

Recap

Terminology recap

- Variable or state
- Differential equation
- Initial condition
- Trajectory
- Parameter
- Steady state
- Transient behaviour
- Perturbation
- Ordinary differential equations (ODE)
- 3 dimensional ODE

Terminology recap

- Phase space/state space
- Vectorfield
- Fixed point (stable/unstable, focus/node)
- Nullcline
- Saddles, separatrix
- Bistability

Dynamical Systems

Yujiang.wang@ncl.ac.uk

Lecture 3 of 5

Overview

- What are dynamical systems?
- How to interpret a differential equation
- How to analyse differential equation systems
- **How to solve differential equation systems**
- Stability analysis, multistability
- Oscillatory solutions
- Parameter variations, bifurcations
- Choice of cool stuff: Chaos, turbulence, spatio-temporal systems, slow-fast systems, transients, and more.

Numerically solving differential equations

Numerically solving differential equations

- Why?
 - Very rare that we can analytically solve equations
 - Implementation speed
 - Convenience
- Why not?
 - Sometimes long simulation times
 - Inaccuracies
 - Variations between different solvers

Numerically solving differential equations: methods

- Euler
- Heun
- Runge-kutta

Euler method

Solves in a fixed step ($h=1$) iterative manner

Example equation: $dy/dt=f(t,y)$

Lets say initial condition, $y_0 = 1$ and $f(t,y)=y$

If we start with $y=0$ at $time(t)=0$, how much does y change between $t=0$ and $t=4$?


Euler method

Solves in a fixed step ($h=1$) iterative manner

Example equation: $dy/dt=f(y)$

Lets say initial condition, $y_0 = 1$ and $f(y)=y$

Time ->


$$y_1 = y_0 + hf(y_0) = 1 + 1 \cdot 1 = 2.$$

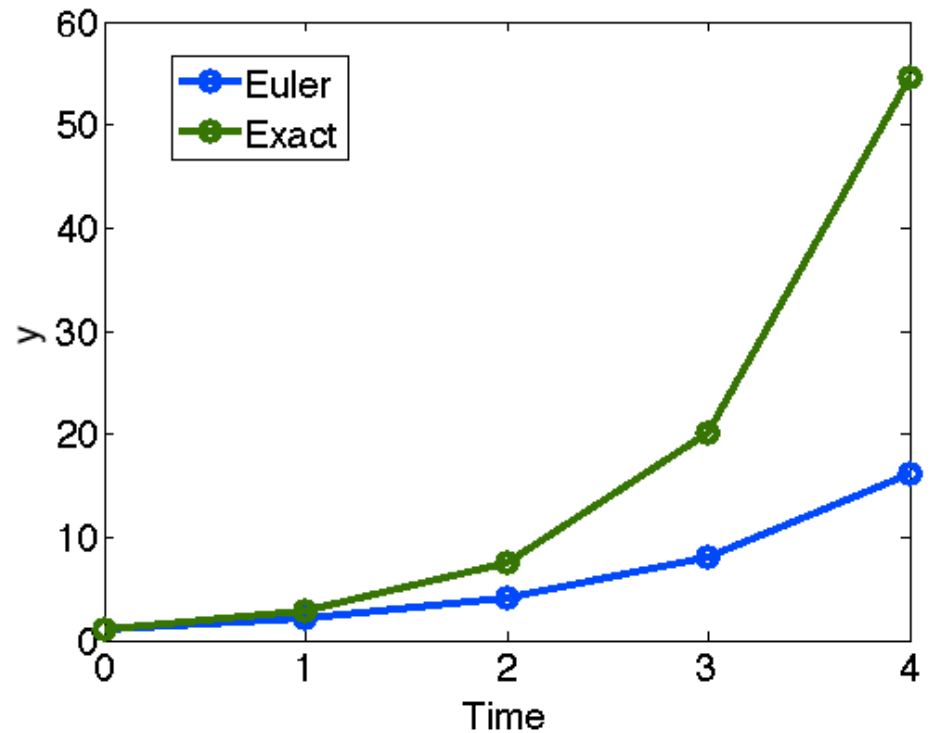
$$y_2 = y_1 + hf(y_1) = 2 + 1 \cdot 2 = 4,$$

$$y_3 = y_2 + hf(y_2) = 4 + 1 \cdot 4 = 8,$$

$$y_4 = y_3 + hf(y_3) = 8 + 1 \cdot 8 = 16.$$

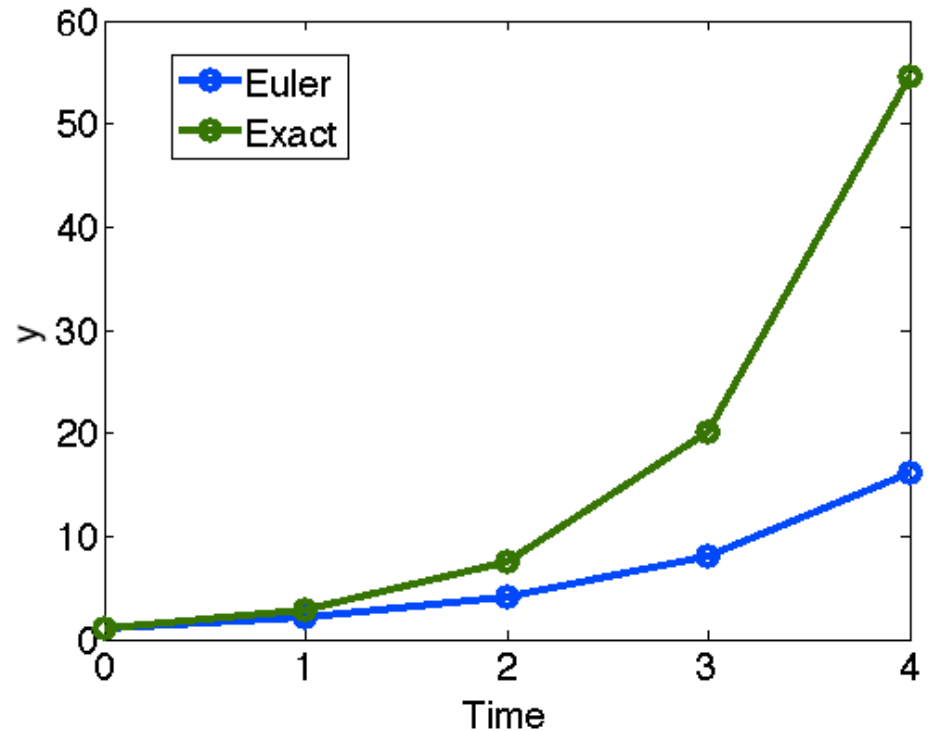
Euler method

- Not very accurate if h is too large

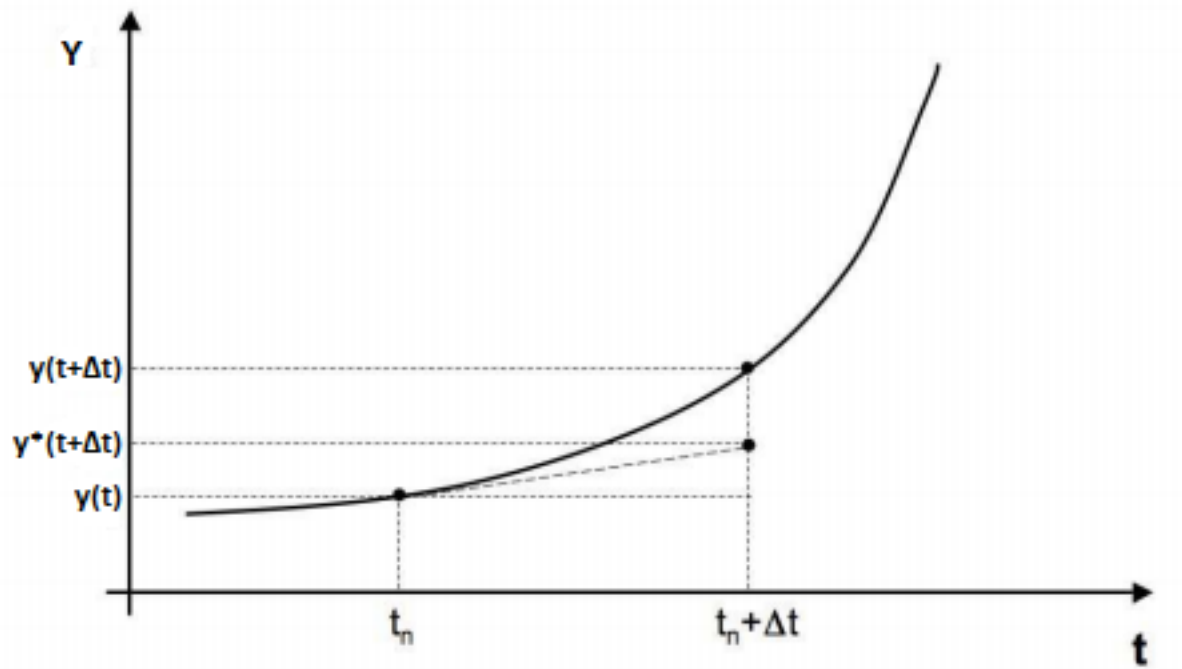


Euler method

- Not very accurate if h is too large
- Alternatively: Not very accurate if the change in y , relative to the change in t (i.e. h) is too large

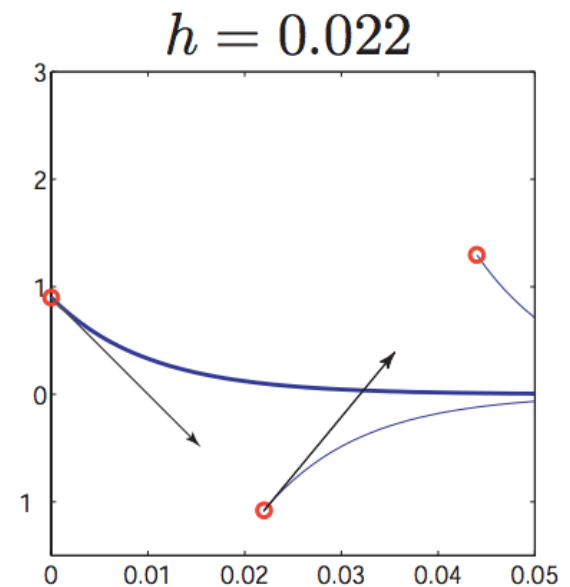
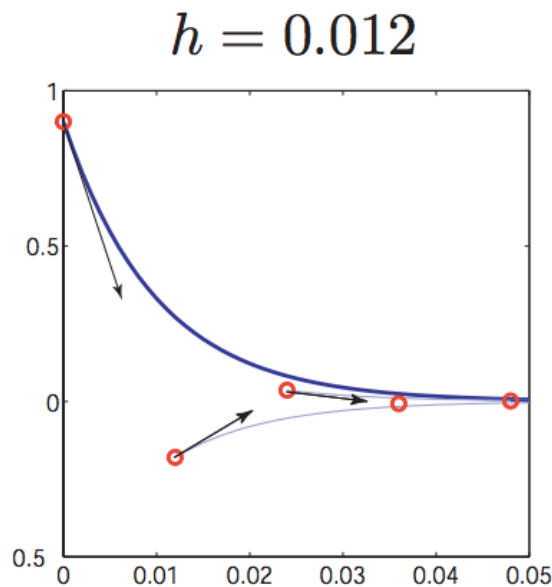
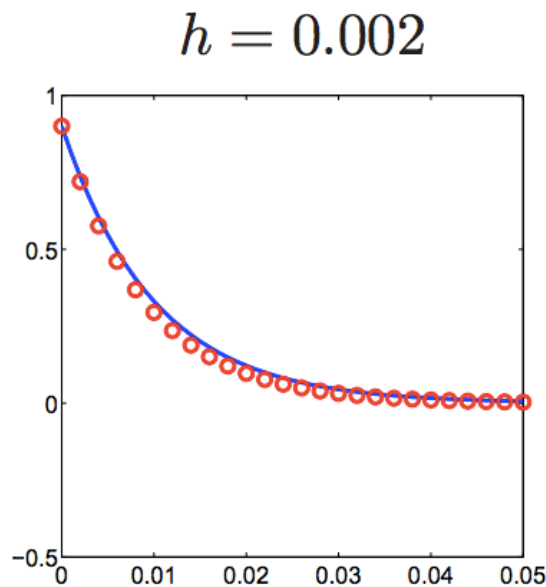


Euler method



Euler method: another example

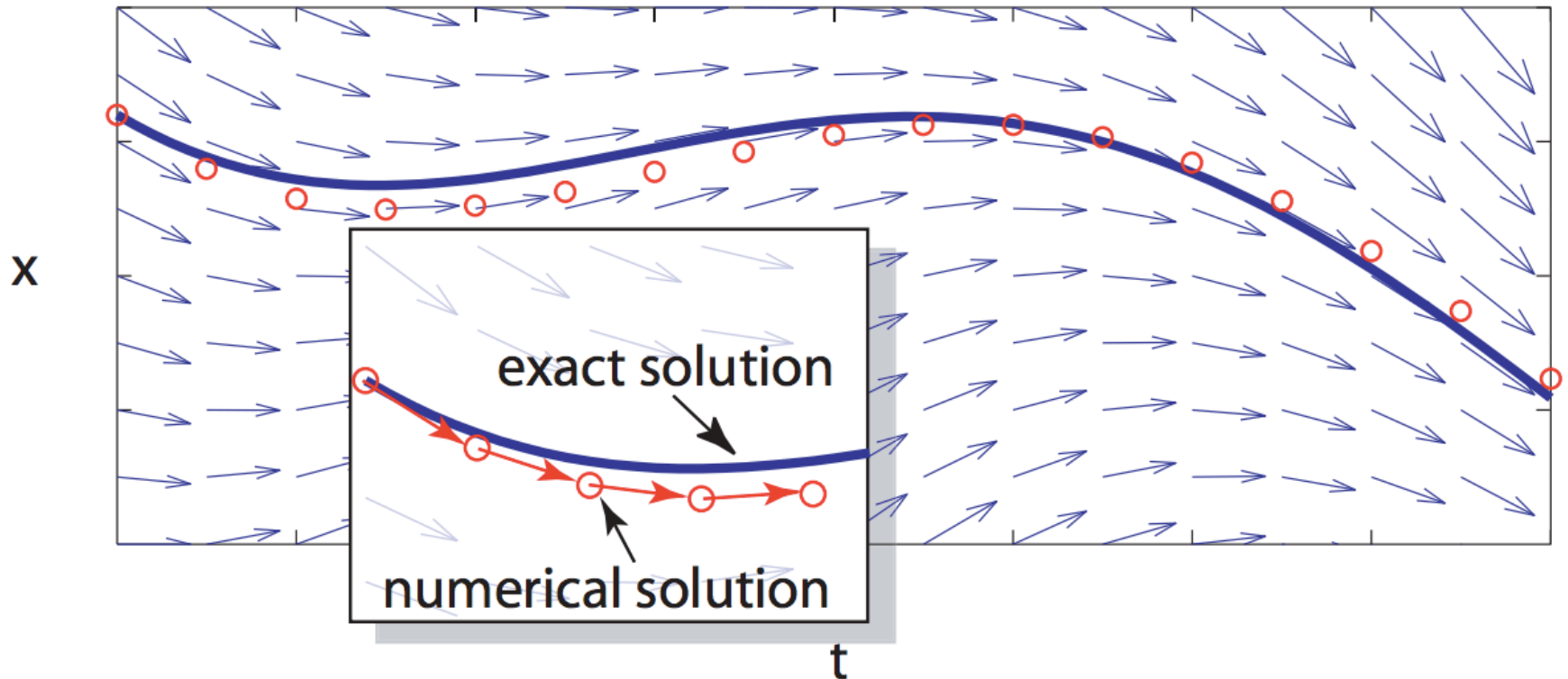
Example: Euler's method on test equation: $\dot{x} = -100x$.



Numerically stable for $h < 0.02$ (but inaccurate long before)

Euler method: another example

- In phase space



Euler method: code

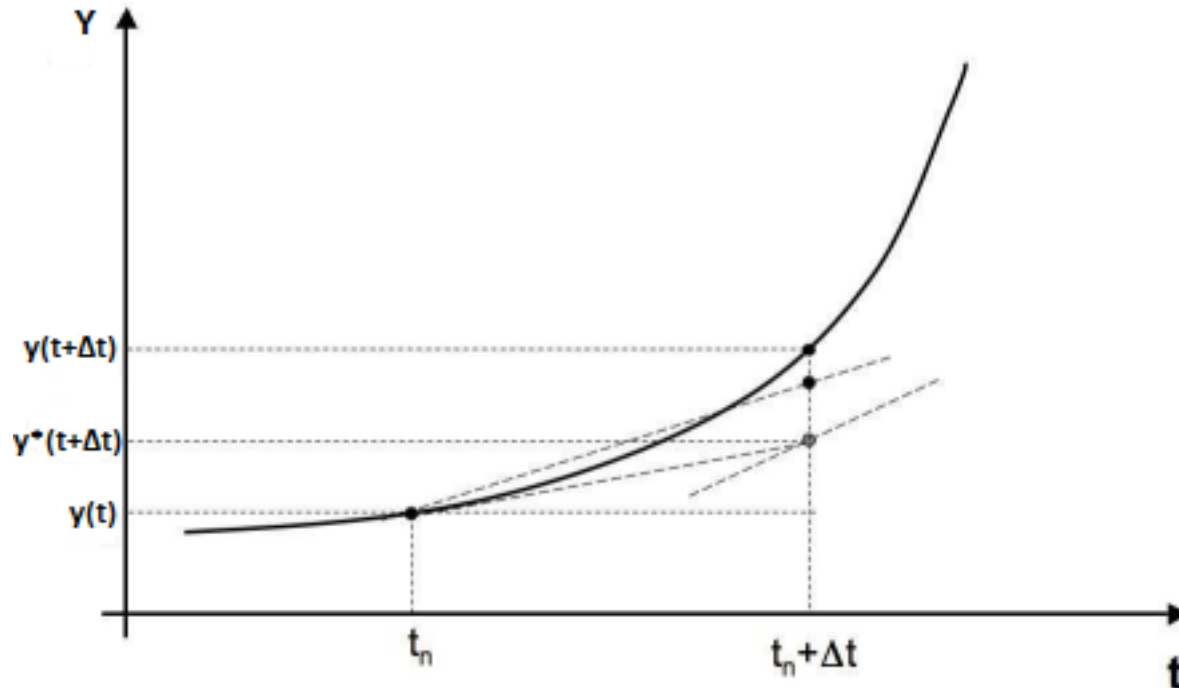
```
1 function Y = ode1(odefun,tspan,y0,varargin)
2 %ODE1 Solve differential equations with a non-adaptive method of order 1.
3 % Y = ODE1(ODEFUN,TSPAN,Y0) with TSPAN = [T1, T2, T3, ... TN] integrates
4 % the system of differential equations  $y' = f(t,y)$  by stepping from T0 to
5 % T1 to TN. Function ODEFUN(T,Y) must return f(t,y) in a column vector.
6 % The vector Y0 is the initial conditions at T0. Each row in the solution
7 % array Y corresponds to a time specified in TSPAN.
8 %
9 % Y = ODE1(ODEFUN,TSPAN,Y0,P1,P2...) passes the additional parameters
10 % P1,P2... to the derivative function as ODEFUN(T,Y,P1,P2...).
11 %
12 % This is a non-adaptive solver. The step sequence is determined by TSPAN.
13 % The solver implements the forward Euler method of order 1.
14
15 h = diff(tspan); % time step
16
17 neq = length(y0); % number of equations
18 N = length(tspan); % number of time steps
19 Y = zeros(neq,N); % pre-allocate neq X time matrix
20
21 Y(:,1) = y0; % set first to the initial conditions
22 for i = 1:N-1 % for each time step...
23     Y(:,i+1) = Y(:,i) + h(i)*feval(odefun,tspan(i),Y(:,i),varargin{:});
24 end
25 Y = Y.';
26
```


Euler method

- Known as a fixed step solver since h is a constant (in these examples always $h=1$)
- Easy to implement
- Predictable runtimes (scales linearly with number of time steps)
- Easily adapted to incorporate delays e.g. where $dy/dt=f(y,t-\tau)$
- Easily adapted to incorporate noise e.g. $dy/dt=f(y,t)+w$
- Can be slow and inaccurate compared to other solvers...

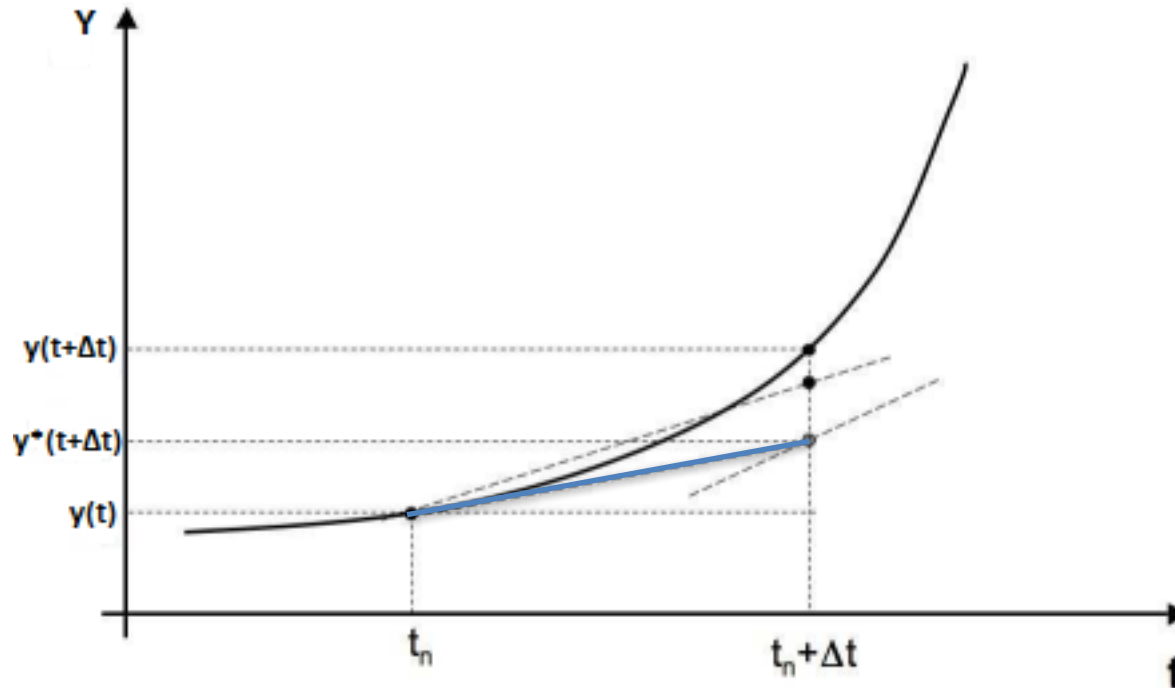
Heun's method

- Uses information from two points
 - Change in y at $y(t)$
 - Change in y at predicted $y(t+\Delta t)$
 - $dy/dt = y(t) + h/2 (f(y,t) + f(t+\Delta t, y + h f(y,t)))$



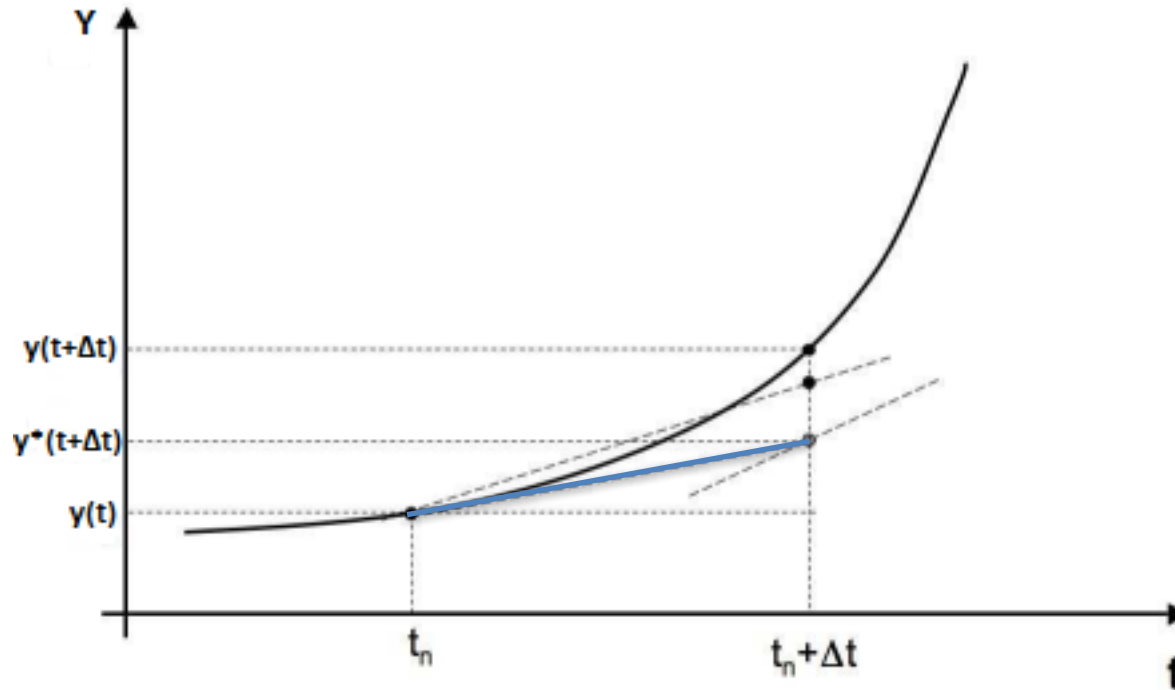
Heun's method

- Uses information from two points
 - Change in y at $y(t)$
 - Change in y at predicted $y(\Delta t)$
 - $dy/dt = y(t) + h/2 (f(y,t) + f(t+\Delta t, y+h f(y,t)))$



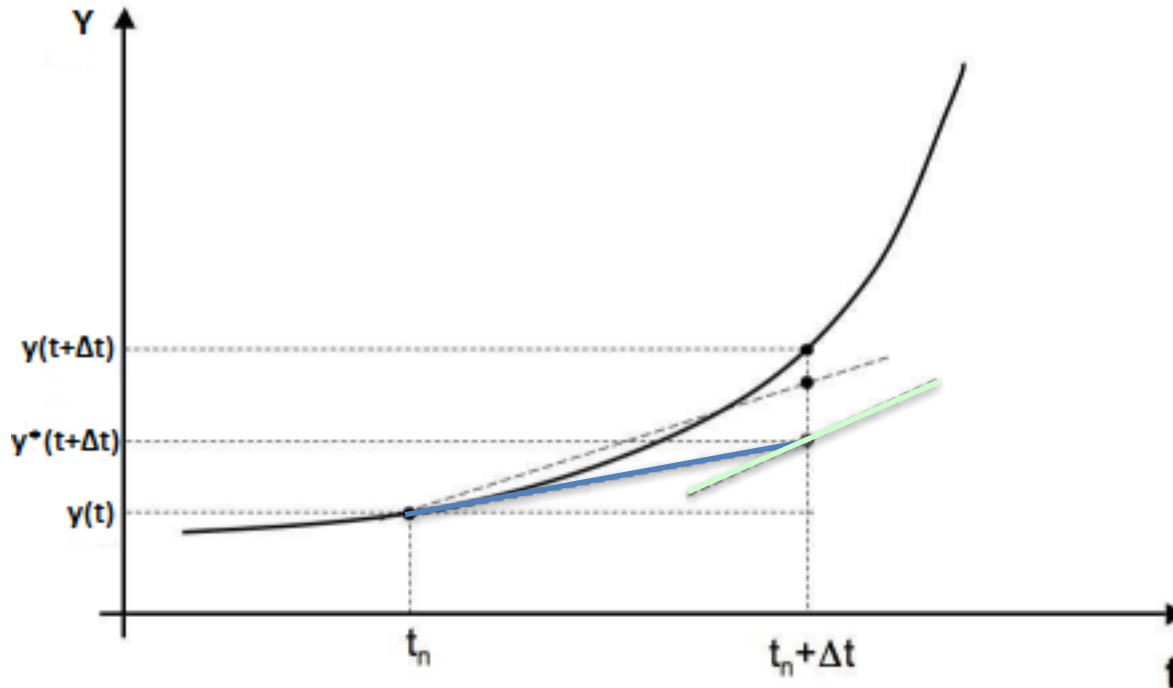
Heun's method

- Uses information from two points
 - Change in y at $y(t)$ → Same as Euler's method
 - Change in y at predicted $y(\Delta t)$
 - $dy/dt = y(t) + h/2 (f(y, t) + f(t + \Delta t, y + h f(y, t)))$



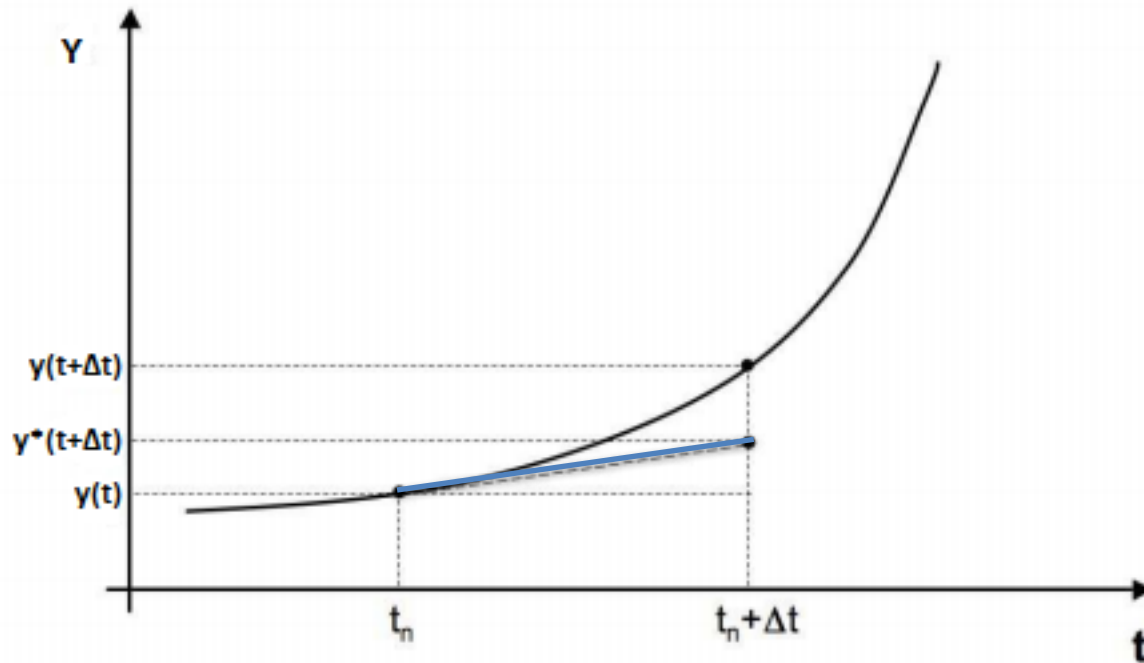
Heun's method

- Uses information from two points
 - Change in y at $y(t)$
 - Change in y at predicted $y(\Delta t)$
 - $dy/dt = y(t) + h/2 (f(y,t) + f(t+\Delta t, y+h f(y,t)))$

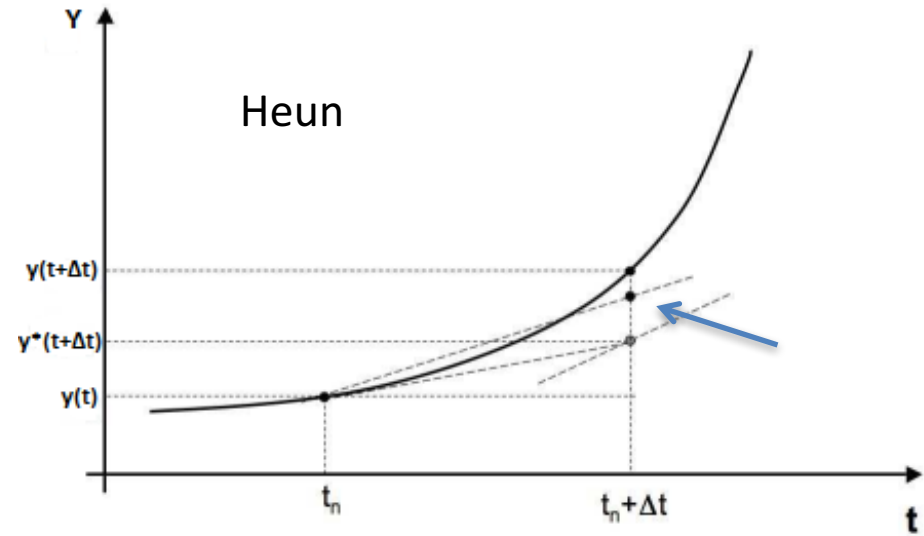
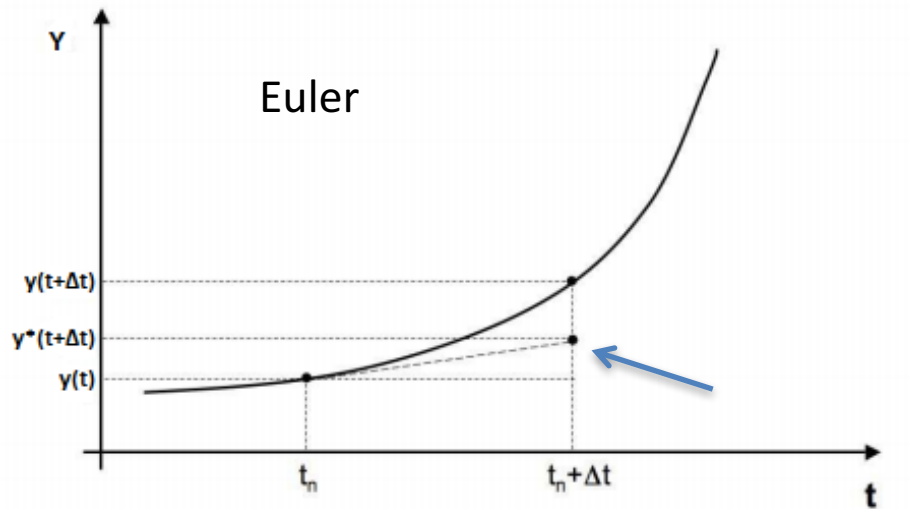


Euler's method

- Uses information from one point
 - Change in y at $y(t)$

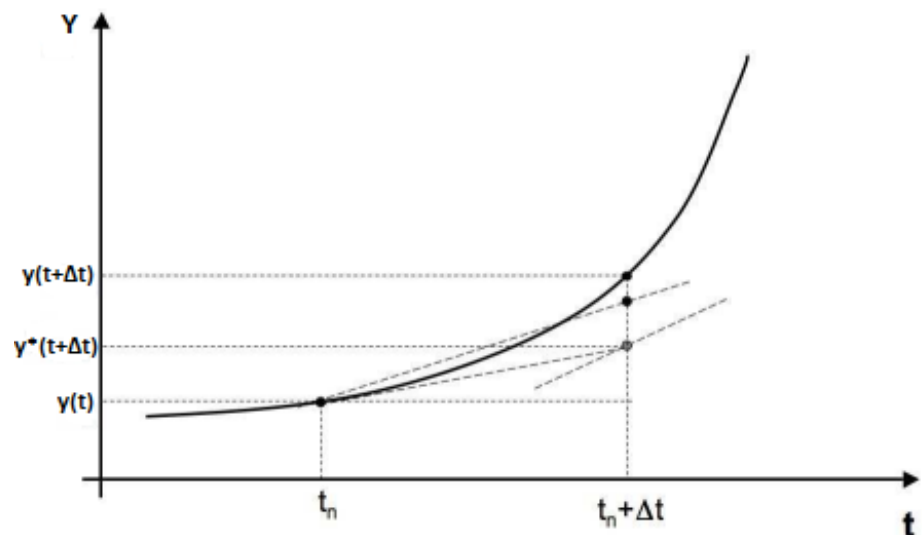


Comparing Euler and Heun methods



Heun's method code

```
24 - h = diff(tspan);
25 - neq = length(y0);
26 - N = length(tspan);
27 - Y = zeros(neq,N);
28 - F = zeros(neq,2);
29
30 - Y(:,1) = y0;
31 - for i = 2:N
32 -     ti = tspan(i-1);
33 -     hi = h(i-1);
34 -     yi = Y(:,i-1);
35 -     F(:,1) = feval(odefun,ti,yi,varargin{:});
36 -     F(:,2) = feval(odefun,ti+hi,yi+hi*F(:,1),varargin{:});
37 -     Y(:,i) = yi + (hi/2)* (F(:,1) + F(:,2));
38 - end
```

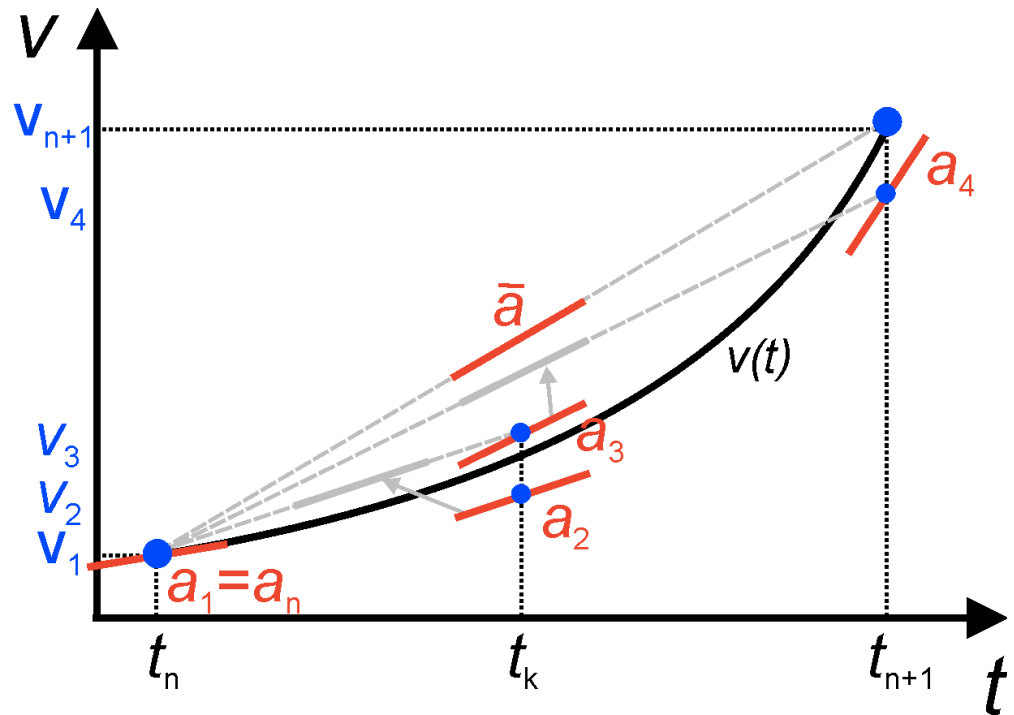


Heun's method

- Known as a second order method
- Is more computationally expensive than Euler's method for the same step size (two function evaluations)
- Outperforms Euler's method for the same step size
- Can incorporate noise (more complicated though)
- Can incorporate delays (again, more complicated)
- Still not the best though...

Runge Kutta

- Fourth order solver
- In Heun's method the mean between the start and end points is taken
- In Runge Kutta different weights are given to different points

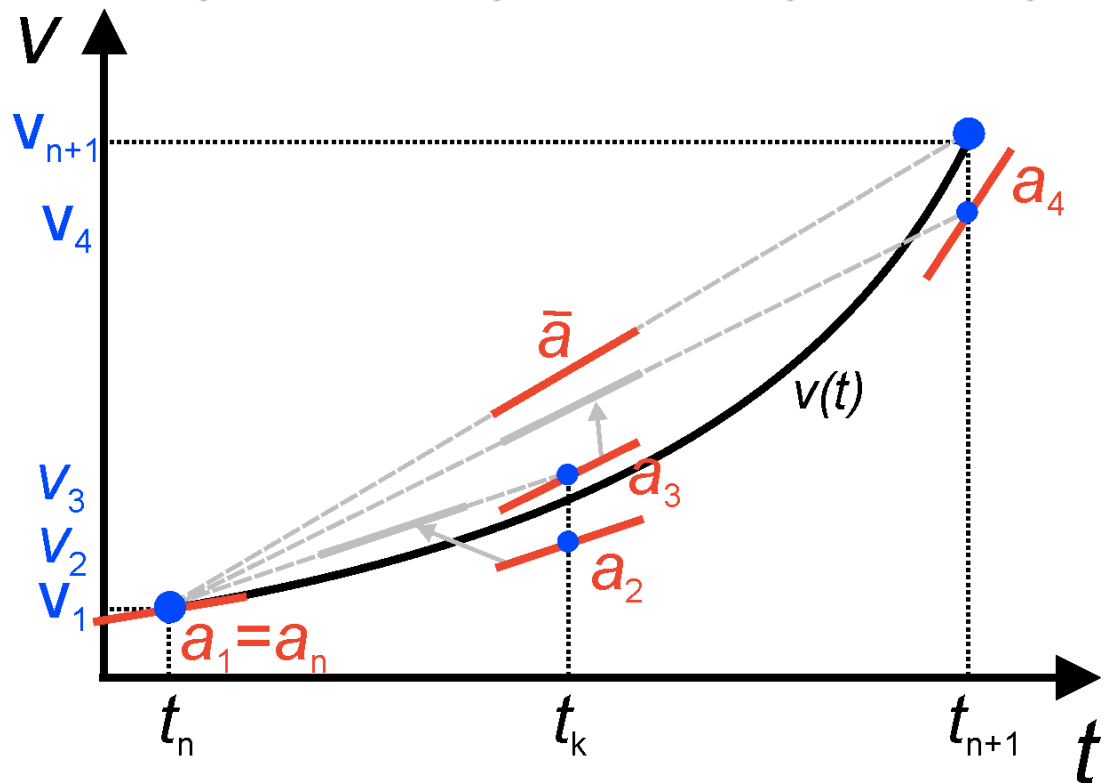


Runge Kutta code

```

55 - for i = 2:N
56 -     ti = tspan(i-1);
57 -     hi = h(i-1);
58 -     yi = Y(:,i-1);
59 -     F(:,1) = feval(odefun,ti,yi,varargin{:});
60 -     F(:,2) = feval(odefun,ti+0.5*hi,yi+0.5*hi*F(:,1),varargin{:});
61 -     F(:,3) = feval(odefun,ti+0.5*hi,yi+0.5*hi*F(:,2),varargin{:});
62 -     F(:,4) = feval(odefun,tspan(i),yi+hi*F(:,3),varargin{:});
63 -     Y(:,i) = yi + (hi/6)*(F(:,1) + 2*F(:,2) + 2*F(:,3) + F(:,4));
64 - end

```



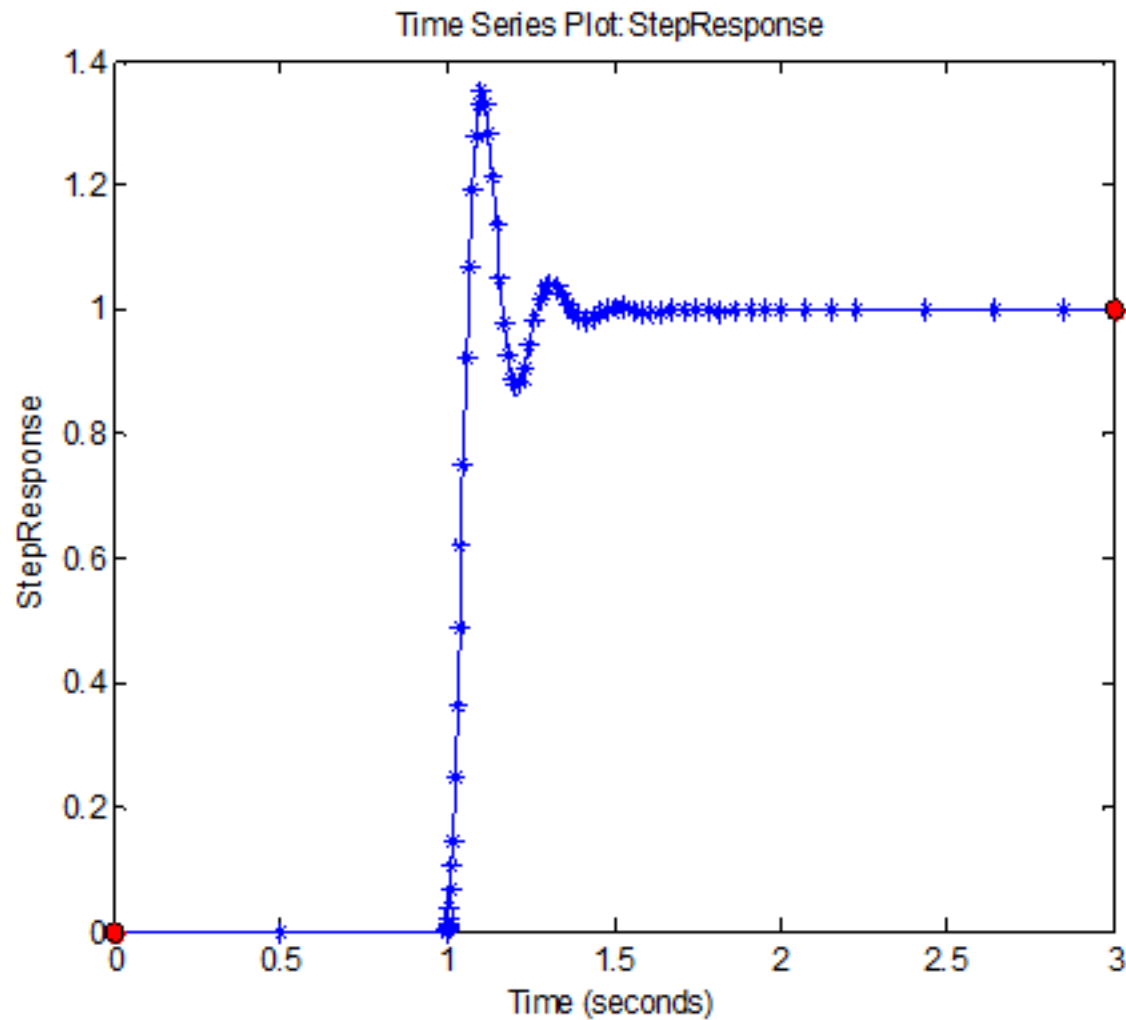
Runge Kutta's method

- Known as a fourth order method
- Is more computationally expensive than Euler's method for the same step size (**four** function evaluations)
- Outperforms Euler's method for the same step size
- Outperforms Heun's method
- Can incorporate delays (complicated)
- **Difficult & complicated to include noise (ongoing research)**

Error tolerance

- Can estimate the error made at each step (in an iterative manner)
- This error is called the absolute error
- Can then calculate the relative error (absolute error/current state)
- We can tell the Matlab solvers what errors we can tolerate:
- `options = odeset('RelTol',1e-3,'AbsTol',1e-6)`
`ode45(@odefunc,timespan,initialConditions,options,...)`

Variable step solvers



Final words of caution

- When simulating a new system it is always worth checking the results with different solvers and error tolerance settings
- Especially when expect your dynamics to change slowly, but with sudden fast bursts of activity
- Numerical solutions are ALWAYS ONLY AN APPROXIMATION