University of Washington
AMATH 301 Winter 2016
Instructor: Dr. King-Fai Li

Lecture Notes
Week 7
Ordinary differential equations (ODEs)

Given $\frac{dy}{dt} = f\left(t, y(t)\right)$ with $y(t = t_1) = y_1$.

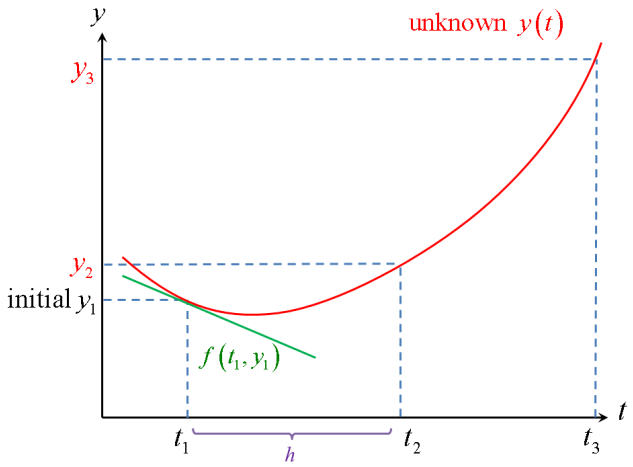
Initial Value Problem: Can we estimate $y(t_2)$ at time $t_2 = t_1 + h$, $t_3 = t_1 + 2h$, … ?

Solution: Fundamental Theorem of Calculus

$$y(t_2) = y(t_1) + \int_{t_1}^{t_2} f\left(t, y(t)\right) dt$$

f is known

But $y(t)$ is not known yet!



Find approximate solutions for $y(t)$.

The form $\frac{dy}{dt} = f\left(t, y(t)\right)$ is general

1. For a 1-D problem, e.g. $\frac{dy}{dt} = t^2 - y$, define `f=@(t,y) t.^2 -y;`

2. For a 2-D or higher dimensional problem, e.g. $\frac{dY_1}{dt} = Y_1 - Y_2 + 2$ and $\frac{dY_2}{dt} = -Y_1 + Y_2 + 4t$, rewrite the system in vector form $\frac{d}{dt}\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} Y_1 - Y_2 + 2 \\ -Y_1 + Y_2 + 4t \end{bmatrix}$, or $\frac{d\mathbf{Y}}{dt} = \mathbf{F}\left(t, \mathbf{Y}(t)\right)$. Then define

```
F=@(t,Y)[Y(1)-Y(2)+2 ; -Y(1)+Y(2)+4*t];
```

3. For high-order ODEs, e.g. $\frac{d^2y}{dt^2} + 8\frac{dy}{dt} + 2y = 0$, define $Y_1 = y$ and $Y_2 = \frac{dY_1}{dt}$, then the high-order ODEs is transformed into a 2-D problem: $\frac{dY_1}{dt} = Y_2$ and $\frac{dY_2}{dt} = -2Y_1 - 8Y_2$. Then we can rewrite them in vector form $\frac{d}{dt}\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} Y_2 \\ -2Y_1 - 8Y_2 \end{bmatrix}$ and define

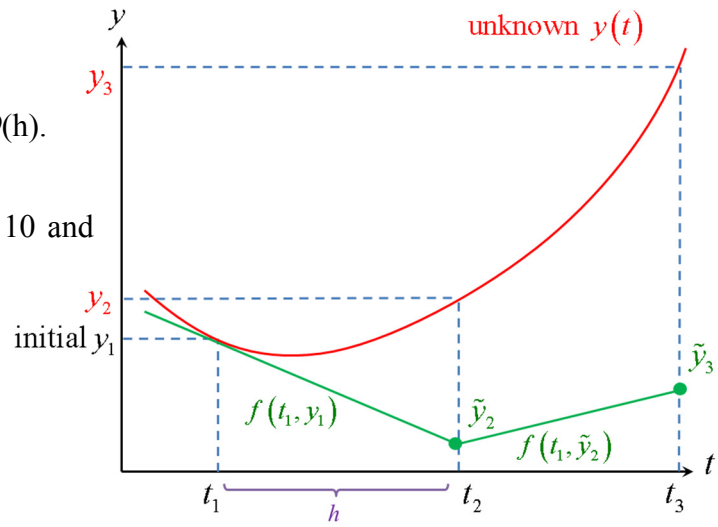
```
F=@(t,Y)[Y(2) ; -8*Y(1)-2*Y(2)];
```

Approximate Solutions

Euler's method

For small time step $t_2 = t_1 + h$, left sum applies to the integral: $\int_{t_1}^{t_2} f\left(t, y(t)\right) dt \approx f\left(t, y_1\right) h$

$$y_2 \approx \underbrace{y_1 + f\left(t_1, y_1\right) h}_{\tilde{y}_2}$$



Global errors associated with Euler's method is *O*(h).

e.g. 1-D problems: $\frac{dy}{dt} = t^2 - y$ with $y(t = 0.5) = 10$ and a time step size $h = 0.8$.
In Matlab,

```
f=@(t,y) t.^2 - y;
h=0.8;
y(1)=10;
t(1)=0.5;
y(2)=y(1)+h*f(t(1),y(1));
t(2)=t(1)+h;
```

Now we know $\tilde{y}_2$, we can then further estimate other times $t_3 = t_1 + 2h$, $t_4 = t_1 + 3h$, …

$$y_3 \approx \underbrace{\tilde{y}_2 + f\left(t_2, \tilde{y}_2\right) h}_{\tilde{y}_3}$$

$$y_4 \approx \underbrace{\tilde{y}_3 + f\left(t_2, \tilde{y}_3\right) h}_{\tilde{y}_4}$$

$$\vdots$$

In Matlab, this can easily be done by replacing the above one-step calculation with a **`for`** loop:

```
for j=1:n
      y(j+1)=y(j)+h*f(t(j),y(j));
      t(j+1)=t(j)+h;
end
```

For our example, if we are to estimate *y*(*t*=3.7), then we calculate the first 5 (i.e. **`n=4`**) values:

```
t=[ 0.50   1.30   2.10   2.90   3.70]
y=[10.00   2.20   1.79   3.89   7.51]
```

e.g. Higher-dimensional problems: $\frac{dY_1}{dt} = Y_1 - Y_2 + 2$ and $\frac{dY_2}{dt} = -Y_1 + Y_2 + 4t$

with $Y_1(t=0.4) = -1$, $Y_2(t=0.4) = 0$ and $h = 0.2$.

```
F=@(t,Y)[Y(1)-Y(2)+2 ;  -Y(1)+Y(2)+4*t];
h=0.2;
Y(:,1)=[-1;0];
t(1)=0.4;
for j=1:10
  Y(:,j+1)=Y(:,j)+h*F(t(j),Y(:,j));
  t(j+1)=t(j)+h;
end
Y1=Y(1,:)';
Y2=Y(2,:)';
```

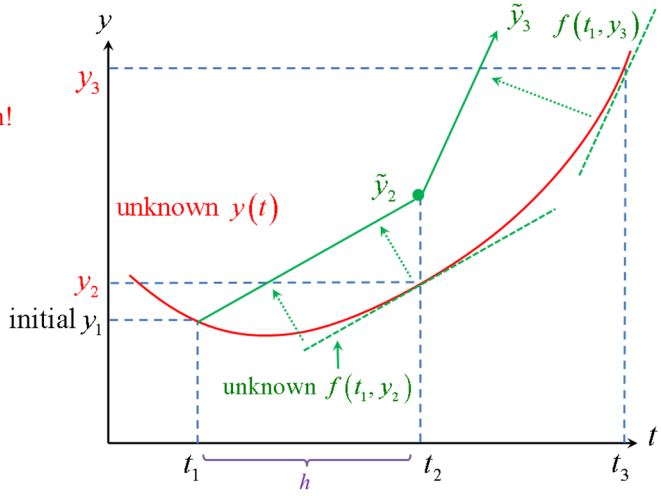
After the above calculation, `Y1(end)` = $Y_1(t=2.4)$ and `Y2(end)` = $Y_2(t=2.4)$.

Implicit Euler's method

This time applies right sum to the integral:

$$y_2 \approx y_1 + f(t_2, y_2)h$$

But y_2 is not known!



Answer: y_2 can be estimated as $\tilde{y}_2$ using `fzero`

Errors associated with Implict Euler's method is O(h).

```
f=@(t,y)  t.^2 - y;
h=0.8;
y(1)=10;
t(1)=0.5;
t(2)=t(1)+h;
y(2)=fzero(@(z)z-y(1)-h*f(t(2),z),y(1));
```

** For multi-dimensional problems, use `fsolve`.

Prof. Brunton's way:

```
y(2)=(y(1)+t(2).^2)./(1+h);
```

For other times, $\tilde{y}_3 = \tilde{y}_2 + f(t_2, \tilde{y}_3)h$, $\tilde{y}_4 = \tilde{y}_3 + f(t_3, \tilde{y}_4)h$, …

```
for j=1:n
    t(j+1)=t(j)+h;
    y(j+1)=fzero(@(z)z-y(j)-h*f(t(j+1),z),y(j));
end
```

In our example,

```
t =[ 0.50   1.30   2.10   2.90   3.70]
y =[10.00   6.31   5.46   6.77   9.85]
```

Implicit Trapezoidal Rule (optional; similar but not identical to **ode23tb**)

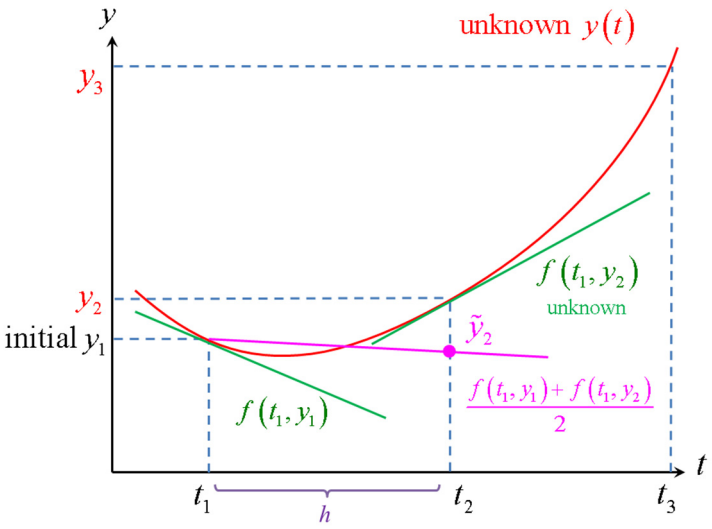
Try averaging the above two estimates.

This is the Trapezoidal Rule:

$$y_2 = y_1 + \int_{t_1}^{t_2} f(t, y(t))\,dt$$

$$\approx y_1 + \frac{h}{2}\left[f(t_1, y_1) + f(t_2, y_2)\right]$$

But still implicit. Global errors ~ $O(h^2)$.



Either:

1. Use **fzero** or **fsolve** to get y_2.
2. Use Euler's method to estimate the y_2 on the right-hand side:

$$\tilde{y}_2 = y_1 + hf(t_1, y_1)$$

$$y_2 \approx y_1 + \frac{h}{2}\left[f(t_1, y_1) + f(t_2, \tilde{y}_2)\right]$$

This is the Heun's method and is an example of the corrector-predictor methods.

4th-order Runge-Kutta (similar but not identical to `ode45`)

This time we try Simpson's rule:

$$y_2 = y_1 + \int_{t_1}^{t_2} f\left(t, y(t)\right) dt$$

$$\approx y_1 + \frac{(h/2)}{3}\left[f\left(t_1, y_1\right) + 4 f\left(t_{1.5}, y_{1.5}\right) + f\left(t_2, y_2\right)\right]$$

where $t_{1.5} = t_1 + \frac{h}{2}$. The mid-point $\left(t_{1.5}, y_{1.5}\right)$ is constructed so that the width of the interval for the Simpson rule is $h/2$. Apply the predictor-corrector method to estimate $y_{1.5}$ and y_2 on the right:

1. Define $k_1 = f\left(t_1, y_1\right)$. This is the slope at t_1.
2. Use the forward Euler method to estimate $y_{1.5}$:

$$\tilde{y}_{1.5} = y_1 + \int_{t_1}^{t_1 + \frac{1}{2}h} f\left(t, y(t)\right) dt \approx y_1 + \frac{h}{2} f\left(t_1, y_1\right) = y_1 + \frac{h}{2} k_1$$

$\tilde{y}_{1.5}$ is the first estimate of $y_{1.5}$. Define $k_2 = f\left(t_{1.5}, \tilde{y}_{1.5}\right)$.

3. Having $\tilde{y}_{1.5}$, now use backward Euler method to estimate $y_{1.5}$ again:

$$\hat{y}_{1.5} \approx y_1 + \frac{h}{2} f\left(t_{1.5}, \tilde{y}_{1.5}\right) = y_1 + \frac{h}{2} k_2$$

$\hat{y}_{1.5}$ is the 2nd estimate of $y_{1.5}$.Define $k_3 = f\left(t_{1.5}, \hat{y}_{1.5}\right)$.

4. y_2 on the left hand side can now be estimated using k_3:

$$\hat{y}_2 \approx y_1 + hf\left(t_{1.5}, \hat{y}_{1.5}\right) = y_1 + hk_3 .$$

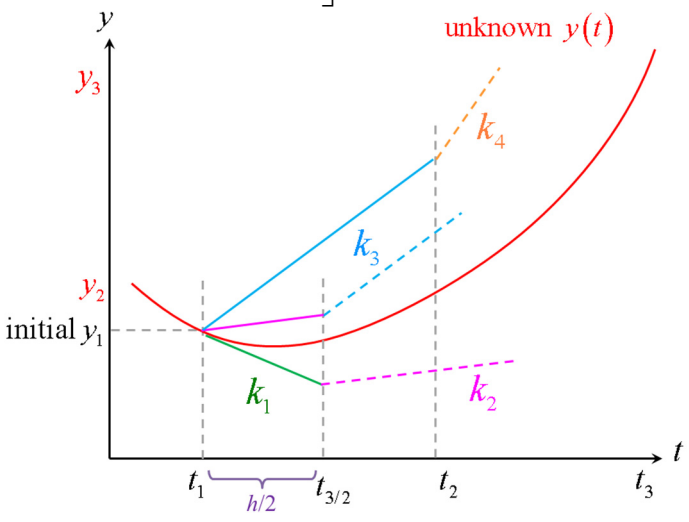
$\hat{y}_2$ is the predictor for y_2.

5. Now y_2 on the left hand side is given by substituting $\tilde{y}_{1.5}$, $\hat{y}_{1.5}$ and $\hat{y}_2$ to the right. To get a better estimate, we use both $\tilde{y}_{1.5}$ and $\hat{y}_{1.5}$ in $f\left(t_{1.5}, y_{1.5}\right)$ but we average them out:

$$y_2 \approx y_1 + \frac{(h/2)}{3}\left[f\left(t_1, y_1\right) + 4\frac{f\left(t_{1.5}, \tilde{y}_{1.5}\right) + f\left(t_{1.5}, \hat{y}_{1.5}\right)}{2} + f\left(t_2, \hat{y}_2\right)\right]$$

$$= y_1 + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

Global errors of estimate are the same as in Simpson's Rule ~ $O(h^4)$.

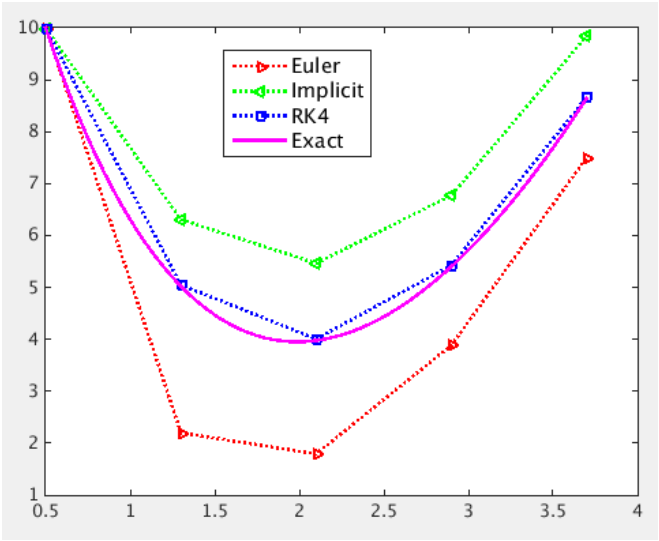


```
f=@(t,y) t.^2 - y;
h=0.8;
y(1)=10;
t(1)=0.5;
for j=1:n
  k1=f(t(j)     ,y(j)        );  % Slope at t(j)
  k2=f(t(j)+h/2,y(j)+h/2*k1);  % 1st Estimated slope at t(j)+h/2
  k3=f(t(j)+h/2,y(j)+h/2*k2);  % 2nd Estimated slope at t(j)+h/2
  k4=f(t(j)+h  ,y(j)+h  *k3);  % Estimated slope at t(j)+h
  y(j+1)=y(j)+h/6*(k1+2*k2+2*k3+k4);
  t(j+1)=t(j)+h;
end
```

(Rule: Each k is used to estimate the next k.)

In our example,

```
t =[ 0.50   1.30   2.10   2.90   3.70]
y =[10.00   5.05   4.01   5.43   8.67]
```



Exact solution in our 1-D example

$$y(t) = 8.75e^{0.5-t} + t^2 - 2t + 2$$

```
syms y(t)
yexact = dsolve(diff(y) == t.^2 - y, y(0.5) == 10)
yout = double(subs(yexact,[.5:.8:3.7]))
```

```
>> yout =[10.00   5.02   3.98   5.40   8.65]
```

Matlab's Functions

The above theories assume a fixed time step *h*. Matlab employs advanced algorithms that choose *h* automatically to minimize errors.

`ode45` — Dormand-Prince Method

4/5th-order Runge-Kutta pair

Compare the 5-th and 4-th order integration to estimate the error term and then choose *h* to reduce the error term to desired accuracy.

```
[tout,yout]=ode45(f,[t1 tmax],y1)
```

The output `tout` has points spanning between `t1` and `tmax` and the points are determined automatically. `yout` are the values of `y` at `tout`.

```
[tout,yout]=ode45(f,[t1 t2 … tmax],y1)
```

Returns `tout=[t1 t2 … tmax]`. `yout` are the values of `y` at `t1`, `t2`, `… tmax`.

In our example,

```
f=@(t,y) t.^2 - y ;
y1 = 10;
t1 = 0.5;
h = 0.8;
tmax = 3.7;
[tout,yout]=ode45(f,[t1:h:tmax],y1);
```

```
>> tout = [  0.50  1.30  2.10  2.90  3.70]
>> yout = [ 10.00  5.02  3.98  5.40  8.65]
```

(Compare this with `[tout,yout]=ode45(f,[t1 tmax],y1);`.)

`Ode23tb` — Implicit Trapezoidal-Backward Difference Method

Implicit 2/3th-order Trapezoidal pair

`[t,y]=ode23tb(f,[t1 t2],y1)`	The output `t` is determined automatically
`[t,y]=ode23tb(f,[t1 t2 … tmax],y1)`	The output `tout=[t1 t2 … tmax]`.