Euler's Method is just one example of a class of methods for solving ODE's.

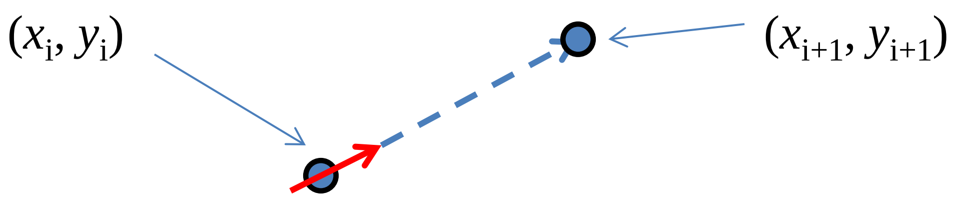
All start with some initial point (x_0, y_0)

All repeatedly apply an iterative formula of the form $y_{i+1} = y_i + \varphi\, \Delta x$

The methods differ only in how φ (the assumed effective slope) is chosen.

For Euler's Method $\varphi = f\,(x_i, y_i)$ where $f(x,y) = dy/dx$

The slope at the beginning of the interval is used as φ (i.e. the slope at the beginning on the interval is assumed to represent the slope over the interval).



Truncation errors (as previously noted): local $O(h^2)$, global $O(h)$

Heun's Method (without iteration)

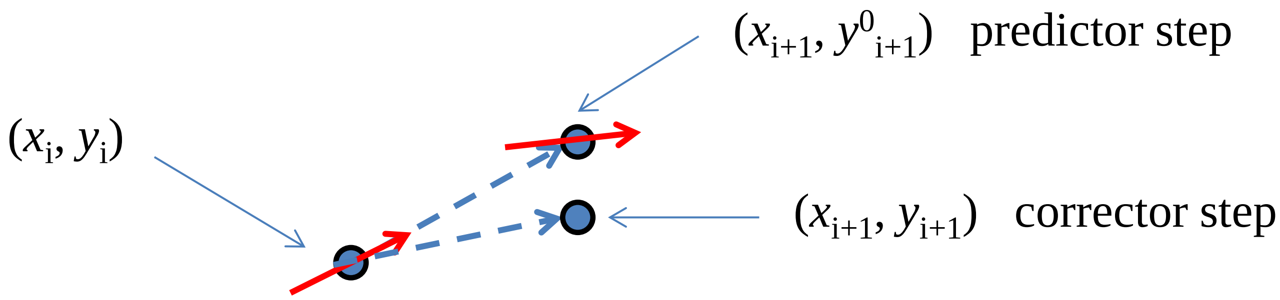
Basic idea: use average of slopes at start and end of interval.

Use Euler's method to get initial estimate of y_{i+1} (y^0_{i+1})

Use average of slopes at (x_i, y_i) and (x_{i+1}, y^0_{i+1}) as φ

$$\varphi = [\, f(x_i, y_i) + f(x_{i+1}, y^0_{i+1}) \,] / 2$$

This is an example of a *predictor*/*corrector* algorithm.



Truncation errors: local $O(h^3)$, global $O(h^2)$

Heun's Method (with iteration)

Basic idea: Repeatedly use latest estimate of y_{i+1} to recalculate φ and y_{i+1} until changes in y_{i+1} become acceptably small.

$\varphi = f(x_i, y_i)$ % initial estimate of φ

$y^0_{i+1} = y_i + \varphi \, \Delta x$ % initial estimate of y_{i+1}

for $k = 1 : \infty$

$\varphi = [\, f(x_i, y_i) + f(x_{i+1}, y^{k-1}_{i+1}) \,] / 2$ % update estimate of φ

$y^k_{i+1} = y_i + \varphi \, \Delta x$ % update estimate of y_{i+1}

if (abs $(y^k_{i+1} - y^{k-1}_{i+1}) / y^k_{i+1}$ < some limit) break

end

y^k_{i+1} is the k^{th} estimate of y_{i+1}

Always breaking out of the loop on the first iteration gives the basic Heun's method (no iteration).

In this case y^1_{i+1} is the final estimate of y_{i+1}.

Midpoint Method

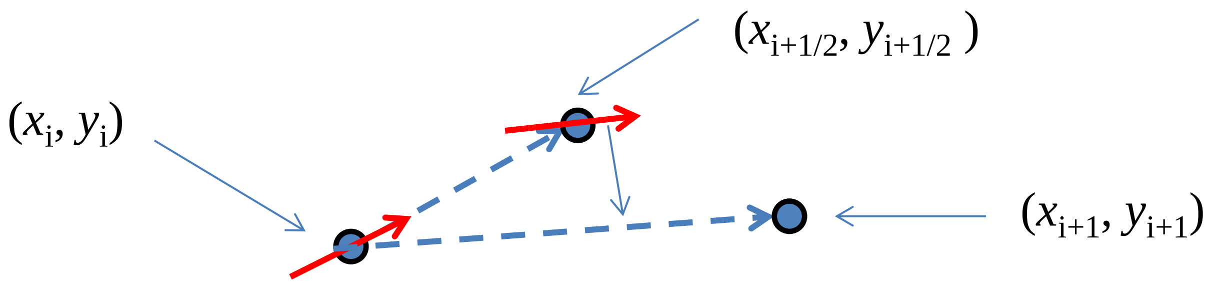
Basic idea: use slope in middle of interval

Use Euler's method to estimate $y_{i+1/2}$ (note the ½ - this is a half step)

$$y_{i+1/2} = y_i + f(x_i, y_i)(\Delta x/2)$$

Use slope at $(x_{i+1/2}, y_{i+1/2})$ as φ

$$\varphi = f(x_{i+1/2}, y_{i+1/2})$$



Truncation errors: local $O(h^3)$, global $O(h^2)$

Runge-Kutta Methods

All of the methods we've looked at are Runge-Kutta methods.

General form:

$y_{i+1} = y_i + \varphi\, h$ where $h = \Delta x$

$\varphi = a_1 k_1 + a_2 k_2 + a_3 k_3 + \ldots + a_n k_n$

$k_1, k_2, k_3 \ldots + k_n$ are slope estimates
$a_1, a_2, a_3 \ldots + a_n$ are corresponding weights

$k_1 = f(x_i, y_i)$
$k_2 = f(x_i + p_1 h,\ y_i + q_{11} k_1 h)$
$k_3 = f(x_i + p_2 h,\ y_i + q_{21} k_1 h + q_{22} k_2 h)$

p_i and q_{ij} are constants (many possible choices)

Each slope estimate makes use of preceding slope estimates.
"Order" of method = number of terms in series for φ

Euler:

1st order Runge-Kutta

Heun without iteration:

2nd order Runge-Kutta

$a_1 = ½ \quad a_2 = ½ \quad p_1 = 1 \quad q_{11} = 1$

Midpoint:

2nd order Runge-Kutta

$a_1 = 0 \quad a_2 = 1 \quad p_1 = ½ \quad q_{11} = ½$

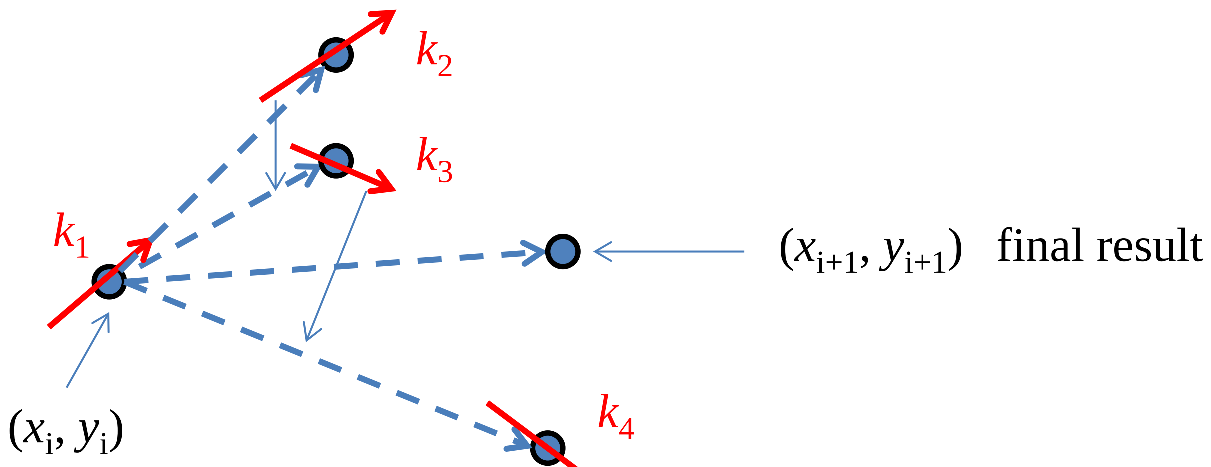
Matlab function *ode45* makes use of a combination of 4th and 5th order Runge-Kutta methods, while *ode23* makes use of a combination of 2nd and 3rd order methods (hence their names).

Classic 4th Order Runge-Kutta

$y_{i+1} = y_i + \varphi\, h \qquad$ where $h = \Delta x$

$\varphi = (1/6)k_1 + (1/3)k_2 + (1/3)k_3 + (1/6)k_4$

$k_1 = f\,(x_i, y_i)$
$k_2 = f\,(x_i + (1/2)h,\; y_i + (1/2)\,k_1\,h)$
$k_3 = f\,(x_i + (1/2)h,\; y_i + (1/2)\,k_2\,h)$
$k_4 = f\,(x_i + h,\; y_i + k_3\,h)$



The comparative results below are for $\dfrac{dy}{dt} = 4e^{0.8t} - 0.5y \qquad t_0 = 0,\ y_0 = 2$

The step size is 1 second

Time	Euler	Heun	(w iteration)	Midpoint
1.000000	5.000000	6.701082	6.360865	6.217299
2.000000	11.402164	16.319782	15.302236	14.940739
3.000000	25.513212	37.199249	34.743276	33.941154
4.000000	56.849311	83.337767	77.735095	75.968632
5.000000	126.554776	185.814935	173.250143	169.340802

Time	ODE45	Analytical
1.000000	6.194622	6.194631
2.000000	14.843903	14.843922
3.000000	33.677130	33.677172
4.000000	75.338869	75.338963
5.000000	167.905943	167.905909

As might be expected, Euler does significantly worse than the other methods.

See ODEmethods.m

Second Order Systems

Suppose $A\frac{d^2y}{dx^2} + B\frac{dy}{dx} + Cy = 0$ (second order differential equation)

Let $y_1 = y, \quad y_2 = \frac{dy_1}{dx}$

Then $A\frac{dy_2}{dx} + By_2 + Cy_1 = 0$

Now we have two dependent variables (y_1 and y_2) and two simultaneous first order differential equations.

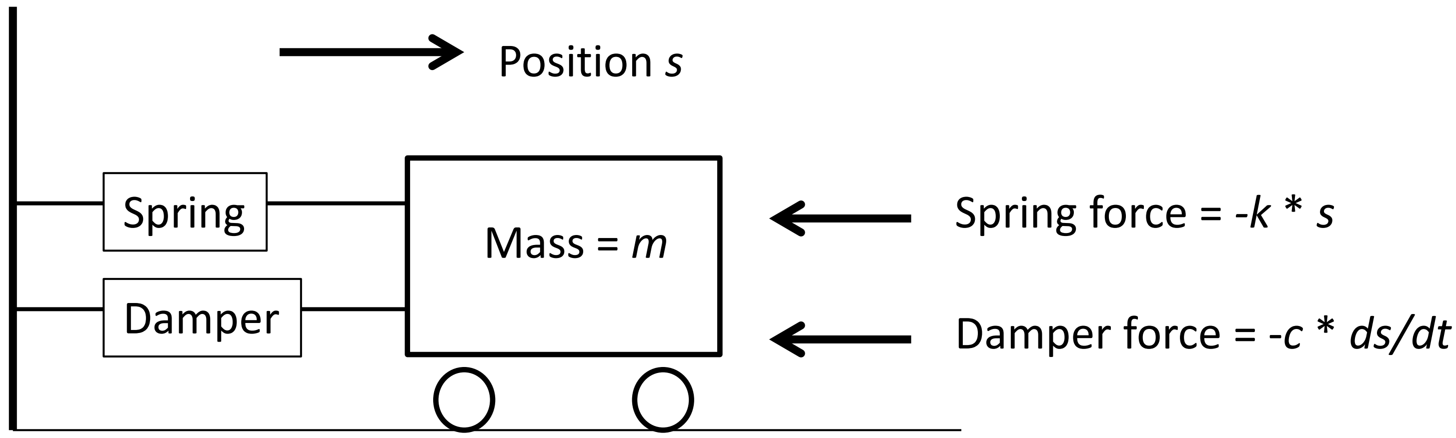
The two slope functions are

$$f_1(x, y_1, y_2) = \frac{dy_1}{dx} = y_2$$

$$f_2(x, y_1, y_2) = \frac{dy_2}{dx} = \frac{-(By_2 + Cy_1)}{A}$$

Solution requires two initial values (one for y_1 and one for y_2).

Example:



Assume: k = 2 N/m
c = 0.5 N/(m/s)
m = 1 kg
cart initially at s = 5m and moving at 10 m/s

How will the position vary over time? What is $s(t)$?

Analysis:

$F = ma \quad => \quad a = F/m$

F = spring force + damper force = $-ks - c(ds/dt)$

Therefore $a = [-ks - c(ds/dt)] / m$

Replacing a with d^2s/dt^2 gives $$\frac{d^2 s}{dt^2} = -\frac{c}{m}\frac{ds}{dt} - \frac{k}{m}s$$

Introducing velocity $v = ds/dt$ allows us to convert this second order differential equation into two simultaneous first order equations:

$$\frac{ds}{dt} = v = f_s(t, s, v) \quad \text{(slope function for } s\text{)}$$

$$\frac{dv}{dt} = -\frac{c}{m}v - \frac{k}{m}s = f_v(t, s, v) \quad \text{(slope function for } v\text{)}$$

Solution using Euler's Method

Let the step size h = 0.5 seconds

Initial Conditions (t = 0): $s_0 = 5$, $v_0 = 10$ (given in problem)

First Iteration (calculate values for $t = h$ = 0.5 seconds):

$$s_1 = s_0 + f_s(t_0, s_0, v_0) * h = 5 + (10)*0.5 = 10$$
$$v_1 = v_0 + f_v(t_0, s_0, v_0) * h = 10 + (-15)*0.5 = 2.5$$

Second Iteration (calculate values for $t = 2h$ = 1.0 seconds):

$$s_2 = s_1 + f_s(t_1, s_1, v_1) * h = 10 + (2.5)*0.5 = 11.25$$
$$v_2 = v_1 + f_v(t_1, s_1, v_1) * h = 2.5 + (-21.25)*0.5 = -8.125$$

And so on….

Solution using ode45

ode45 can handle multiple first order equations.

The slope function becomes:

```
function [ der ] = slope (t, y) % t not used
    m = 1; k = 2; c = 0.5;
    der(1,1) = y(2);
    der(2,1) = (-k/m)*y(1) - (c/m) *y(2);
end
```

Input '*y*' is a column vector containing the values of the dependent variables.

Output '*der*' is a column vector containing the slopes for the dependent variables.

The order of the variables is up to us. In our case

$y(1) = s$ der(1) = ds/dt
$y(2) = v$ der(2) = dv/dt

ode45 must be given a vector of initial values (one for each dependent variable):

```
[t, y] = ode45 (@slope, [0 20], [initialS initialV]);
```

The '*y*' returned is a matrix. The first column contains values for the first dependent variable (in our case distance *s*), the second column contains values for the second dependent variable (in our case velocity *v*), and so on.

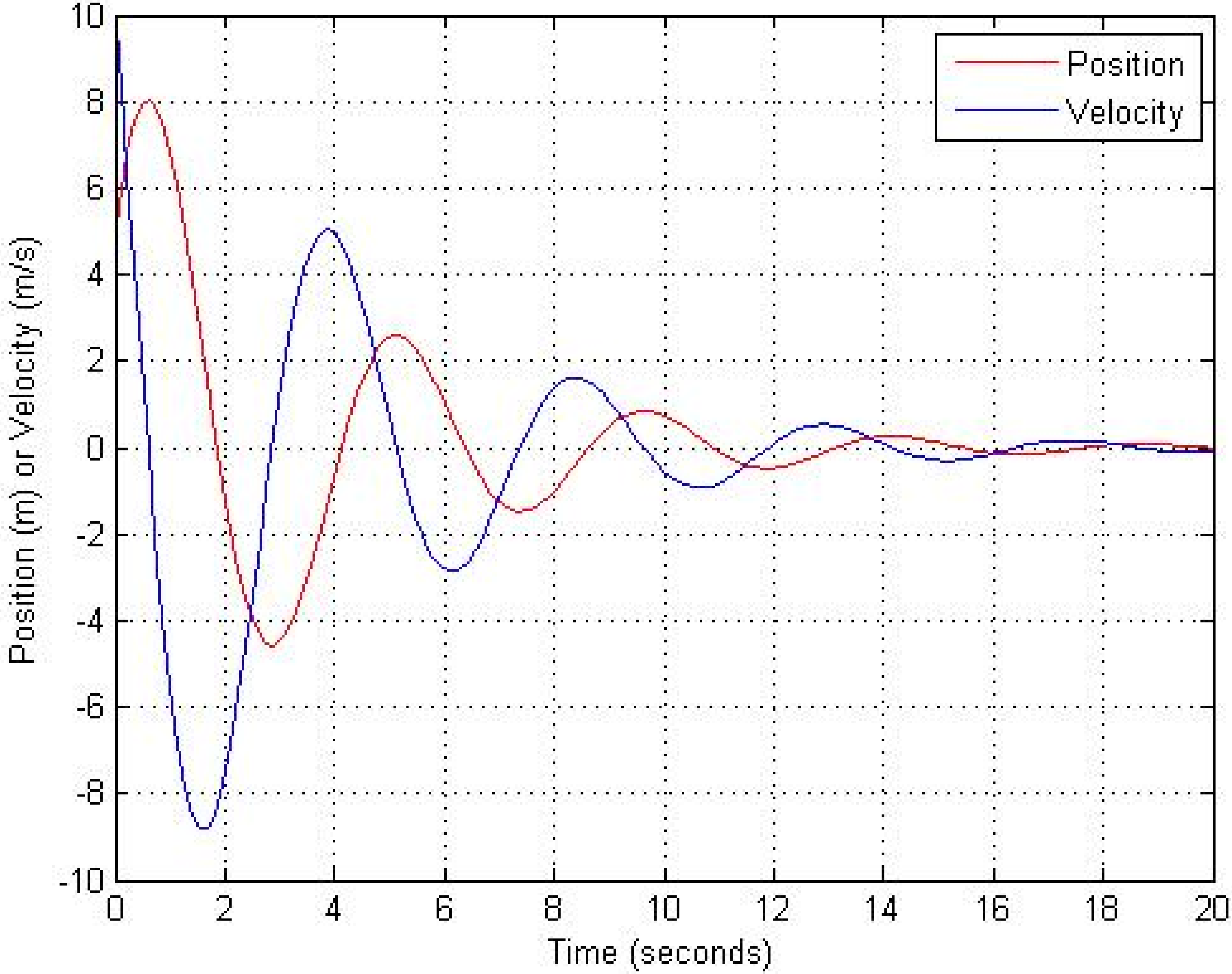
The following command plots distance and velocity against time:

```
plot (t, y(:, 1), 'r', t, y(:, 2), 'b');

% y(:, 1) gives the first column (i.e. the distance valves)
% y(:, 2) gives the second column (i.e. the velocity valves)
```

The resulting plot is shown on the next slide.

See secondOrder.m



Position
Velocity
Position (m) or Velocity (m/s)
Time (seconds)
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
2
4
6
8
10
12
14
16
18
20

Yet More on the Cart Problem (from the last set of slides)

Suppose we would like to determine the distance that the cart will travel before the braking force will bring the cart to a stop.

One possibility is to use numerical integration to integrate velocity between the starting time at the time at which the velocity becomes zero.

```
% solve until velocity zero
options = odeset ('Events', @stopper);
[t, v, eventT] = ode45 (slope, [0 tEnd], v0, options);
fprintf ('The velocity hits zero at t = %f\n', eventT(1));

% Obtain distance by integrating velocity values.
distance = trapz (t,v);
fprintf ('The distance travelling before stopping is %fm\n', distance);
```

The last pair of time and velocity values returned (t(length(t)) and v(length(v))) correspond to the point at which the velocity becomes zero. This is true even if evenly spaced output points are requested (i.e. if [0 tEnd] is replaced with 0:1:tEnd). In this case the last point becomes an exception to the rule.

It is also possible to obtain a solution by adding the distance as second dependent variable (i.e. by converting the problem into a second order problem in *s*).

The slope equations become

$$\frac{ds}{dt} = v = f_s(t, s, v) \quad \text{(slope function for } s\text{)}$$

$$\frac{dv}{dt} = \frac{-(bt + cv)}{m} = f_v(t, s, v) \quad \text{(slope function for } v\text{)}$$

In this case the required distance is simply the final distance value.

```
[t,y, eventT] = ode45 (@slope2, [0 tEnd], [v0 0], options);

% the last data point is the one we want...
n = length(t);
fprintf ('At time %f velocity is %f, distance is %f\n', t(n), y(n, 1), y(n, 2));
```

The velocity is output just as a check and should be zero.

The fact that the distance can be obtained either by integrating (*trapz*) or solving a differential equation (*ode45*) reflects a connection between the two processes.

$$\text{Assume} \quad \frac{dy}{dt} = f(t, y) \quad \text{and} \quad y_0 \quad \text{given}$$

The problem of finding *y(t)* can be addressed as either an exercise in solving a differential equation (using tools like *ode45*) or as an exercise in integration (as shown below).

$$\frac{dy}{dt} = f(t, y) \quad \Rightarrow \quad dy = f(t, y)dt \quad \Rightarrow \quad y = \int f(t, y)dt + C$$

$$y(t) = \int_{-\infty}^{t} f(t', y)dt' + C = \int_{0}^{t} f(t', y)dt' + y_0$$

In our case *s(t)* can be seen as either

$$\text{The solution to} \quad \frac{ds}{dt} = v \quad \text{given} \quad s_0 = 0$$

$$\text{The evaluation of} \int_{0}^{t} v(t')dt' + s_0$$