\underline{y_{n+1}} + \delta y_{n+1} = \underline{y_n} + \delta y_n + \underline{h f(y_n, t_n)} + h f_y \delta y_n$$

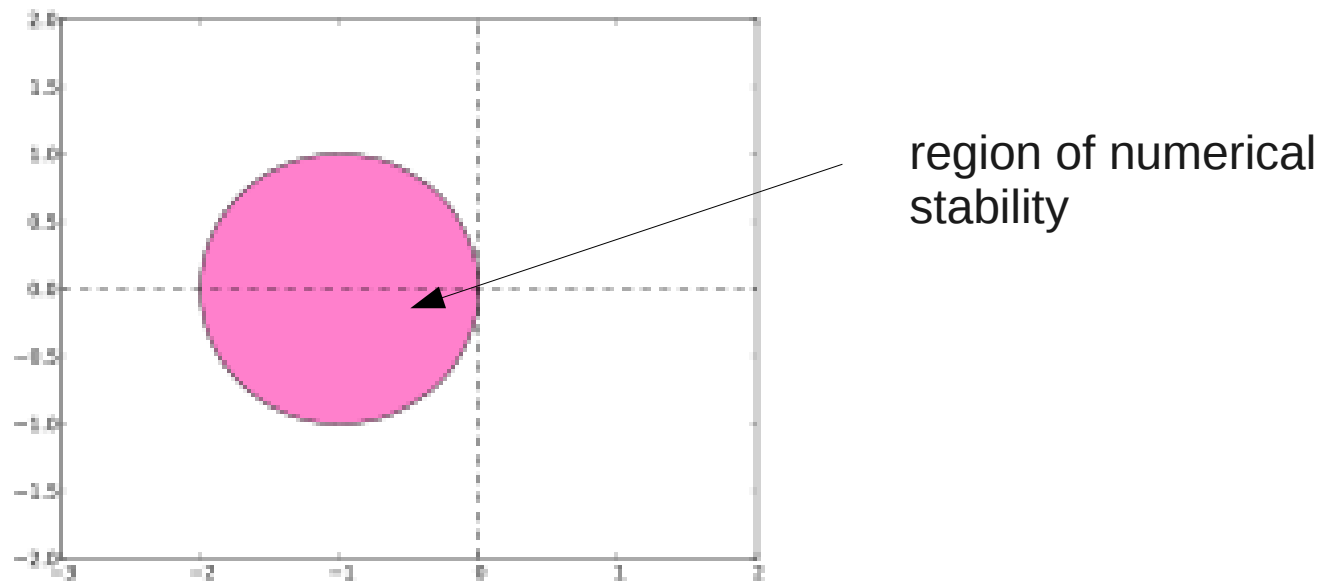
$$\delta y_{n+1} = \delta y_n (1 + h f_y)$$

Numerical Stability of Euler Scheme

- Hence: Euler scheme is numerically stable if


$$|1 + hf_y| < 1$$

- Can monitor this to ensure step size is small enough
- More generally for complex f:



Backwards Euler

- Stability problems can be avoided with *implicit methods*, e.g. Backwards Euler

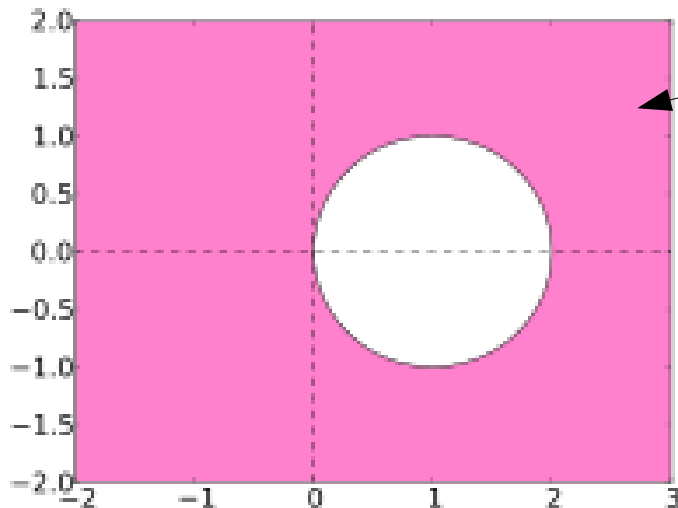
$$y_{n+1} = y_n + h f(y_{n+1}, t_n + h)$$


Note that y_{n+1} appears on left and right hand side!

- Can generally not explicitly solve for y_{n+1}
->**implicit method**

Backwards Euler

- Need some trick to determine next step
 - e.g. Fixed point iteration
$$y_{n+1}^{(0)} = y_n, y_{n+1}^{(i+1)} = y_n + h f(y_{n+1}^{(i)}, t_n + h)$$
 - Or could use Newton-Raphson or some other scheme method to find a solution
- Advantage: larger area of stability!
 - e.g. suitable for solving stiff equations (see later)



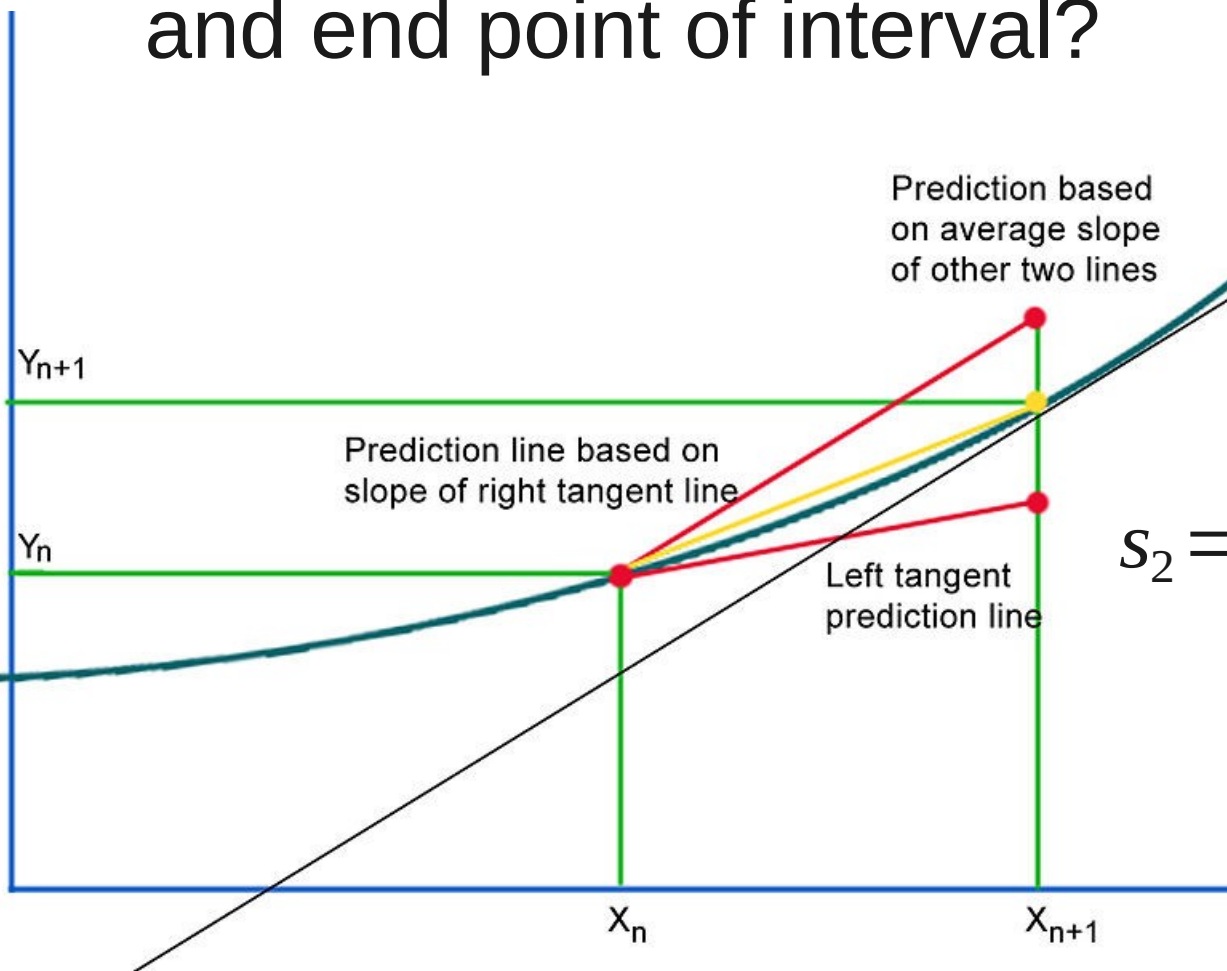
pink = stability region of backwards Euler
(why? Show this in seminar)

Explicit/Implicit methods

- **Explicit schemes:** (e.g. forward Euler)
 - Future information based on past and present information
 - Easy to program, easy to blow out!
 - To avoid instability, small time steps require (-> more computational cost)
- **Implicit schemes:** (e.g. backwards Euler)
 - Future information based on past, present, and future information
 - Difficult to program, require more memory
 - Stable in large time step
 - Depending on problem this can save CPU time

Improved Euler/Heun Scheme

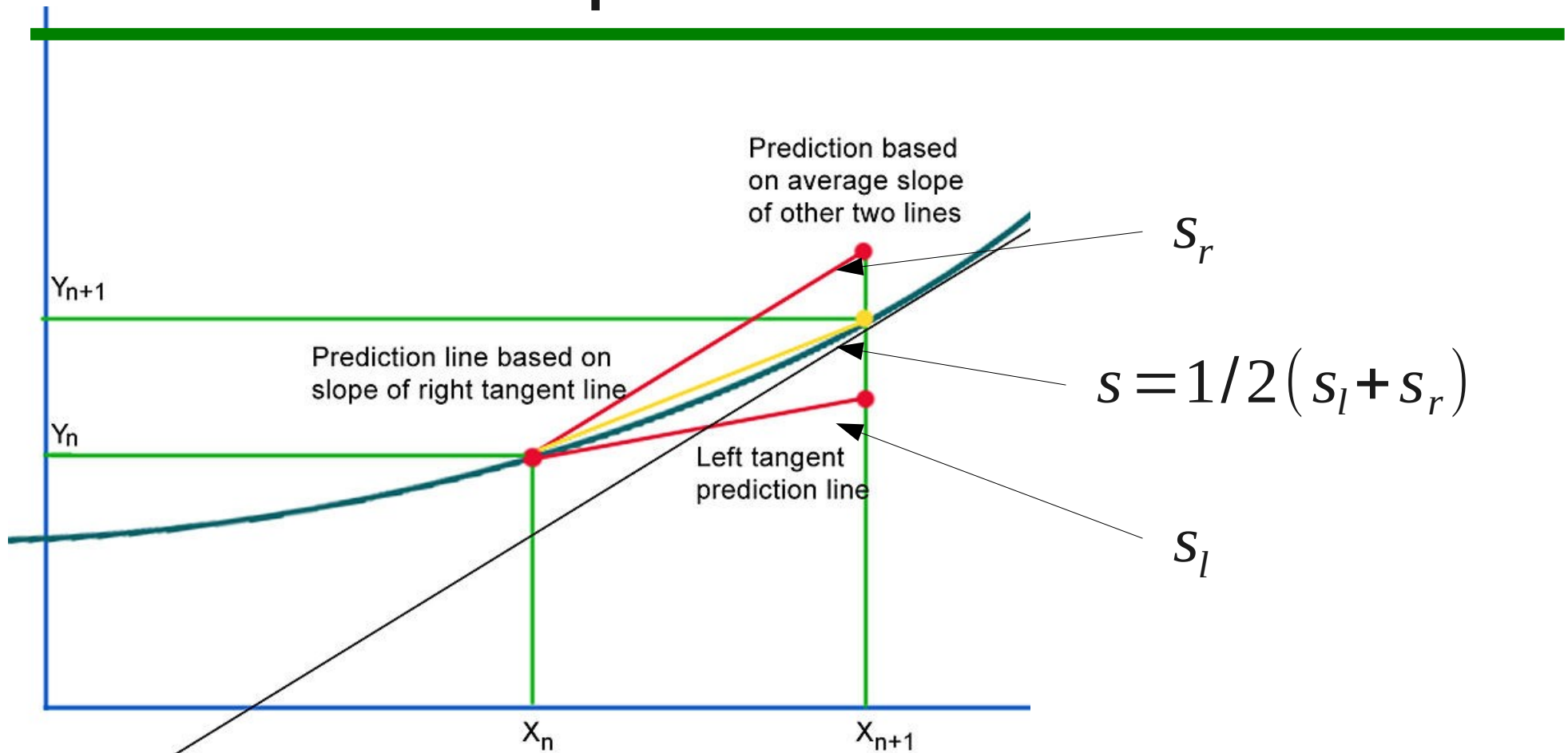
- Euler schemes use first or last point of interval to estimate derivative over the entire interval
- Maybe better: take average of estimate at start and end point of interval?



$$slope_1 = f(y_n, t_n)$$

$$s_2 = f(y_n + hf(y_n, t_n), t_n + h)$$

Improved Euler



- slope left: $s_l = f(y_n, t_n)$
- slope right: $s_r = f(y_n + h f(y_n, t_n), t_n + h)$
- Improved estimate: $s = 1/2(s_l + s_r)$

$$y_{n+1} = y_n + h/2(s_l + s_r) = y_n + h/2(f(y_n, t_n) + f(y_n + h f(y_n, t_n), t_n + h))$$

Improved Euler/Heun

- Is a predictor-corrector method:
 - First step predicts a tentative new value of y
 - Second step corrects this value
- Is a second order method
 - Local error $\sim h^3$
 - Global error $\sim h^2$
 - Try to verify this (seminar)

Order of Heun's Method

- Can write Heun's method as:

$$y_{n+1} = y_n + 1/2(K_1 + K_2) \quad (*)$$

$$K_1 = hf(y_n, t_n)$$

$$K_2 = hf(y_n + K_1, t_n + h)$$

- To check the order we expand (*) using Taylor:

$$\begin{aligned} K_2 &\approx h \left(f + hf_t + K_1 f_x + O(h^2) \right) \\ &= h \left(f + hf_t + hf f_x + O(h^2) \right) \end{aligned}$$

$$\begin{aligned} y_{n+1} &= y_n + 1/2(hf + hf + hf_t + hf f_x) \\ &= y_n + hf + 1/2 h^2(f_t + f f_x) + O(h^3) \end{aligned}$$

Order of Heun's Method

- To determine the order of the method we remember that this expression y_{n+1}

$$y_{n+1} = y_n + h f + 1/2 h^2 (f_t + f f_x) + O(h^3)$$

is meant to approximate $y(t+h)$ which can be expanded into a Taylor series in h :

$$\begin{aligned} y(t_n + h) &\approx y(t_n) + h y'(t_n) + 1/2 h^2 y''(t_n) + O(h^3) \\ &= y(t_n) + h f + 1/2 h^2 df(y, t)/dt + O(h^3) \\ &= y(t_n) + h f + 1/2 h^2 [f_t + f_y y'] + O(h^3) \\ &= y(t_n) + h f + 1/2 h^2 [f_t + f_y f] + O(h^3) \end{aligned}$$

- I.e. Expansion of Heun's method and of exact solution coincide up to and including 2nd order

General Runge Kutta Methods

- In principle, we can use the idea which we used to check the order of Heun's method to develop methods of much higher order accuracy
- General Runge-Kutta methods can be defined as follows:

$$y_{n+1} = y_n + w_1 K_1 + w_2 K_2 + \dots + w_m K_m$$

$$K_1 = h f(y_n, t_n)$$

$$K_2 = h f(y_n + b_2 K_1, t_n + a_2 h)$$

$$K_3 = h f(y_n + b_3 K_1 + c_3 K_2, t_n + a_3 h)$$

...

$$K_m = h f\left(y_n + \sum_{i=1}^{m-1} K_i \phi_i, t_n + a_m h\right)$$

- An carefully tuned $a_i, b_i, c_i, \phi_i, w_i$

Standard Today: 4th Order RK

- Trade-off:
 - improving truncation error comes at increasing computational costs
 - Pragmatic solution: go to 4th order accuracy

$$y_{n+1} = y_n + 1/6 [K_1 + 2K_2 + 2K_3 + K_4]$$

$$K_1 = hf(y_n, t_n)$$

$$K_2 = hf(y_n + 1/2 K_1, t_n + h/2)$$

$$K_3 = hf(y_n + 1/2 K_2, t_n + h/2)$$

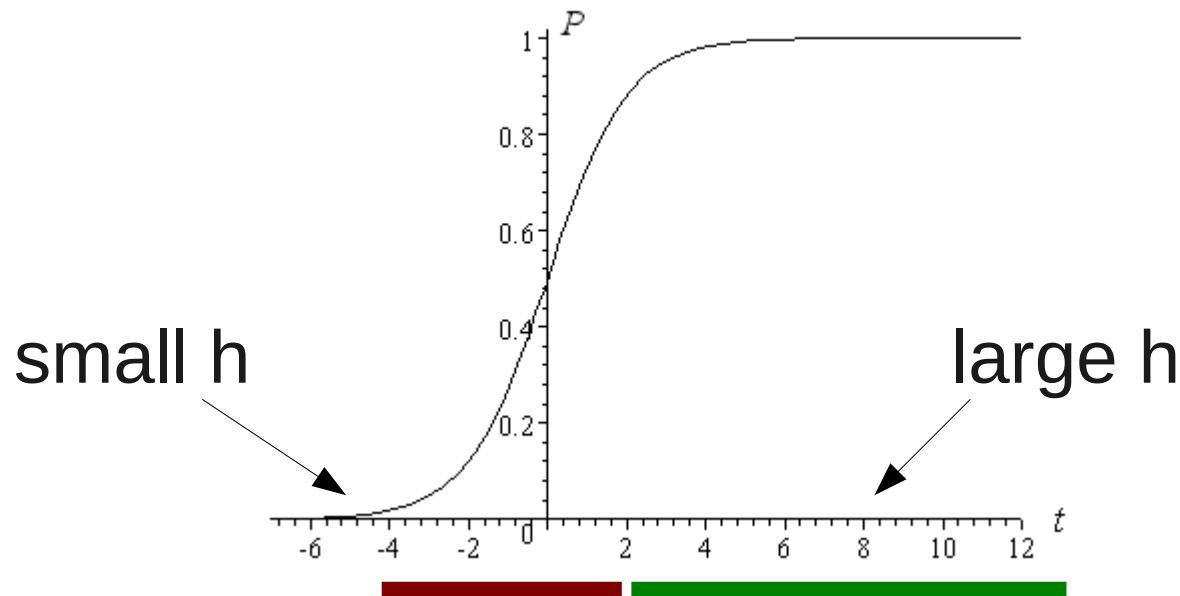
$$K_4 = hf(y_n + K_3, t_n + h)$$

What About Higher Order ODEs or Systems of ODEs?

- Remember:
 - Can always write a higher order ODE as a system of ODEs
 - E.g. $y'' + y = 1 \longrightarrow \begin{aligned} y' &= x \\ x' &= 1 - y \end{aligned}$
- Systems: $\vec{y}' = \vec{f}(\vec{y}, t)$
 - Use RK as before and interpret y's and K's as vectors

Adaptive Step Size Methods

- General dilemma:
 - Smaller step size less truncation error, but more steps required \rightarrow more steps \rightarrow larger round-off errors
- Better:
 - Only use small step size when required and use large step size if f does not vary much



Adaptive Step Size Methods

- Optimal:
 - Tune step size such that error is the same for all steps
 - How to get error estimates?
- Idea:
 - Calculate $y(t+h)$ from $t(t)$ and $y_1(t+h)$ from $y(t+h/2)$ (i.e. by combining two half steps)
 - $|y-y_1|$ gives an estimate of the error
 - If error \gg tol: half step size
 - If error \ll tol: double step size
 - If error \sim tol: leave step size as it is
- This is wasteful of computing time

An Example: Runge Kutta Fehlberg

- Better:
 - Use a fourth order RK method and add a term to calculate 5th order RK
 - Use difference as error estimate
 - Pseudocode:

```
Given t0, x0, h0, nmax, emin, emax, hmin, hmax
set h = h0, x = x0, k = 0,
while k < nmax do
    if h < hmin then h = hmin
    else if h > hmax then h = hmax
    end
    Compute RK4, RK5, and e = |RK4 - RK5|
    if e > emax, then h = h/2;
    else
        k=k+1; t=t+h; x=RK5;
        If e < emin, the h = 2 * h; end
    end
end (while)
```

Stiff Systems

- A system is called **stiff** if the solution contains components that vary with very different speed/frequency
 - No really accepted formal definition of stiffness
 - Implies that numerical integration with certain methods becomes unstable unless step size is chosen extremely small
 - Usually: step size needs to be small if solutions vary a lot, otherwise step size can be larger. In stiff systems very small step size might be required for numerical stability even though solutions don't vary much
 - Stiff systems should be integrated with implicit methods

Example of a Stiff System

- Consider:
$$\begin{aligned}x' &= -20x - 19y \\ y' &= -19x - 20y\end{aligned} \quad x(0)=2, y(0)=0$$
- Can show that exact solution is (seminar)
$$\begin{aligned}x(t) &= \exp(-39t) + \exp(-t) \\ y(t) &= \exp(-39t) - \exp(-t)\end{aligned}$$
- Observations:
 - $x \rightarrow 0, y \rightarrow 0$ as $t \rightarrow \infty$
 - Two components $\exp(-39t)$, $\exp(-t)$ that vary at very different rate
 - $t \gg 0$: $\exp(-t)$ dominates, $\exp(-39t)$ is a *transient* term

Stiff Systems Example (2)

- Euler scheme:

$$x_{n+1} = x_n + h(-20x_n - 19y_n)$$

$$y_{n+1} = y_n + h(-19x_n - 20y_n)$$

- Can show by induction that the solution is

$$x_n = (1 - 39h)^n + (1 - h)^n$$

$$y_n = (1 - 39h)^n + (1 - h)^n$$

- For convergence to true solution we require $x_n \rightarrow 0$ and $y_n \rightarrow 0$ for n to infinity
- I.e. need: $|1 - 39h| < 1$ and $|1 - h| < 1$

Stiff Systems Example (3)

- Or $h < 2/39$ and $h < 2$
- Hence:
 - First condition corresponds to transient term that quickly tends to zero for large times
 - First condition much stronger than second, i.e. need step size $h < 2/39$
 - But: even though the transient term does not influence the solution much for large times it dictates a requirement for very small step size to guarantee numerical stability!
- Problem: Show that an implicit scheme works with large step size here.

Summary

- Important points to remember:
 - Stability of numerical schemes
 - Implicit/explicit schemes
 - Idea of Backwards Euler
 - Improving truncation error:
 - Idea and order of Heun's scheme
 - Idea of Runge-Kutta
 - Adaptive step size methods
- Stiff systems – what it is and how to deal with stiff systems