Overview of Topics

① Numerical Integration

(a) rectangles

(b) trapezoids

② Numerical Solutions to ODEs

(a) Forward & Backward Euler

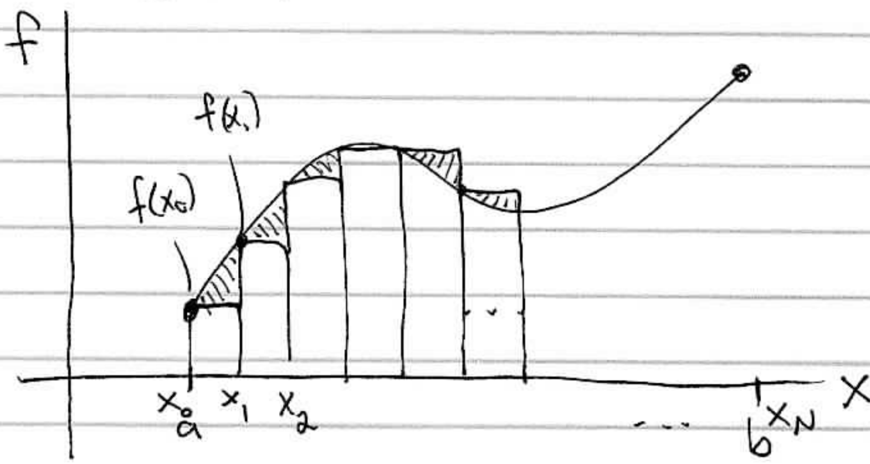
(b) Numerical Example

$$\int_a^b f(x)\,dx = \lim_{N\to\infty} \sum_{k=1}^{N} \left(f\left(a + \frac{b-a}{N}k\right)\right)\left(\frac{b-a}{N}\right)$$

$$= \lim_{\Delta x\to 0} \sum_{k=1}^{N} f(a+k\Delta x)\,\Delta x$$

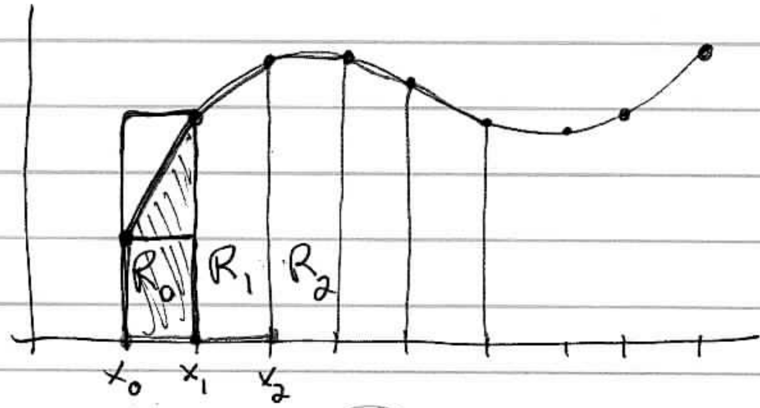
$$\boxed{= \lim_{\Delta x\to 0} \sum_{k=1}^{N} f(x_k)\,\Delta x}$$ Right-sided rectangle

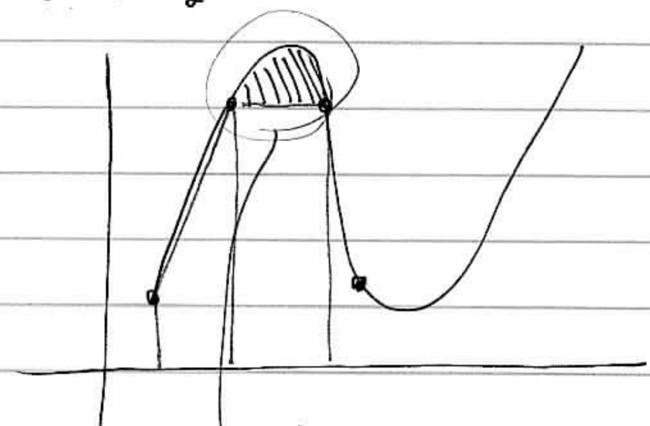
$$= \lim_{\Delta x\to 0} \sum_{k=0}^{N-1} f(x_k)\,\Delta x$$ Left-sided



N is # of rectangles

$\Delta x = (b-a)/N$ - width





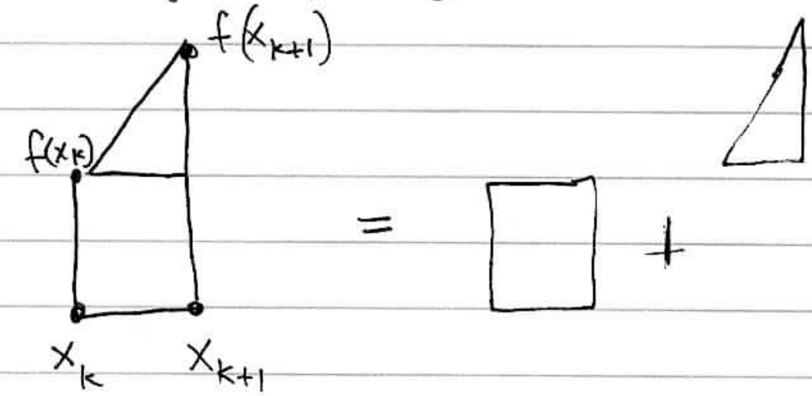
high curvature

large $f''(x)$

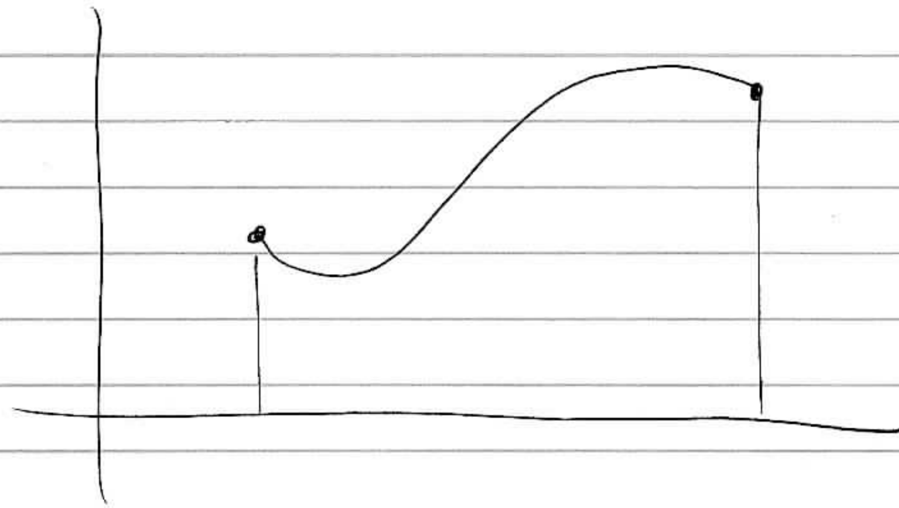
Trapezoid

$$\boxed{\int_a^b f(x)\,dx \cong \sum_{k=0}^{N-1} \frac{1}{2}\left(f(x_k)+f(x_{k+1})\right)\Delta X}$$

```
>> area = trapz(X, Y);
```

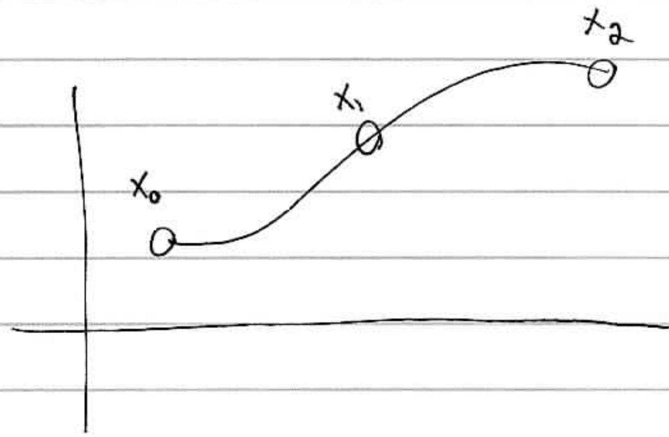


Trapz (x, y) doesn't require Δx spacing



Simpsons Rule:

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{\Delta x}{3}\left(f_0 + 4f_1 + f_2\right) - \frac{h^5}{90} f^{(4)}(c)$$



```
clear all

a = 0;
b = 10;
dxf = 0.01;
xf = a:dxf:b;
yf = sin(xf);
plot(xf,yf)

dxc = 0.5;
xc = a:dxc:b;
yc = sin(xc);
hold on
stairs(xc,yc,'r')

n = length(xc);

% left-rectangle rule
area1 = 0;
for i=1:n-1  % number of rectangles
    area1 = area1 + yc(i)*dxc;
end
area1

% right-rectangle rule
area2 = 0;
for i=1:n-1  % number of rectangles
    area2 = area2 + yc(i+1)*dxc;
end
area2

% trapezoid rule
area3 = 0;
for i=1:n-1
    area3 = area3 + (dxc/2)*(yc(i)+yc(i+1));
end
area3


% we can also use built in matlab functions
area1 = sum(yc(1:end-1))*dxc;
area2 = sum(yc(2:end))*dxc;
area3 = trapz(xc,yc);
area3 = trapz(yc)*dxc;

% we can also figure out better estimate using fine resolution data
area1f = sum(yf(1:end-1))*dxf;
area2f = sum(yf(2:end))*dxf;
area3f = trapz(xf,yf);
area3f = trapz(yf)*dxf;
```

$$\dot{x} = f(x), \quad x(0) = x_0$$

x - may be vector of states
f - may be nonlinear function.

$\dot{x} = Ax$, $x(0) = x_0$ is much simpler for matrix A.
system of first-order linear differential Eq's.

$x(t) = e^{At} x_0$ ... different class.

We are interested in numerically solving this, by starting with x_0 and iterating to get $x_0 \to x_1 \to x_2 \to \dots \to x_N$. (trajectory)

Forward Euler:

$$\frac{x_{k+1} - x_k}{\Delta t} \approx \dot{x} = f(x_k) \implies \boxed{x_{k+1} = x_k + \Delta t\, f(x_k)}$$

$$\text{If } \dot{x} = Ax \implies \boxed{x_{k+1} = (\underbrace{I}_{\text{identity matrix.}} + \Delta t A) x_k}$$

(not very stable)

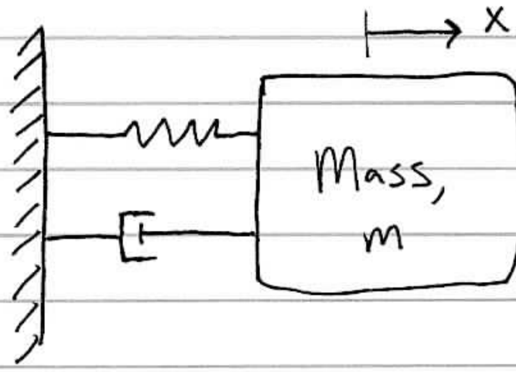
Backward (Implicit) Euler:

$$\frac{x_{k+1} - x_k}{\Delta t} \approx f(x_{k+1}) \implies x_{k+1} = x_k + \Delta t\, f(x_{k+1})$$

$$\text{if } \dot{x} = Ax \implies x_{k+1} = x_k + A \Delta t\, x_{k+1}$$

$$\implies \boxed{x_{k+1} = (I - A\Delta t)^{-1} x_k}$$ better stability.

Spring - Mass - Damper



$$m\ddot{x} = -kx - c\dot{x}$$

$$m\ddot{x} + kx + c\dot{x} = 0$$

$$\ddot{x} + \frac{k}{m}x + \frac{c}{m}\dot{x} = 0$$

If $\omega_0 = \sqrt{\frac{k}{m}}$ natural frequency

$\zeta = \frac{c}{2\sqrt{km}}$ damping ratio.

$$\boxed{\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_0^2 x = 0}$$

Second order linear differential equation.

$$\left.\begin{aligned} \dot{x} &= v \\ \dot{v} &= -2\zeta\omega_0 v - \omega_0^2 x \end{aligned}\right\} \frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -2\zeta\omega_0 \end{bmatrix}}\begin{bmatrix} x \\ v \end{bmatrix}$$

ω_0 & ζ determine eigenvalues of A, hence, the behavior of the system.

Cases: (1) Under-damped $\zeta < 1$

system oscillates w/ freq $\omega_d = \omega_0 \sqrt{1-\zeta^2}$

(2) Over-damped $\zeta > 1$

(3) Critically Damped $\zeta = 1$

Lets code up forward Euler

$$x_{k+1} = (I + A\Delta t)\, x_k$$

... try $dt = .01$ $T = 10$

... compare w/ RK4

... try $dt = 0.1$, $d = 0.5$, $d = 1$, $d = 2$.

What went wrong?..

Look at $\mathrm{eig}(I + A\Delta t)$.

Next time:

- error analysis
- global stability properties
- why is ODE45 so good?

```
clear all

w = 2*pi;
d = 1.75;  % will break for d=20

A = [0 1; -w^2 -2*d*w];

dt = .1;  % time step
T = 10;    % amount of time to integrate

x0 = [2; 0];  % initial condition

% iterate forward euler
xF(:,1) = x0;
tF(1) = 0;
for i=1:T/dt
    tF(i+1) = i*dt;
    xF(:,i+1) = (eye(2) + A*dt)*xF(:,i);
end
plot(tF,xF(1,:),'k')
hold on

% iterate backward euler
xB(:,1) = x0;
tB(1) = 0;
for i=1:T/dt
    tB(i+1) = i*dt;
    xB(:,i+1) = inv(eye(2)-A*dt)*xB(:,i);
end
plot(tB,xB(1,:),'b')

% compute better integral using build-in Matlab code
[t,y] = ode45(@(t,y) A*y, 0:dt:T,x0);
hold on
plot(t,y(:,1),'r')
```