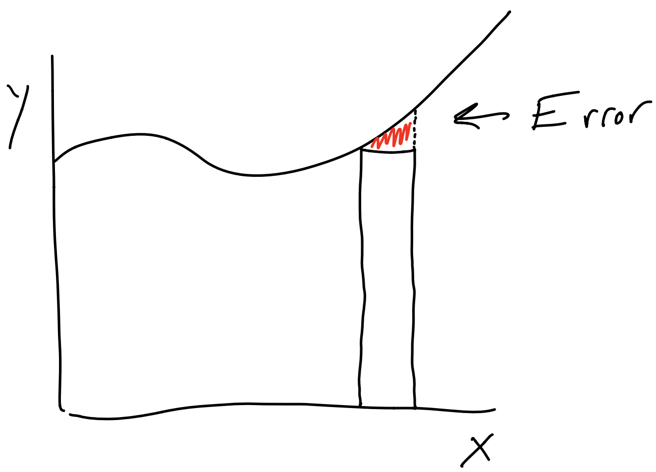
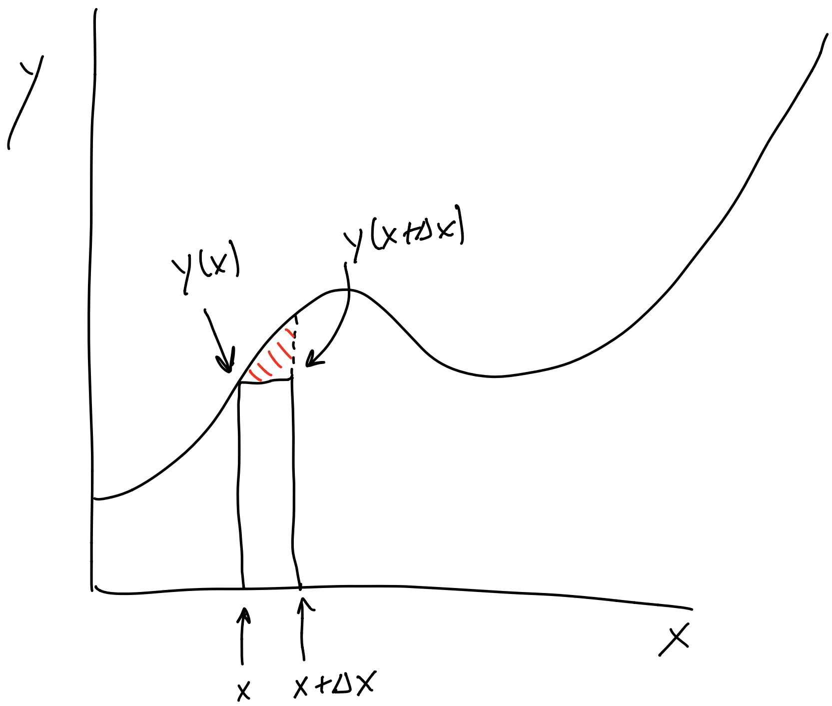
# Uncertainty

Where does our error come from?




$$A_{true} = \int_{x}^{x+\Delta x} y(x)\,dx$$

$$A_{approx} = y(x)\,\Delta x$$

$$\epsilon = A_{true} - A_{approx}$$

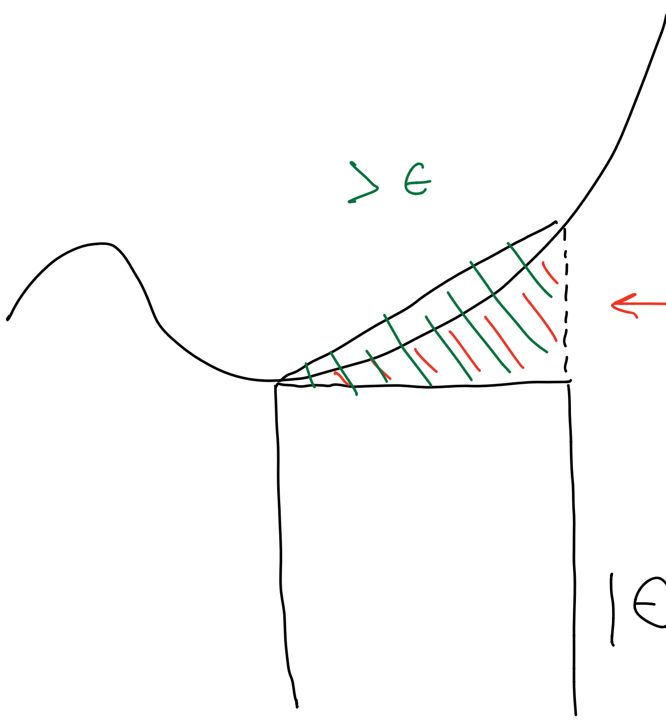
$$\epsilon = A_{true} - A_{approx}$$

$$= \int_x^{x+\Delta x} y(x)\,dx - y(x)\Delta x$$

We can always find a line w/ slope M such that

$$\left| \int_x^{x+\Delta x} Mx\,dx - y(x)\Delta x \right| \geq \epsilon$$

triangle w/ width $\Delta x$, height $M\Delta x$




$\leftarrow \epsilon$

$$|\epsilon| \leq \frac{1}{2} M \Delta x^2$$

$$|\epsilon| \leq \frac{1}{2} M \Delta x^2$$

Each rect. we deviate from the true result by a maximum of $\frac{1}{2} M \Delta x^2$,

where $M$ is some number so that

$$M \Delta x \geq \left| y(x+\Delta x) - y(x) \right|$$

$$M \geq \left| \frac{y(x+\Delta x) - y(x)}{\Delta x} \right|$$

$$M \geq | y'(x) |$$

We deviate this amount each rectangle,

there are n many rectangles

Total error

$$|E| \leq \frac{1}{2} M \Delta x^2 \cdot n$$

$$\Delta x = \frac{x_f - x_i}{n}$$

$$|E| \leq \frac{1}{2} M \frac{(x_f - x_i)^2}{n^2} \cdot n$$

$$\boxed{|E| \leq \frac{1}{2} \frac{M (x_f - x_i)^2}{n}}, \quad M = \max_{x_i \to x_f} |y'(x)|$$

$|E|$ is the <u>maximum possible error</u>

We could be closer to the true value, but we are guaranteed to be within $+/- E$

If our estimate is $A$, then the true answer <u>must</u> be:

$$A - |E| \leq A_{true} \leq A + |E|$$

```
def force(t):
    return 5000*(1-np.exp(-t**3/5-t**2))

mcar = 1100
ti = 0
tf = 8
n = 10

dt = (tf - ti) / n
total = 0
for i in range(n):
    t = ti + i * dt
    f = force(t)
    area = f * dt
    total += area

pfinal = total
vfinal = pfinal / mcar
print(vfinal)
```

```
def force(t):
    return 5000*(1-np.exp(-t**3/5-t**2))

def calc_uncertainty(xi,xf,M,n):
    """We need to find M somehow?"""
    return 0.5 * M * (xf-xi)**2 / n

mcar = 1100
ti = 0
tf = 8
n = 10

dt = (tf - ti) / n
total = 0
for i in range(n):
    t = ti + i * dt
    f = force(t)
    area = f * dt
    total += area

pfinal = total
vfinal = pfinal / mcar
print(vfinal)
```

Ex:

$$F(t) = 5000\,N \cdot \left(1 - e^{-0.2t^3 - t^2}\right)$$

$$M = \max\left(F'(t),\ 0 \leq t \leq 8\,s\right)$$

$$F'(t) = 5000\,N\left[0 - \left(-3 \cdot 0.2t^2 - 2t\right)e^{-0.2t^3 - t^2}\right]$$

$$F'(t) = 5000\,N\left(0.6t^2 + 2t\right)e^{-0.2t^3 - t^2}$$

<u>Plot</u>

Technically, we find max of $F'(t)$

by finding the critical points $(F''(t) = 0)$

- I don't feel like doing that!

```
"""Plot F'(t)"""
def force_deriv(t):
    return 5000*(0.6*t**2+2*t)*np.exp(-t**3/5-t**2)
t = np.linspace(0,8,1000)
dfdt = force_deriv(t)
plt.plot(t,dfdt)
```

```
"""Plot F'(t)"""
def force_deriv(t):
    return 5000*(0.6*t**2+2*t)*np.exp(-t**3/5-t**2)
t = np.linspace(0,8,100000)
dfdt = force_deriv(t)
plt.plot(t,dfdt)
M = dfdt.max()
plt.axhline(ftmax)
print(ftmax)
```

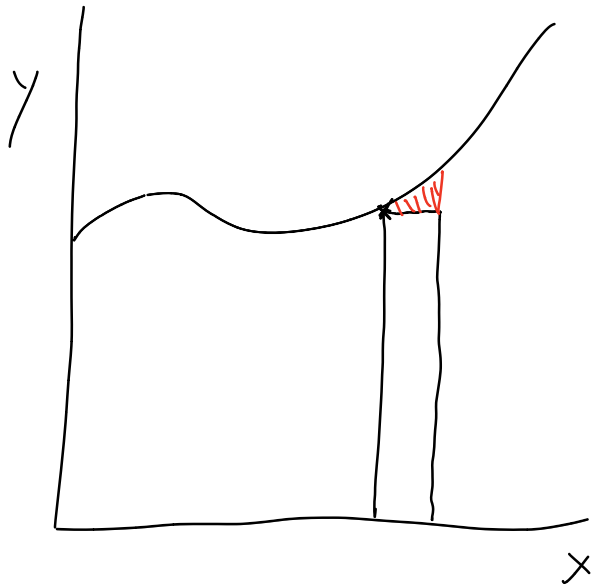
```
def force(t):
    return 5000*(1-np.exp(-t**3/5-t**2))

def force_deriv(t):
    return 5000*(0.6*t**2+2*t)*np.exp(-t**3/5-t**2)

def calc_uncertainty(xi,xf,M,n):
    """We need to find M somehow?"""
    return 0.5 * M * (xf-xi)**2 / n

mcar = 1100
ti = 0
tf = 8
n = 100
t = np.linspace(ti,tf,10000)
dfdt = force_deriv(t)
M = dfdt.max()

dt = (tf - ti) / n
total = 0
for i in range(n):
    t = ti + i * dt
    f = force(t)
    area = f * dt
    total += area

pfinal = total
pfinal_uncert = calc_uncertainty(ti,tf,M,n)
vfinal = pfinal / mcar
vfinal_uncert = pfinal_uncert / mcar
print('The speed after {} seconds is {}+-{}'.format(tf-
ti,vfinal,vfinal_uncert))
```

```
def func(x):
    pass

def deriv(x):
    pass

def calc_uncertainty(xi,xf,n):
    """We need to find M somehow?"""
    x = np.linspace(xi,xf,10000)
    dydx = deriv(x)
    M = dydx.max()
    return 0.5 * M * (xf-xi)**2 / n

def integrate(xi,xf,n):
    #First get estimate
    dx = (xf - xi) / n
    total = 0
    for i in range(n):
        x = xi + i * dx
        f = func(x)
        area = f * dx
        total += area

    #Now get uncertainty
    uncert = calc_uncertainty(xi,xf,n)
    return total,uncert
```

```python
def func(x):
    return x**2

def deriv(x):
    return 2*x

def calc_uncertainty(xi,xf,n):
    """We need to find M somehow?"""
    x = np.linspace(xi,xf,10000)
    dydx = deriv(x)
    M = dydx.max()
    return 0.5 * M * (xf-xi)**2 / n

def integrate(xi,xf,n):
    #First get estimate
    dx = (xf - xi) / n
    total = 0
    for i in range(n):
        x = xi + i * dx
        f = func(x)
        area = f * dx
        total += area

    #Now get uncertainty
    uncert = calc_uncertainty(xi,xf,n)
    return total,uncert

nvals = np.logspace(1,4,20,dtype=int) #10**np.linspace(1,4,50)
avals = []
davals = []
for n in nvals:
    a,da = integrate(0,10,n)
    avals.append(a)
    davals.append(da)
```

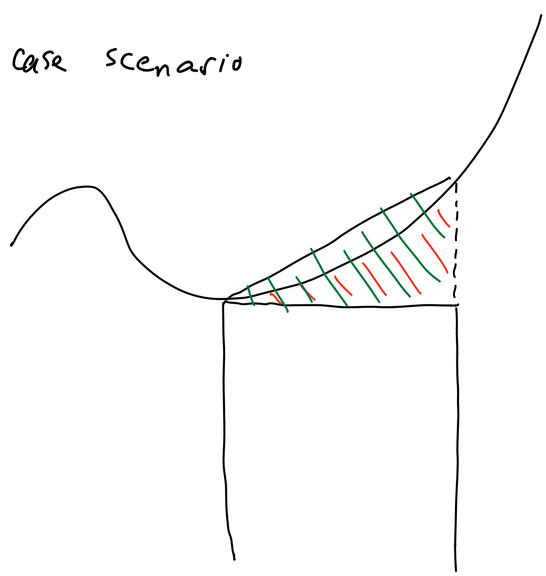
```
atrue = 1/3 * 10**3
plt.errorbar(nvals,avals,yerr=davals,fmt='o',color='blue',capsize=2)
plt.xscale('log')
plt.axhline(atrue,c='k',ls='--')
plt.xlabel('n')
plt.ylabel('A_est')
```

# Improving the approximation

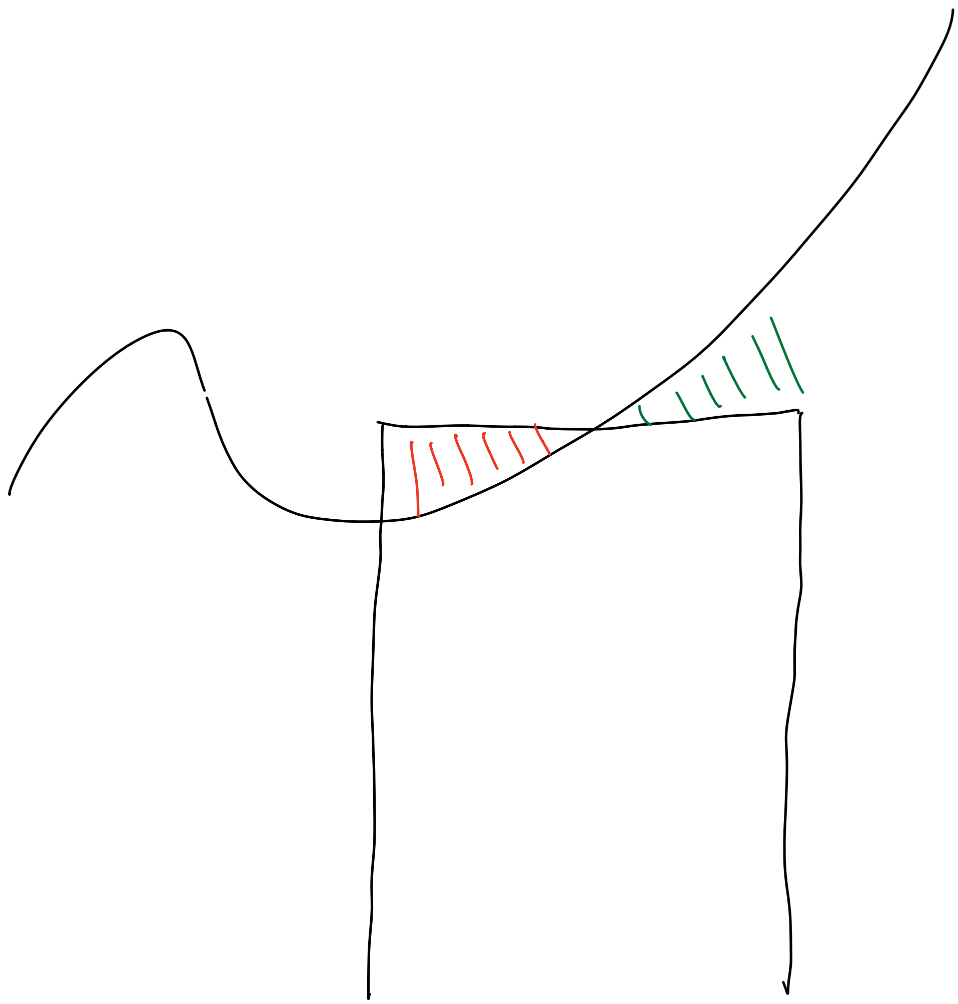



We get errors because the function deviates from our rectangle at the edge

- Worst case scenario



What if I shift each rectangle so that instead of the left edge touching the curve, the center does



$$A = y\left(x_i + \frac{\Delta x}{2}\right) \cdot \Delta x + y\left(x_i + \frac{3\Delta x}{2}\right) \cdot \Delta x + \cdots$$

$$= \sum_{k=0}^{n-1} y\left(x_i + \left(k+\tfrac{1}{2}\right)\Delta x\right) \Delta x$$

```
def integrate_midpoint(xi,xf,n):
    #First get estimate
    dx = (xf - xi) / n
    total = 0
    for i in range(n):
        x = xi + (i+1/2) * dx
        f = func(x)
        area = f * dx
        total += area
    return total
```

```python
def func(x):
    return x**2

def deriv(x):
    return 2*x

def calc_uncertainty(xi,xf,n):
    """We need to find M somehow?"""
    x = np.linspace(xi,xf,10000)
    dydx = deriv(x)
    M = dydx.max()
    return 0.5 * M * (xf-xi)**2 / n

def integrate_midpoint(xi,xf,n):
    #First get estimate
    dx = (xf - xi) / n
    total = 0
    for i in range(n):
        x = xi + (i+1/2) * dx
        f = func(x)
        area = f * dx
        total += area
    return total


def integrate(xi,xf,n):
    #First get estimate
    dx = (xf - xi) / n
    total = 0
    for i in range(n):
        x = xi + i * dx
        f = func(x)
        area = f * dx
        total += area
```

```
    #Now get uncertainty
    uncert = calc_uncertainty(xi,xf,n)
    return total,uncert

nvals = np.logspace(1,4,20,dtype=int) #10**np.linspace(1,4,50)
vals_left = []
vals_mid = []
#davals = []
for n in nvals:
    yleft,dyleft = integrate(0,10,n)
    ymid = integrate_midpoint(0,10,n)
    vals_left.append(yleft)
    vals_mid.append(ymid)

atrue = 1/3 * 10**3
plt.scatter(nvals,vals_left,c='blue')
plt.scatter(nvals,vals_mid,c='green')
plt.xscale('log')
plt.axhline(atrue,c='k',ls='--')
plt.xlabel('n')
plt.ylabel('A_est')
```