In [1]:

```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt
import xlrd
from scipy.integrate import ode
from scipy import integrate
import xlsxwriter
from IPython.display import Image
import pandas as pd
import math as ma
```

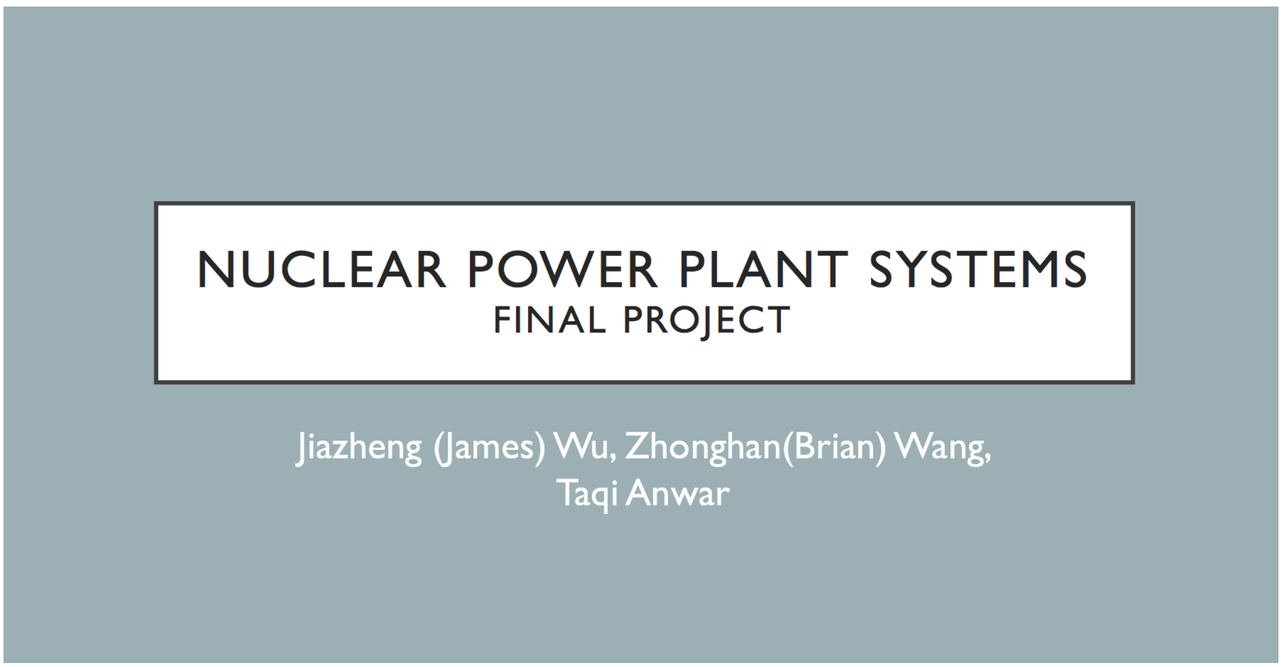
In [6]:

```
comparision='table.png'
dt='data_table.png'
hand1='handcal-1.png'
hand2='handcal-2.png'
hand3='handcal-3.png'
hand4='handcal-4.png'
vartable='vartable.png'
sen='sen.png'
title='title.png'
logic='logic.png'
thermtable='thermtable.png'
drawing='drawing.png'
flow='flow.png'
```

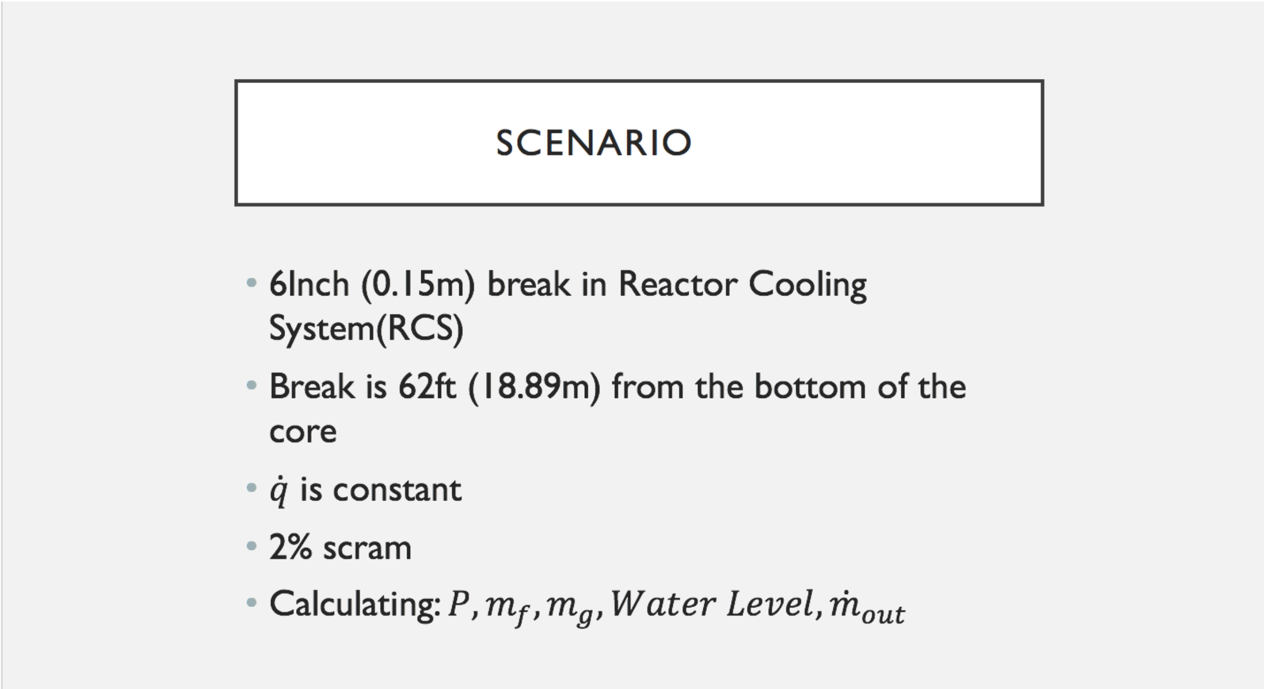
In [3]:

```
Image(title)
```

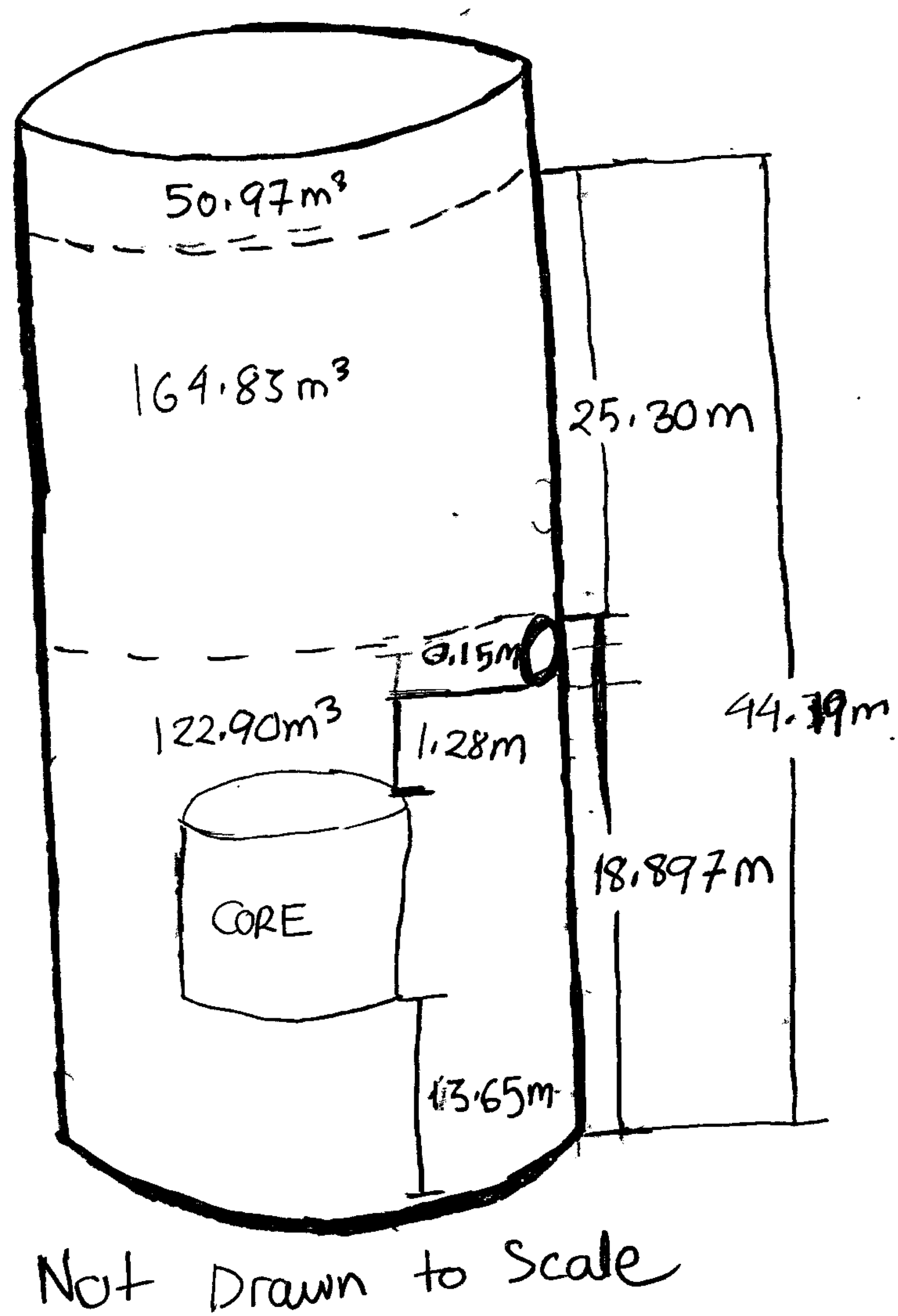
Out[3]:



In [34]:
```
Image(sen)
```
Out[34]:

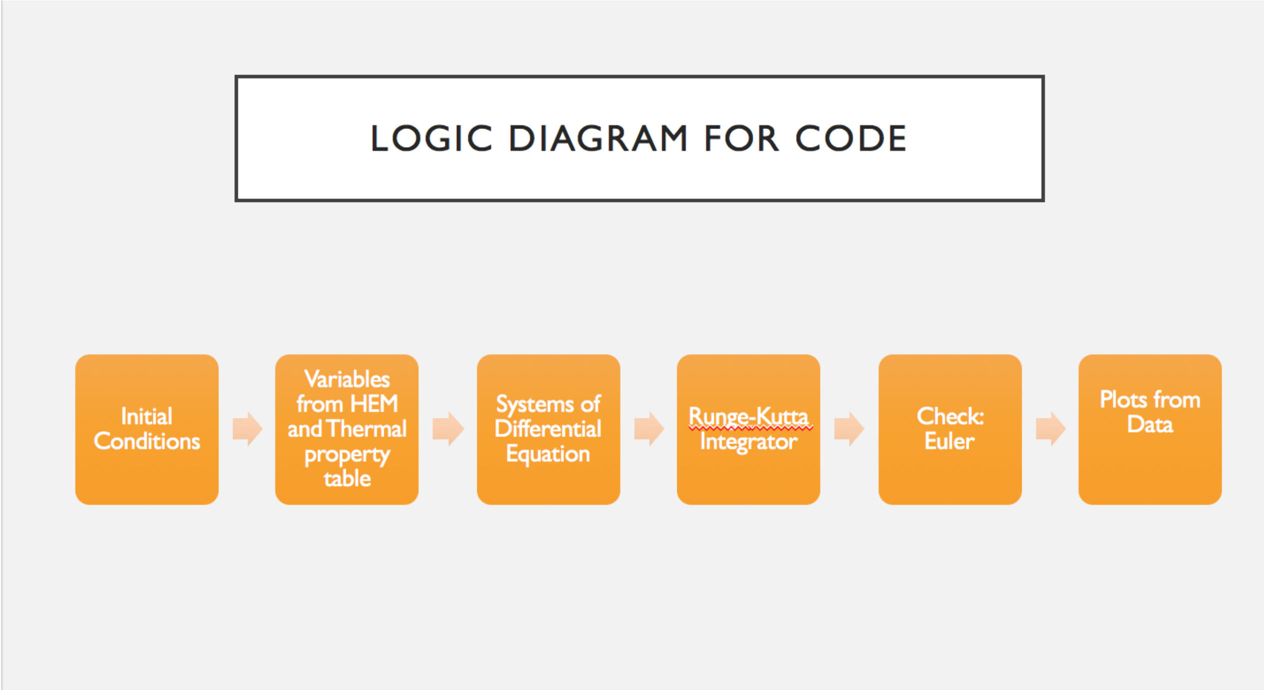


In [5]:
```
Image(drawing)
```
Out[5]:



In [35]:
```
Image(logic)
```

Out[35]:



Initial Condition

In [4]:
```
# All in units of metric

HPSI=0.02524
temp_HPSI=32.22
Break_Area=0.01824
Break_Height=18.90
Cross_Area=6.503
RCS_temp=304.4
RCS_pressure=9142448
Water_V=287.7
Steam_V=50.97
hi=137000
qdot=64e6 #Assume 2% of 3200 MW


workbook = xlrd.open_workbook("C:/Users/Zhonghan/Desktop/nuclear_power_plant/Therm_Pressure_Table.xlsx")
Thermo_data=np.zeros([73,6])
for i in np.arange(73):
    for j in np.arange(6):
        Thermo_data[i,j]=workbook.sheet_by_index(1).cell_value(i+1,j)
        if j == 0 or j == 3 or j == 4:
            Thermo_data[i,j]= Thermo_data[i,j]*1000
```

In [37]:
```
Image(thermtable)
```

Out[37]:

TABLE A–5

Saturated water—Pressure table

		Specific volume, m^3/kg		Internal energy, kJ/kg			Enthalpy, kJ/kg			Entropy, kJ/kg·K		
Press., P kPa	Sat. temp., T_{sat} °C	Sat. liquid, v_f	Sat. vapor, v_g	Sat. liquid, u_f	Evap., u_{fg}	Sat. vapor, u_g	Sat. liquid, h_f	Evap., h_{fg}	Sat. vapor, h_g	Sat. liquid, s_f	Evap., s_{fg}	Sat. vapor, s_g
1.0	6.97	0.001000	129.19	29.302	2355.2	2384.5	29.303	2484.4	2513.7	0.1059	8.8690	8.9749
1.5	13.02	0.001001	87.964	54.686	2338.1	2392.8	54.688	2470.1	2524.7	0.1956	8.6314	8.8270
2.0	17.50	0.001001	66.990	73.431	2325.5	2398.9	73.433	2459.5	2532.9	0.2606	8.4621	8.7227
2.5	21.08	0.001002	54.242	88.422	2315.4	2403.8	88.424	2451.0	2539.4	0.3118	8.3302	8.6421
3.0	24.08	0.001003	45.654	100.98	2306.9	2407.9	100.98	2443.9	2544.8	0.3543	8.2222	8.5765
4.0	28.96	0.001004	34.791	121.39	2293.1	2414.5	121.39	2432.3	2553.7	0.4224	8.0510	8.4734
5.0	32.87	0.001005	28.185	137.75	2282.1	2419.8	137.75	2423.0	2560.7	0.4762	7.9176	8.3938
7.5	40.29	0.001008	19.233	168.74	2261.1	2429.8	168.75	2405.3	2574.0	0.5763	7.6738	8.2501
10	45.81	0.001010	14.670	191.79	2245.4	2437.2	191.81	2392.1	2583.9	0.6492	7.4996	8.1488
15	53.97	0.001014	10.020	225.93	2222.1	2448.0	225.94	2372.3	2598.3	0.7549	7.2522	8.0071
20	60.06	0.001017	7.6481	251.40	2204.6	2456.0	251.42	2357.5	2608.9	0.8320	7.0752	7.9073
25	64.96	0.001020	6.2034	271.93	2190.4	2462.4	271.96	2345.5	2617.5	0.8932	6.9370	7.8302
30	69.09	0.001022	5.2287	289.24	2178.5	2467.7	289.27	2335.3	2624.6	0.9441	6.8234	7.7675
40	75.86	0.001026	3.9933	317.58	2158.8	2476.3	317.62	2318.4	2636.1	1.0261	6.6430	7.6691
50	81.32	0.001030	3.2403	340.49	2142.7	2483.2	340.54	2304.7	2645.2	1.0912	6.5019	7.5931
75	91.76	0.001037	2.2172	384.36	2111.8	2496.1	384.44	2278.0	2662.4	1.2132	6.2426	7.4558
100	99.61	0.001043	1.6941	417.40	2088.2	2505.6	417.51	2257.5	2675.0	1.3028	6.0562	7.3589
101.325	99.97	0.001043	1.6734	418.95	2087.0	2506.0	419.06	2256.5	2675.6	1.3069	6.0476	7.3545
125	105.97	0.001048	1.3750	444.23	2068.8	2513.0	444.36	2240.6	2684.9	1.3741	5.9100	7.2841
150	111.35	0.001053	1.1594	466.97	2052.3	2519.2	467.13	2226.0	2693.1	1.4337	5.7894	7.2231

Variables of P

In [6]:
```python
def vf(p=None): #returns vf, and slope at a pressure
    if p in Thermo_data[:,0]:
        a=list(Thermo_data[:,0]).index(p)
        return ((Thermo_data[a,1]),(Thermo_data[a+1,1]-Thermo_data[a-1,1])/(Thermo_data[a+1,0]-Thermo_data[a-1,0
]))
    else:
        a=list(Thermo_data[:,0] > p).index(1)
        return (((Thermo_data[a,1]-Thermo_data[a-1,1])/(Thermo_data[a,0]-Thermo_data[a-1,0]))*(p-Thermo_data[a-1,
0])
                +Thermo_data[a-1,1],(Thermo_data[a,1]-Thermo_data[a-1,1])/(Thermo_data[a,0]-Thermo_data[a-1,0]))

def vg(p=None):
    if p in Thermo_data[:,0]:
        a=list(Thermo_data[:,0]).index(p)
        return (Thermo_data[a,2],(Thermo_data[a+1,2]-Thermo_data[a-1,2])/(Thermo_data[a+1,0]-Thermo_data[a-1,0]))
    else:
        a=list(Thermo_data[:,0] > p).index(1)
        return (((Thermo_data[a,2]-Thermo_data[a-1,2])/(Thermo_data[a,0]-Thermo_data[a-1,0]))*(p-Thermo_data[a-1,
0])
                +Thermo_data[a-1,2],(Thermo_data[a,2]-Thermo_data[a-1,2])/(Thermo_data[a,0]-Thermo_data[a-1,0]))

def hf(p=None):
    if p in Thermo_data[:,0]:
        a=list(Thermo_data[:,0]).index(p)
        return (Thermo_data[a,3],(Thermo_data[a+1,3]-Thermo_data[a-1,3])/(Thermo_data[a+1,0]-Thermo_data[a-1,0]))
    else:
        a=list(Thermo_data[:,0] > p).index(1)
        return (((Thermo_data[a,3]-Thermo_data[a-1,3])/(Thermo_data[a,0]-Thermo_data[a-1,0]))*(p-Thermo_data[a-1,
0])
                +Thermo_data[a-1,3],(Thermo_data[a,3]-Thermo_data[a-1,3])/(Thermo_data[a,0]-Thermo_data[a-1,0]))

def hg(p=None):
    if p in Thermo_data[:,0]:
        a=list(Thermo_data[:,0]).index(p)
        return (Thermo_data[a,4],(Thermo_data[a+1,4]-Thermo_data[a-1,4])/(Thermo_data[a+1,0]-Thermo_data[a-1,0]))
    else:
        a=list(Thermo_data[:,0] > p).index(1)
        return (((Thermo_data[a,4]-Thermo_data[a-1,4])/(Thermo_data[a,0]-Thermo_data[a-1,0]))*(p-Thermo_data[a-1,
0])
                +Thermo_data[a-1,4],(Thermo_data[a,4]-Thermo_data[a-1,4])/(Thermo_data[a,0]-Thermo_data[a-1,0]))
def mout_f(p=None):
    return(Break_Area*8300.54*(p/1378952)**0.71)
def mout_g(p=None):
    return(Break_Area*1953.0678*(p/1378952)**1.02)
```

```
In [33]: Image(vartable)
```

Out[33]:

Variables	Values
vf	0.001422 m^3/kg
vg	0.020140 m^3/kg
d(vf)/dP	3.4 × 10−11 $m^5/N·kg$
d(vg)/dP	-2.461 × 10−9 $m^5/N·kg$
vfg	0.0187 m^3
hf	1,370,000 J/kg
hg	2,740,000 J/kg
hi	137,000 J/kg
d(hf)/dP	0.0441 m^3/kg
d(hg)/dP	-0.0174 m^3/kg
hfg	1370000
$\dot{q}$	64,000000 J/s
$\dot{w}$	0
$\dot{m}i$	17.581 kg/s
$\dot{m}o$	578.7 kg/s

Some unit conversion

$$\dot{m}_{out.f} = 1700\frac{lb}{s\cdot ft^2}\times\left(\frac{P}{200psia}\right)^{0.71} = 1700\frac{lb}{s\cdot ft^2}\left(\frac{0.4536kg}{1lb}\right)\left(\frac{1ft^2}{0.0929m^2}\right)\times\left(\frac{P}{200psia}\frac{1psia}{6894.76pa}\right)^{0.71}$$
$$= 8300.54\frac{kg}{s\cdot m^2}\times\left(\frac{P}{1,378,952pascal}\right)^{0.71}$$

In [56]:

```
print(vf(RCS_pressure)[0],vg(RCS_pressure)[0])
Image(hand1)
```

```
0.001422843232 0.020138435472
```

Out[56]:

1

$$A = \pi \left(\frac{6}{2}\right)^2 = 28.27 \text{ in}^2 = 0.196 \text{ ft}^2 = 0.0182 \text{ m}^2$$

mass flux at 1326 psia = $6200 \frac{\text{lb}}{\text{s}\cdot\text{ft}^2}$

$$\dot{m}_{out} = 0.19 \text{ ft}^2 \times \frac{6200 \text{ lb}}{\text{s}\cdot\text{ft}^2} \times \frac{253.1 \text{ kg}}{558 \text{ lb}} = 578.9 \frac{\text{kg}}{\text{s}}$$

$$p = 1326 \text{ psia} \times \frac{6.89 \text{ kPa}}{1 \text{ psia}} = 9142.4 \text{ kPa}$$

$$v_f = \left(\frac{0.001452 - 0.001418}{10{,}000 - 9{,}000}\right)(9142.4 - 9000) + 0.001418$$

$$= 0.001422 \text{ m}^3/\text{kg}$$

$$v_g = \left(\frac{0.018028 - 0.020489}{10{,}000 - 9{,}000}\right)(9142.4 - 9000) + 0.020489$$

$$= 0.020140 \text{ m}^3/\text{kg}$$

$$\frac{dv_f}{dp} = \frac{0.001452 - 0.001418}{10{,}000 - 9{,}000} = 3.4 \times 10^{-11} \frac{\text{m}^5}{\text{N}\cdot\text{kg}}$$

$$\frac{dv_g}{dp} = \frac{0.018028 - 0.020489}{10{,}000 - 9{,}000} = -2.461 \times 10^{-9} \frac{\text{m}^5}{\text{N}\cdot\text{kg}}$$

$$v_{fg} = v_g - v_f = 0.020140 - 0.001422 = 0.0187 \frac{\text{m}^3}{\text{kg}}$$

In [52]: Image(hand2)

Out[52]:

2

$$h_f = \frac{1407.8 - 1363.7}{(10{,}000 - 9{,}000)}(9142.4 - 9{,}000) + 1363.7 = 1{,}370{,}000 \ \frac{J}{kg}$$

$$h_g = \left(\frac{2725.5 - 2742.9}{10{,}000 - 9{,}000}\right)(9142.4 - 9000) + 2742.9 = 2{,}740{,}000 \ \frac{J}{kg}$$

$$h_i \ (h_f \text{ at } 32.2^\circ C) = 137{,}000 \ J/kg$$

$$\frac{dh_f}{dp} = \frac{1407.8 - 1363.7}{10{,}000 - 9{,}000} = 0.0441 \ \frac{m^3}{kg}$$

$$\frac{dh_g}{dp} = \frac{2725.8 - 2742.9}{10{,}000 - 9{,}000} = -0.0174 \ \frac{m^3}{kg}$$

$$h_{fg} = h_f - h_g = (2740.42 - 1370) \times 10^3 = 1{,}370{,}000 \ J/kg$$

$$\dot{q} = 64 \times 10^6 \ J/s, \quad \dot{W} = 0, \quad \dot{m}_r =$$

$$\dot{m}_{in} = 400 \text{ gpm} \times \left(\frac{1 \ m^3/s}{15850.32 \text{ gpm}}\right)\left(\frac{1 \ kg}{0.001422 \ m^3}\right) = 17.587 \ kg/s$$

In [7]:
```
def system_eq1( t=None,y=None): # y is a vector [pressure, water mass, steam mass]
    vfg=vg(y[0])[0]-vf(y[0])[0]
    hfg=hg(y[0])[0]-hf(y[0])[0]
    a=vfg*((hi*HPSI/vf(y[0])[0]-hf(y[0])[0]*mout_f(y[0]))+qdot)
    b=(HPSI/vf(y[0])[0]-mout_f(y[0]))*(vg(y[0])[0]*hfg-hg(y[0])[0]*vfg)
    c=vfg*(y[1]*hf(y[0])[1]+y[2]*hg(y[0])[1]-(y[1]*vf(y[0])[0]+y[2]*vg(y[0])[0]))
    d=hfg*(y[1]*vf(y[0])[1]+y[2]*vg(y[0])[1])
    return (np.array([(a+b)/(c-d),HPSI/vf(y[0])[0]-mout_f(y[0])-(qdot/(hf(y[0])[0]-hi)),qdot/(hf(y[0])[0]-hi)]))

def system_eq2( t=None, y=None):
    vfg=vg(y[0])[0]-vf(y[0])[0]
    hfg=hg(y[0])[0]-hf(y[0])[0]
    a=vfg*((hi*HPSI/vf(y[0])[0]-hg(y[0])[0]*mout_g(y[0]))+qdot)
    b=(HPSI/vf(y[0])[0]-mout_g(y[0]))*(vg(y[0])[0]*hfg-hg(y[0])[0]*vfg)
    c=vfg*(y[1]*hf(y[0])[1]+y[2]*hg(y[0])[1]-(y[1]*vf(y[0])[0]+y[2]*vg(y[0])[0]))
    d=hfg*(y[1]*vf(y[0])[1]+y[2]*vg(y[0])[1])
    return (np.array([(a+b)/(c-d),HPSI/vf(y[0])[0]-(qdot/(hg(y[0])[0]-hi)),qdot/(hg(y[0])[0]-hi)-mout_g(y[0])]))

y0=np.array([RCS_pressure, Water_V/(vf(RCS_pressure)[0]), Steam_V/(vg(RCS_pressure)[0])])
```

The problem can be model as an IVP, using system of ODE of the form:

$$\frac{\mathrm{d}}{\mathrm{dt}}\vec{y} = f(t,\vec{y}) \qquad y_0 = y(t=0)$$

We can calculate the mass of water and steam through the conservation of mass:

$$\frac{\mathrm{d}}{\mathrm{dt}}M_{Total} = \dot{M}_{In} - \dot{M}_{Out} \implies \frac{\mathrm{d}M_{Water}}{\mathrm{dt}} + \frac{\mathrm{dM_{Steam}}}{\mathrm{dt}} = \dot{M}_{HPSI} - \dot{M}_{Break}$$

We have two systems of ODE, one for when the water level is above the pipe break and the other for water level below the pipe break:

$$\frac{\mathrm{d}}{\mathrm{dt}}\begin{bmatrix} P \\ M_{Water} \\ M_{Steam} \end{bmatrix} = \begin{bmatrix} f(P, M_{Water}, M_{Steam}) \\ g(P) \\ h(P) \end{bmatrix} = \begin{bmatrix} \dfrac{v_{fg}\left(\sum(\dot{m}h)_j + \dot{q} - \dot{w}_s\right) + \sum \dot{m}_j(v_g h_{fg} - h_g v_{fg})}{v_{fg}\left(m_f\dfrac{\mathrm{d}h_f}{\mathrm{d}P} + m_g\dfrac{\mathrm{d}h_g}{\mathrm{d}P} - V\right) - h_{fg}\left(m_f\dfrac{\mathrm{d}v_f}{\mathrm{d}P} + m_g\dfrac{\mathrm{d}v_g}{\mathrm{d}P}\right)} \\ \dot{M}_{HPSI} - \dot{M}_{Water.Out}(P) - \dfrac{\dot{q}}{h_f(P) - h_{In}} \\ \dfrac{\dot{q}}{h_f(P) - h_{In}} \end{bmatrix} \tag{1}$$

$$\frac{\mathrm{d}}{\mathrm{dt}}\begin{bmatrix} P \\ M_{Water} \\ M_{Steam} \end{bmatrix} = \begin{bmatrix} f(P, M_{Water}, M_{Steam}) \\ g(P) \\ h(P) \end{bmatrix} = \begin{bmatrix} \dfrac{v_{fg}\left(\sum(\dot{m}h)_j + \dot{q} - \dot{w}_s\right) + \sum \dot{m}_j(v_g h_{fg} - h_g v_{fg})}{v_{fg}\left(m_f\dfrac{\mathrm{d}h_f}{\mathrm{d}P} + m_g\dfrac{\mathrm{d}h_g}{\mathrm{d}P} - V\right) - h_{fg}\left(m_f\dfrac{\mathrm{d}v_f}{\mathrm{d}P} + m_g\dfrac{\mathrm{d}v_g}{\mathrm{d}P}\right)} \\ \dot{M}_{HPSI} - \dfrac{\dot{q}}{h_g(P) - h_{In}} \\ -\dot{M}_{Steam.out}(P) + \dfrac{\dot{q}}{h_g(P) - h_{In}} \end{bmatrix} \tag{2}$$

```
In [53]: Image(hand3)
```

Out[53]:

③

$$\boxed{\Sigma(\dot{m}h) = \left[(17.581)(137) - (578.9)(1370)\right] \times 10^3 = -7.9 \times 10^8 \tfrac{J}{s}}$$

$$\boxed{\Sigma(\dot{m})_j = (17.581 - 578.9) = -561.32 \ kg/s}$$

$$m_f = \frac{287.7}{0.001422} = 202,320.68 \ kg, \quad m_g = \frac{51}{0.020140} = 2523.3 \ kg$$

$$V_f = 10160 \ ft^3 \times \frac{0.0283 \ m^3}{1 \ ft^3} = 287.7 \ m^3$$

$$V_g = 1800 \ ft^3 \times \frac{0.0283 \ m^3}{1 \ ft^3} = 51 \ m^3$$

$$\frac{dP}{dt} = \frac{(0.0187)(-7.9\times10^8 + 64\times10^6) - (561.32)(0.02014 \cdot 1.37\times10^6 - 2.74\times10^6 \cdot 0.0187)}{\left[(0.0187)\left[202320.68(0.0441) - 2532.3(0.0174) - 338.7\right] - (1.37\times10^6)(202320.68)(3.4\times10^{-11}) + 2532.3(-2.461\times10^{-9}) \right]}$$

$$= \frac{-13576200 - (561.32)(-23646.2)}{(158.8)}$$

$$\boxed{\frac{dP}{dt} = -1908.78 \ \frac{pa}{s} = -1.9 \ \frac{kpa}{s}}$$

In [57]:
```
print(system_eq1(0,y0))
Image(hand4)
```

```
[-1938.15185994  -614.13772396    51.9066801 ]
```

Out[57]:

4

$P_i = 9142.4$ kpa $\quad M_{fi} = 202320.68$ kg $\quad M_{gi} = 2532.3$ kg

using conservation of mass: $\frac{dM_g}{dt}$

$$\dot{m}_{in} - \dot{m}_{out} = \frac{dM_f}{dt} + \frac{\dot{q}}{(h_{out} - h_{in})}$$

$$\boxed{\frac{dM_f}{dt} = \dot{m}_{in} - \dot{m}_{out} - \frac{\dot{q}}{(h_o - h_i)} = (17.581 - 578.7) - \frac{64\times10^6}{(1370000 - 137000)} = -613.019\,\text{kg/s}}$$

$$\boxed{\frac{dM_g}{dt} = \frac{\dot{q}}{(h_o - h_i)} = 51.9\,\text{kg/s}.}$$

$\Delta t = 0.1$

1st interation:

$$P_1 = P_i + \frac{dP}{dt}\Delta t = 9142.4\,\text{kpa} + (-1.9)(0.1) = 9142.21\,\text{kpa}$$

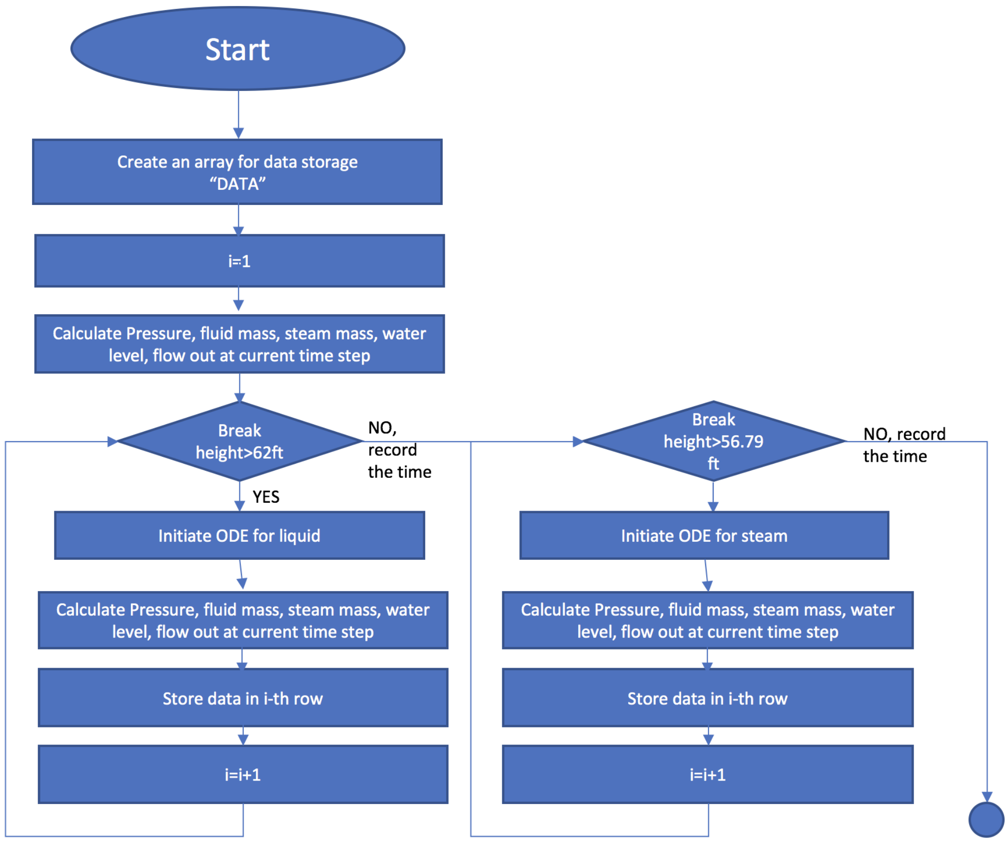
$$M_{f1} = M_{fi} + \frac{dM_f}{dt} = 202320.68 + (-613.019)(0.1) = 202259.37\,\text{kg}$$

$$M_{g1} = M_{gi} + \frac{dM_g}{dt} = 2532.3 + (51.9)(0.1) = 2537.49\,\text{kg}$$

Runge Kutta Integrator Method

In [7]:
```
Image(flow)
```

Out[7]:



In [8]:
```
Solver = ode(system_eq1).set_integrator("dopri5")
Solver.set_initial_value(y0,0)
t=0.1
i=1
val=Solver.integrate(t)

DATA1=np.append(np.append(0,y0),[Water_V/Cross_Area,mout_f(RCS_pressure)])
# time, pressure, fluid mass, steam mass, water level, flow out
DATA1=np.vstack([DATA1,np.append(np.append(t,val),[(val[1]*vf(val[0])[0])/Cross_Area,mout_f(val[0])])])

while DATA1[i,4] >  Break_Height: #62 ft
    t += 0.1
    i += 1
    val=Solver.integrate(t)
    DATA1=np.vstack([DATA1,np.append(np.append(t,val),[(val[1]*vf(val[0])[0])/Cross_Area,mout_f(val[0])])])
```

In [9]:
```
t_crit=t
i_crit=i

Solver2=ode(system_eq2).set_integrator("dopri5")
Solver2.set_initial_value(DATA1[i][1:4],t)
while DATA1[i,4] >  17.31: #56.79 ft
    t += 0.1
    i += 1
    val=Solver2.integrate(t)
    DATA1=np.vstack([DATA1,np.append(np.append(t,val),[(val[1]*vf(val[0])[0])/Cross_Area,mout_f(val[0])])])
```

In [59]:
```
print(t_crit,t)
```

```
189.8999999999935 264.2999999999916
```

Water level drops below the pipe break **189.9** seconds after the break, and core begin uncover at **264.3** second.

Euler Method

$$\vec{y}(t+\Delta t) = \vec{y}(t) + f(t, \vec{y}) \cdot \Delta t$$

Time Step = 10 sec

In [10]:
```
def Euler_Method(Table=None, dt=None):
    j=1

    Table=np.append(np.append(0,y0),[Water_V/Cross_Area,mout_f(RCS_pressure)])
    # time, pressure, fluid mass, steam mass, water level, flow out
    val2 = Table[1:4]+system_eq1(0,Table[1:4])*dt
    layer= np.append(np.append(j*dt,val2),[(val2[1]*vf(val2[0])[0])/Cross_Area,mout_f(val2[0])])
    Table=np.vstack([Table,layer])

    while Table[j,4] >  Break_Height: #62 ft
        j += 1
        val2 = Table[j-1][1:4]+system_eq1(0,Table[j-1][1:4])*dt
        layer= np.append(np.append(j*dt,val2),[(val2[1]*vf(val2[0])[0])/Cross_Area,mout_f(val2[0])])
        Table=np.vstack([Table,layer])

    j_crit=j

    while Table[j,4] >   17.31: #56.79 ft
        j += 1
        val2 = Table[j-1][1:4]+system_eq2(0,Table[j-1][1:4])*dt
        layer= np.append(np.append(j*dt,val2),[(val2[1]*vf(val2[0])[0])/Cross_Area,mout_f(val2[0])])
        Table=np.vstack([Table,layer])
    return(j_crit*dt, Table)
```

In [12]:
```
DATA2=np.append(np.append(0,y0),[Water_V/Cross_Area,mout_f(RCS_pressure)])
DATA2=Euler_Method(DATA2,10)
```

Time Step = 1 sec

In [13]:
```
DATA3=np.append(np.append(0,y0),[Water_V/Cross_Area,mout_f(RCS_pressure)])
DATA3=Euler_Method(DATA3,1)
```

Time Step = 0.1sec

In [14]:
```
DATA4=np.append(np.append(0,y0),[Water_V/Cross_Area,mout_f(RCS_pressure)])
DATA4=Euler_Method(DATA4,0.1)
```

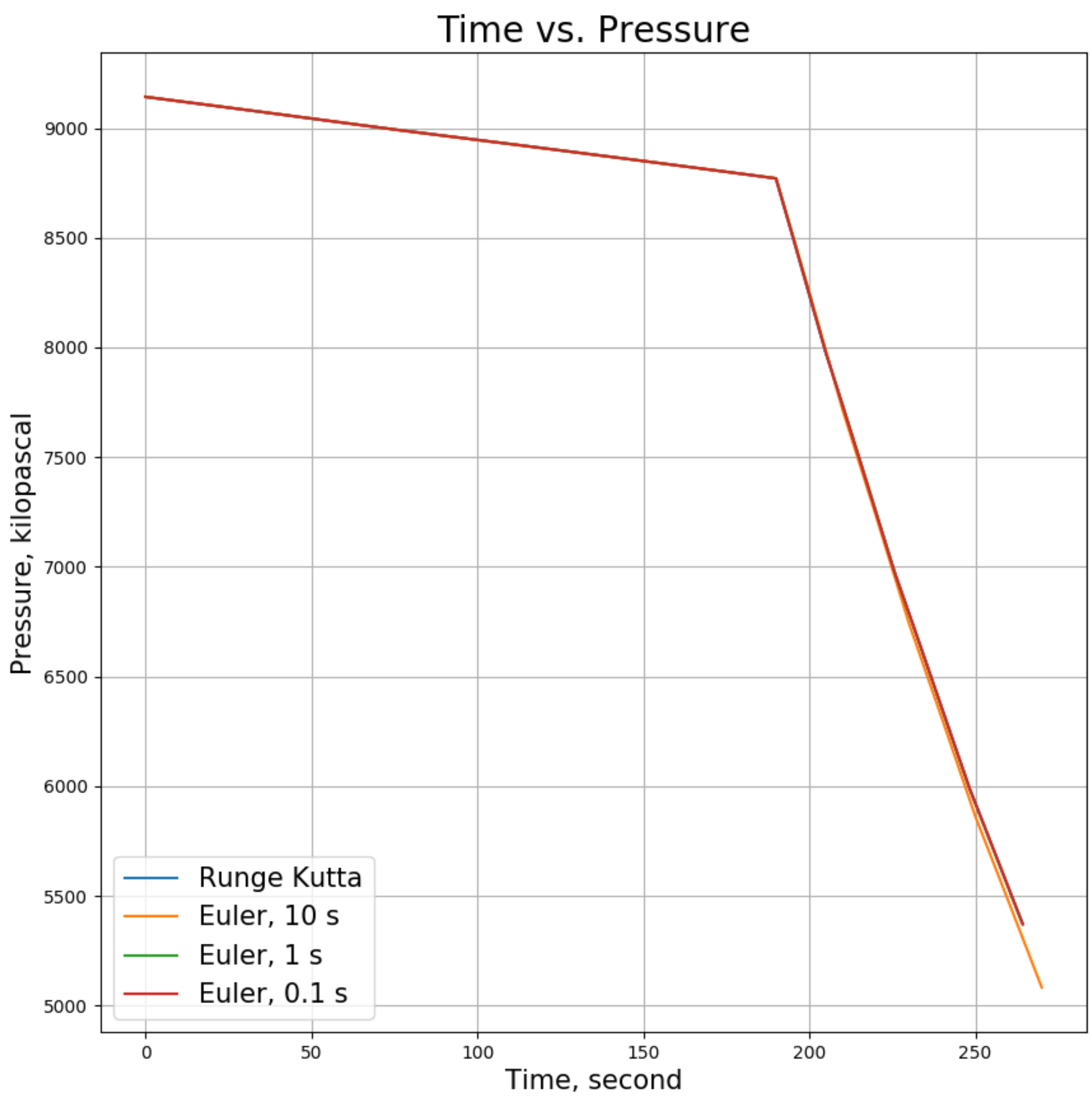
Results

In [15]:
```
head=np.array(['Runge-Kutta, Kpa','Euler-10s, Kpa','Euler-1s, Kpa','Euler-0.1s, Kpa'])
compare=pd.DataFrame(np.c_[DATA1[:,1][::100]/1000,DATA2[1][:,1][:-1]/1000,DATA3[1][:,1][::10]/1000,DATA4[1][:,1]
[::100]/1000],
            index=DATA1[:,0][::100],
            columns=head)
compare.style
```

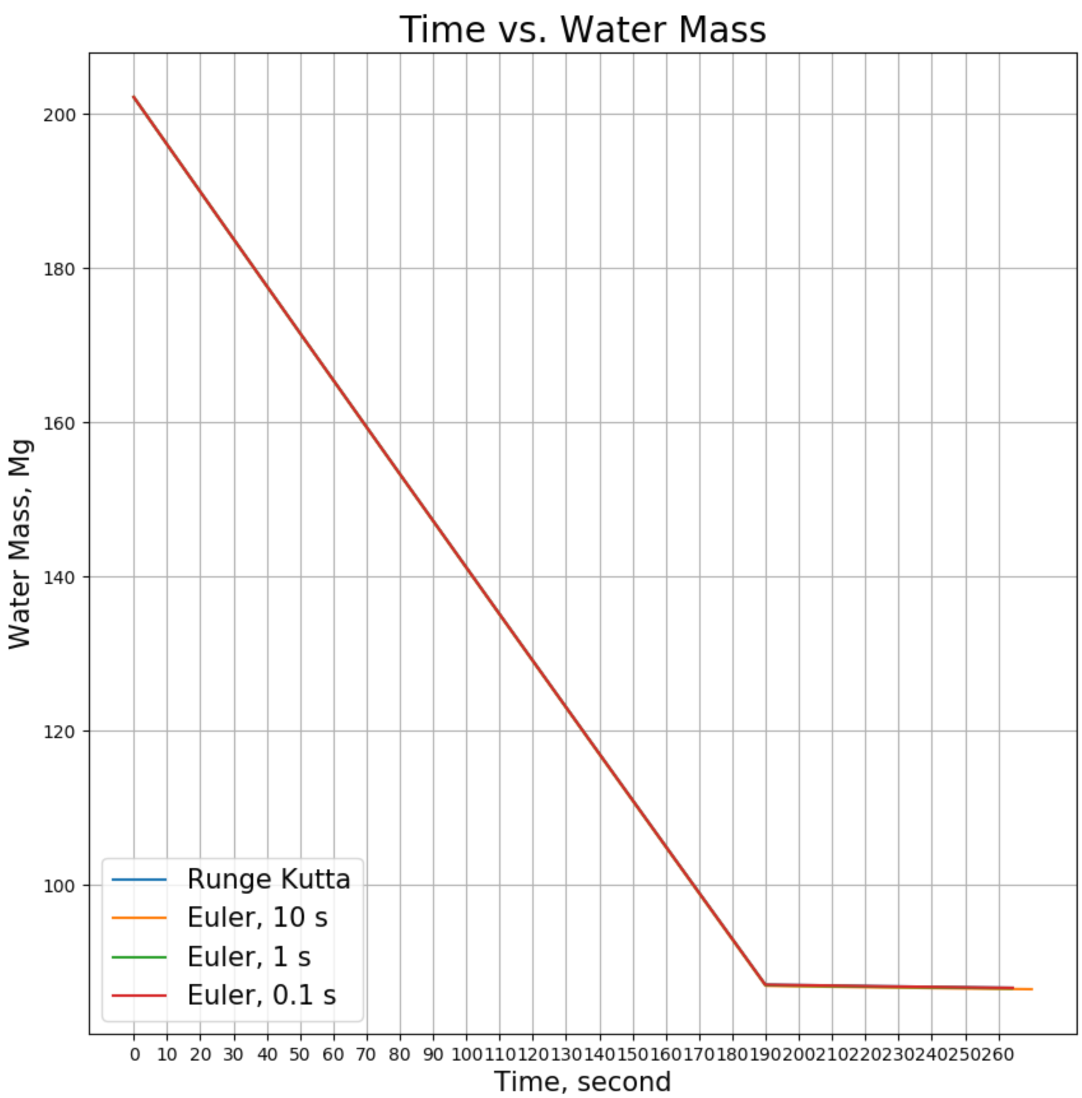
Out[15]:

	Runge-Kutta, Kpa	Euler-10s, Kpa	Euler-1s, Kpa	Euler-0.1s, Kpa
0.0	9142.45	9142.45	9142.45	9142.45
10.0	9123	9123.07	9123.01	9123
20.0	9103.42	9103.55	9103.43	9103.42
30.0	9083.7	9083.9	9083.72	9083.7
40.0	9063.83	9064.1	9063.86	9063.83
50.0	9043.81	9044.15	9043.85	9043.81
60.0	9023.64	9024.05	9023.68	9023.64
70.0	9003.3	9003.78	9003.35	9003.3
80.0	8984.26	8983.35	8984.25	8984.25
90.0	8965.41	8964.56	8965.4	8965.4
100.0	8946.47	8945.67	8946.46	8946.46
110.0	8927.42	8926.68	8927.42	8927.41
120.0	8908.27	8907.59	8908.27	8908.26
130.0	8889.01	8888.39	8889.02	8889
140.0	8869.63	8869.07	8869.64	8869.62
150.0	8850.12	8849.63	8850.14	8850.12
160.0	8830.49	8830.06	8830.51	8830.48
170.0	8810.72	8810.36	8810.75	8810.71
180.0	8790.81	8790.51	8790.84	8790.8
190.0	8765.77	8770.52	8770.78	8765.76
200.0	8238.8	8252.53	8244.61	8238.87
210.0	7736.5	7715.68	7738.97	7736.3
220.0	7251.7	7235.29	7254.46	7251.53
230.0	6790.13	6742.32	6787.08	6789.26
240.0	6352.72	6305.86	6349.75	6351.87
250.0	5923.68	5862.31	5920.49	5922.88
260.0	5537.33	5473.89	5533.88	5536.52

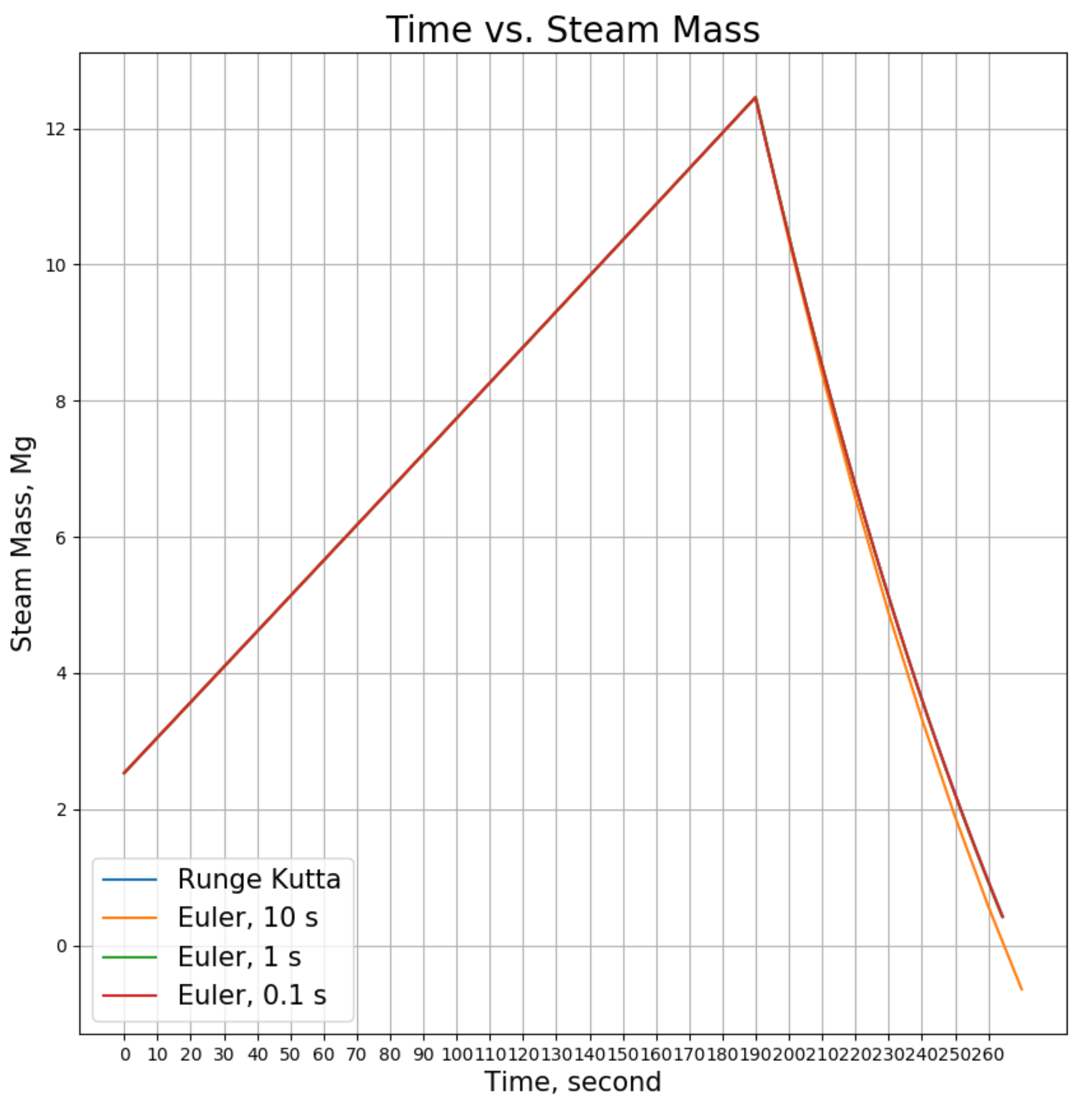
In [70]:
```
plt.figure(figsize=(10,10))
plt.title(r"Time vs. Pressure", fontsize=20)
plt.xlabel(r"Time, second", fontsize=15)
plt.ylabel(r"Pressure, kilopascal", fontsize=15)
plt.plot(DATA1[:,0], DATA1[:,1]/1000, label="Runge Kutta")
plt.plot(DATA2[1][:,0], DATA2[1][:,1]/1000, label="Euler, 10 s")
plt.plot(DATA3[1][:,0], DATA3[1][:,1]/1000, label="Euler, 1 s")
plt.plot(DATA4[1][:,0], DATA4[1][:,1]/1000, label="Euler, 0.1 s")
plt.grid("True")
plt.legend(loc=3,fontsize=15)
#plt.xticks(np.arange(min(DATA1[:,0]), max(DATA1[:,0])+1, 10))
plt.show()
```



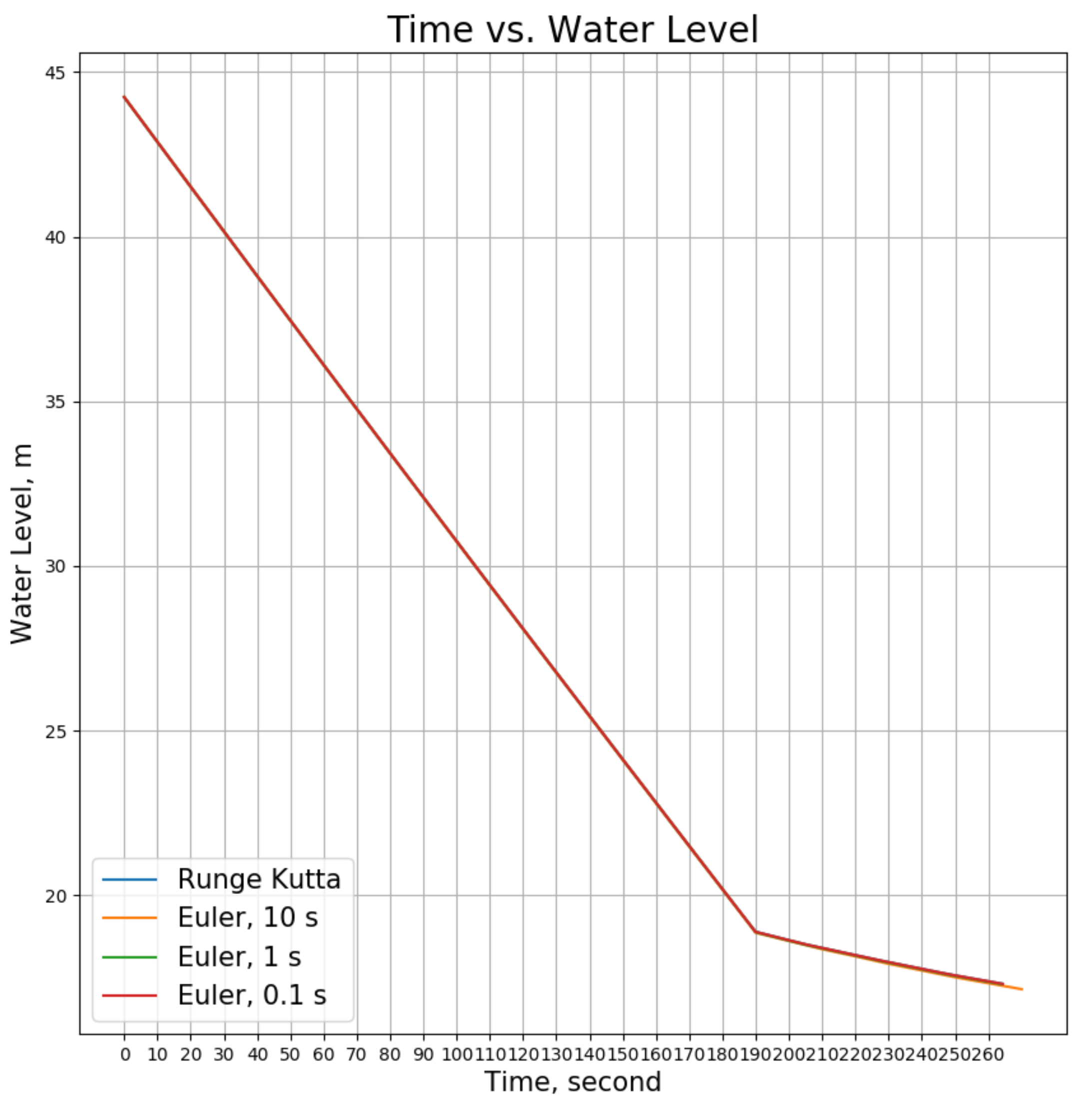
In [69]:
```
plt.figure(figsize=(10,10))
plt.title(r"Time vs. Water Mass", fontsize=20)
plt.xlabel(r"Time, second", fontsize=15)
plt.ylabel(r"Water Mass, Mg", fontsize=15)
plt.plot(DATA1[:,0], DATA1[:,2]/1000, label="Runge Kutta")
plt.plot(DATA2[1][:,0], DATA2[1][:,2]/1000, label="Euler, 10 s")
plt.plot(DATA3[1][:,0], DATA3[1][:,2]/1000, label="Euler, 1 s")
plt.plot(DATA4[1][:,0], DATA4[1][:,2]/1000, label="Euler, 0.1 s")
plt.grid("True")
plt.legend(loc=3,fontsize=15)
plt.xticks(np.arange(min(DATA1[:,0]), max(DATA1[:,0])+1, 10))
plt.show()
```



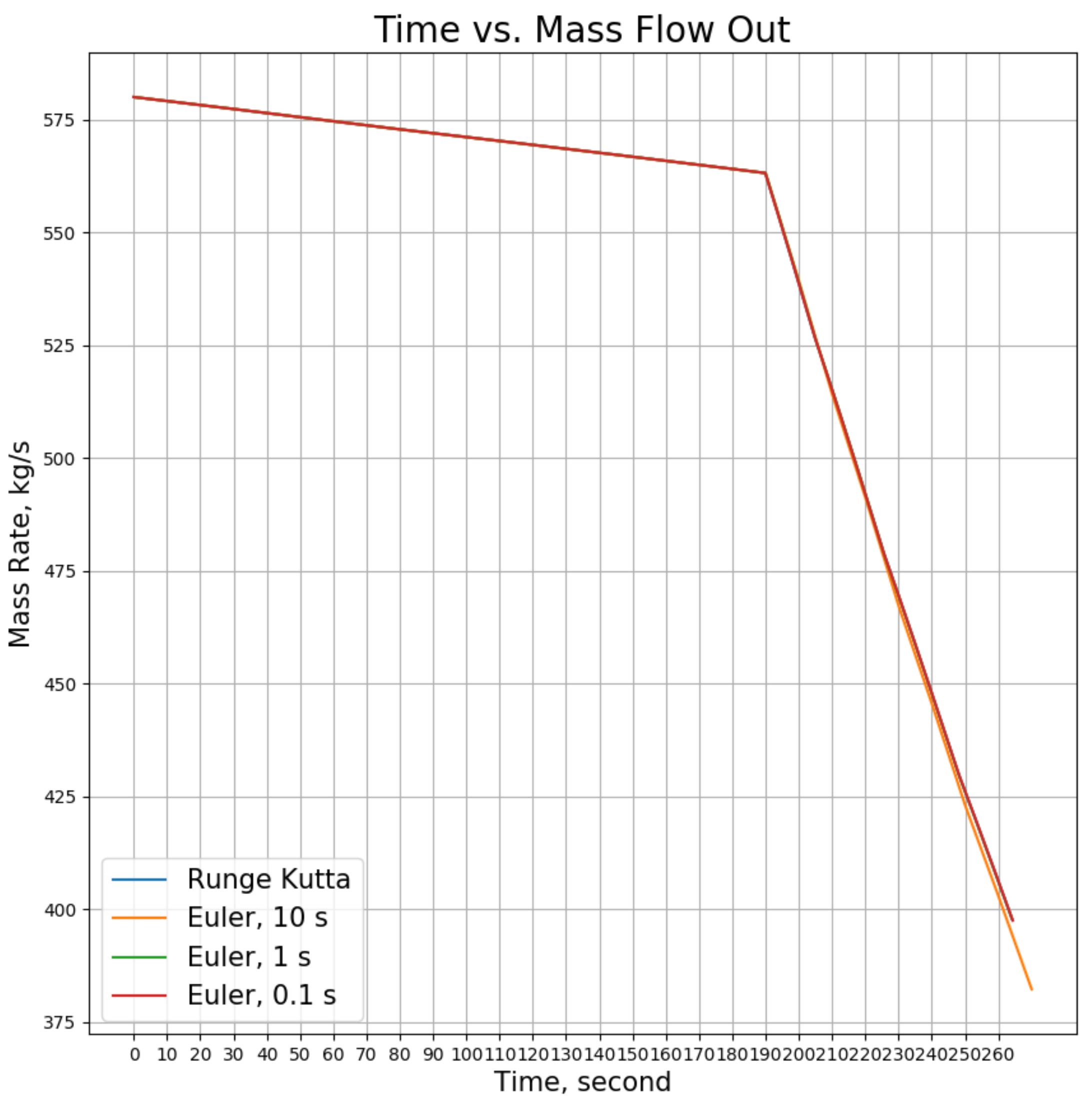
In [65]:
```
plt.figure(figsize=(10,10))
plt.title(r"Time vs. Steam Mass", fontsize=20)
plt.xlabel(r"Time, second", fontsize=15)
plt.ylabel(r"Steam Mass, Mg", fontsize=15)
plt.plot(DATA1[:,0], DATA1[:,3]/1000, label="Runge Kutta")
plt.plot(DATA2[1][:,0], DATA2[1][:,3]/1000, label="Euler, 10 s")
plt.plot(DATA3[1][:,0], DATA3[1][:,3]/1000, label="Euler, 1 s")
plt.plot(DATA4[1][:,0], DATA4[1][:,3]/1000, label="Euler, 0.1 s")
plt.grid("True")
plt.legend(loc=3,fontsize=15)
plt.xticks(np.arange(min(DATA1[:,0]), max(DATA1[:,0])+1, 10))
plt.show()
```



In [66]:
```
plt.figure(figsize=(10,10))
plt.title(r"Time vs. Water Level", fontsize=20)
plt.xlabel(r"Time, second", fontsize=15)
plt.ylabel(r"Water Level, m", fontsize=15)
plt.plot(DATA1[:,0], DATA1[:,4], label="Runge Kutta")
plt.plot(DATA2[1][:,0], DATA2[1][:,4], label="Euler, 10 s")
plt.plot(DATA3[1][:,0], DATA3[1][:,4], label="Euler, 1 s")
plt.plot(DATA4[1][:,0], DATA4[1][:,4], label="Euler, 0.1 s")
plt.grid("True")
plt.legend(loc=3,fontsize=15)
plt.xticks(np.arange(min(DATA1[:,0]), max(DATA1[:,0])+1, 10))
plt.show()
```



In [67]:
```
plt.figure(figsize=(10,10))
plt.title(r"Time vs. Mass Flow Out", fontsize=20)
plt.xlabel(r"Time, second", fontsize=15)
plt.ylabel(r"Mass Rate, kg/s", fontsize=15)
plt.plot(DATA1[:,0], DATA1[:,5], label="Runge Kutta")
plt.plot(DATA2[1][:,0], DATA2[1][:,5], label="Euler, 10 s")
plt.plot(DATA3[1][:,0], DATA3[1][:,5], label="Euler, 1 s")
plt.plot(DATA4[1][:,0], DATA4[1][:,5], label="Euler, 0.1 s")
plt.grid("True")
plt.legend(loc=3,fontsize=15)
plt.xticks(np.arange(min(DATA1[:,0]), max(DATA1[:,0])+1, 10))
plt.show()
```



In [23]:
```
head2=np.array(['Pressure, pa','Water Mass, kg','Steam Mass, kg','Water Level, m','Mass Flow Rate Out, kg/s'])
Resultdata=pd.DataFrame(np.vstack([DATA1[:5,1:],
           ['Pressure, pa','Water Mass, kg','Steam Mass, kg','Water Level, m','Mass Flow Rate Out, kg/s'],
           DATA1[1895:1905,1:],
           ['Pressure, pa','Water Mass, kg','Steam Mass, kg','Water Level, m','Mass Flow Rate Out, kg/s'],
           DATA1[-6:,1:]]),
           index=np.r_[DATA1[:5,0],['...'],DATA1[1895:1905,0],['....'],DATA1[-6:,0]],
           columns=head2)

Resultdata
```

Out[23]:

	Pressure, pa	Water Mass, kg	Steam Mass, kg	Water Level, m	Mass Flow Rate Out, kg/s
0.0	9142448.0	202200.77203839135	2530.9811216897892	44.24111948331539	579.9701731143964
0.1	9142254.178316783	202139.35868859704	2536.1718076915345	44.22747752502005	579.96144329609
0.2	9142060.343635967	202077.94618404726	2541.3625296788227	44.2138358623971	579.9527128386877
0.30000000000000004	9141866.495952826	202016.53452480363	2546.553287654567	44.20019449546499	579.9439817419642
0.4	9141672.635262629	201955.12371092782	2551.744081621681	44.18655342424217	579.9352500056942
...	Pressure, pa	Water Mass, kg	Steam Mass, kg	Water Level, m	Mass Flow Rate Out, kg/s
189.49999999999352	8771747.240710955	87358.38564102375	12434.545917156347	18.944523741596996	563.1737739584515
189.5999999999935	8771545.872033577	87298.59559025333	12439.808820866758	18.93146577164666	563.1645946909704
189.6999999999935	8771344.487378396	87238.80642552527	12445.071765190563	18.918408112450898	563.1554146340321
189.7999999999935	8771143.08673672	87179.01814691543	12450.334750131613	18.905350764034385	563.1462337872241
189.8999999999935	8770941.670099853	87119.23075449977	12455.59777569376	18.89229372642182	563.1370521501335
189.9999999999935	8765765.513246043	87118.56814725586	12434.544311847832	18.88979236468101	562.9010745237226
190.09999999999349	8760587.270174354	87117.9058403483	12413.504927532045	18.887290153679874	562.6649613503193
190.19999999999348	8755406.947625255	87117.24383394893	12392.479628248215	18.884787096523095	562.4287128576034
190.29999999999347	8750224.552354526	87116.58212822929	12371.468419479508	18.88228319632212	562.1923292740275
190.39999999999347	8745040.091133216	87115.92072336058	12350.47130669041	18.879778456195172	561.9558108288174
....	Pressure, pa	Water Mass, kg	Steam Mass, kg	Water Level, m	Mass Flow Rate Out, kg/s
263.7999999999915	5390316.390339157	86705.37204861298	480.0426971000626	17.3181472971352	398.5633932006167
263.8999999999915	5386452.662179673	86704.90342125439	468.1495689045616	17.316353689761947	398.3605346996277
263.99999999999153	5382589.292789439	86704.4350183909	456.26686899566704	17.314560303450516	398.1576528383989
264.09999999999155	5378726.289308221	86703.96684003797	444.3945962461997	17.31276714132964	397.9547479646751
264.1999999999916	5374863.658869014	86703.49888621065	432.53274950965516	17.310974206524914	397.7518204261141
264.2999999999916	5371001.408597976	86703.03115692356	420.68132762022344	17.30918150215877	397.5488705702839

In [ ]:
```
book = xlsxwriter.Workbook("Testing.xlsx")
sheet1=book.add_worksheet('Pressure')
sheet2=book.add_worksheet('Water Mass')
sheet3=book.add_worksheet('Steam Mass')
sheet4=book.add_worksheet('Water Level')
sheet5=book.add_worksheet('Mass Flow')


def writedata(array=None, sheet=None,col=None):
    for i in np.arange(len(array)):
        sheet.write(i+1,col+1,array[i])

writedata(DATA1[:,0][::100],sheet1,0)
writedata(DATA1[:,1][::100]/1000,sheet1,1)
writedata(DATA2[1][:,1]/1000,sheet1,2)
writedata(DATA3[1][:,1][::10]/1000,sheet1,3)
writedata(DATA4[1][:,1][::100]/1000,sheet1,4)

writedata(DATA1[:,0],sheet2,6)
writedata(DATA1[:,1]/1000,sheet2,7)
writedata(DATA1[:,2]/1000,sheet2,8)
writedata(DATA1[:,3]/1000,sheet2,9)
writedata(DATA1[:,4],sheet2,10)
writedata(DATA1[:,5],sheet2,11)

book.close()
```