Question 1

A)

To derive an equation, begin with the mass balance equation:

$$ Input + Generation = Output + Accumulation $$
where:

> $ Input = 10 L/s $
> $ Generation = 0 $
> $ Output = 2 L/s $
> $ Accumulation = ? $

Solving for Accumulation we find:

$$ Accumulation = 8 L/s = V'(t) $$

Giving us the first order differential equation: $$ V'(t) = 8$$

Solving:

$$ V = \int_{0}^t 8 dt $$$$ V = 8 *t + C $$

Since at $ t = 0$, $ V = 0$, therefore $C = 0 $.

Max volume = 400 L, so we solve $ 400 L = 8 L/s * t $

$$ t = 50 s$$
The time to fill the tank to maximum capacity (400L) would be 50 seconds.

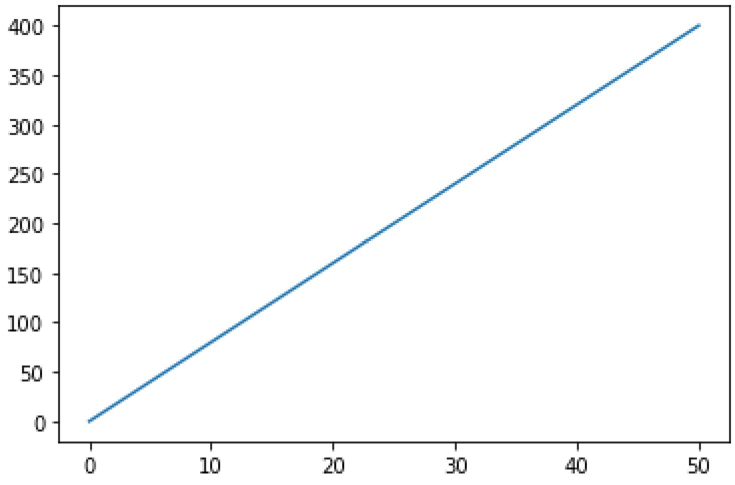
B)

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Defining the function for the ODE derived in part A): dV/dt = 8.
def f(t,V):
    return 8

# Define Euler method

h = 0.1 # Define step size
t = np.arange(0, 50 + h, h) # Define range
V0 = 0 # Initial volume of water

V = np.zeros(len(t)) # Initialize volume vector
V[0] = V0 # Set initial volume in vector
```

```
for i in range(0,len(t)-1): # Iterate based on step size
    V[i+1] = V[i] + h*f(t[i],V[i])

plt.plot(t,V) # Plot to see value

print('At t=',t[len(t)-1],"the volume is",V[len(t)-1]) # Print solution
```

```
At t= 50.0 the volume is 400.0000000000035
```

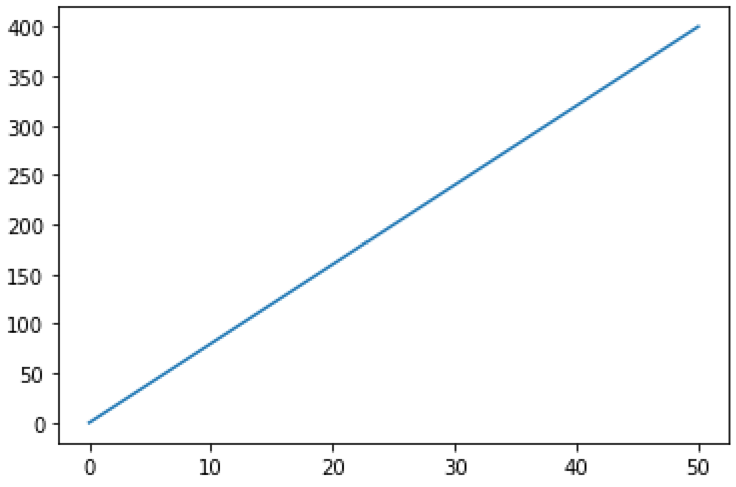


C)

In [138…

```
from scipy.integrate import solve_ivp

# Initialize ODE function, form dV/dt = fc(t)

def fc(t,V):
    return [8]

h = 0.1 # Define step size
t_span = np.array([0,50])


V0c = np.array([0]) # Initial value

# Set T-Eval for percent error calc later
teval = np.linspace(0,50,len(t))

# Call solve_ivp function with specified parameters
sol = solve_ivp (fc, [0,50], V0c, method ='Radau', max_step = h, t_eval = teval)

# Sort results
t_s = sol.t
Vc = sol.y[0]

# Plot results
plt.plot(t_s,Vc)

print('At t=',t_s[len(t_s)-1],"the volume is",Vc[len(Vc)-1])
```

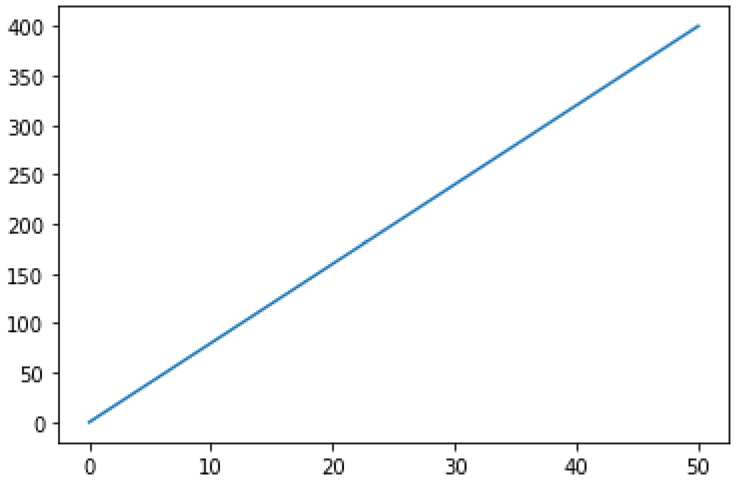
```
At t= 50.0 the volume is 400.0
```



In [140…

```
h2 = 0.001 # Define step size

# Call funciton again
sol2 = solve_ivp (fc, [0,50], V0c, method='Radau', max_step = h2, t_eval = teval)

# Sort output
t_s2 = sol2.t
Vc2 = sol2.y[0]

# Plot results
plt.plot(t_s2,Vc2)

print('At t=',t_s2[len(t_s2)-1],"the volume is",Vc2[len(Vc2)-1])
```

```
At t= 50.0 the volume is 400.0
```



D)

There are very little differences between each solution, likely due to the simplicity of the function being tested. Looking at the axis of the percent difference over time, we see that the error is extremely minimal and stays around zero.

In [148…

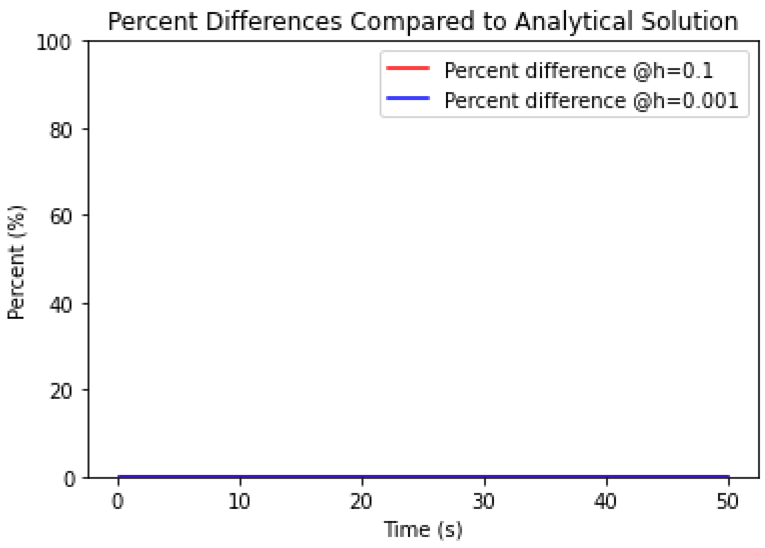
```
print(len(V),len(Vc),len(Vc2))

VcPD = np.zeros(len(t))
```

```
Vc2PD = np.zeros(len(t))

for i in range(0,len(V)-1):
    VcPD[i] = (V[i]-Vc[i])/V[i]
    Vc2PD[i] = (V[i]-Vc2[i])/V[i]

plt.plot(t,VcPD,color='r', label = 'Percent difference @h=0.1')
plt.plot(t,Vc2PD,color='b', label = 'Percent difference @h=0.001')
plt.legend()
plt.title('Percent Differences Compared to Analytical Solution')
plt.xlabel('Time (s)')
plt.ylabel('Percent (%)')
plt.ylim(0,100)
```

```
501 501 501
<ipython-input-148-9d520ffe1a97>:7: RuntimeWarning: invalid value encountered in double_
scalars
  VcPD[i] = (V[i]-Vc[i])/V[i]
<ipython-input-148-9d520ffe1a97>:8: RuntimeWarning: invalid value encountered in double_
scalars
  Vc2PD[i] = (V[i]-Vc2[i])/V[i]
```

Out[148…

```
(0.0, 100.0)
```

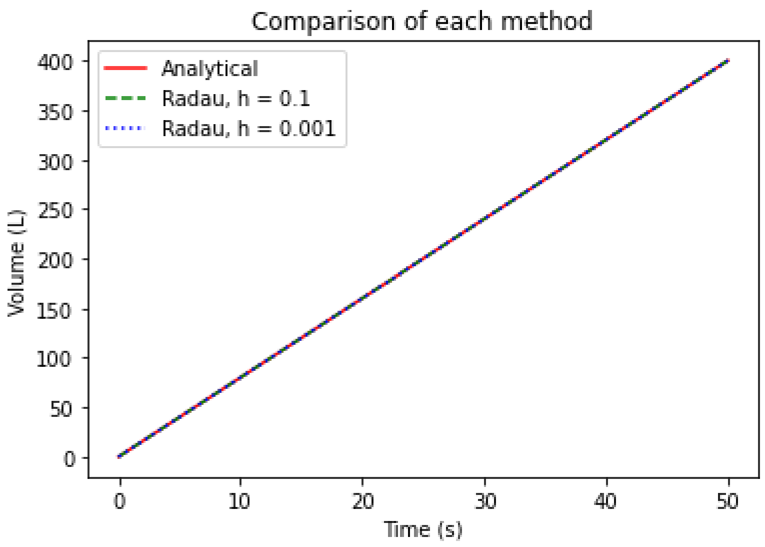


In [149…

```
# Sanity Check
plt.plot(t,V, color='r', label='Analytical')
plt.plot(t_s,Vc, color='g', label='Radau, h = 0.1',linestyle = 'dashed')
plt.plot(t_s2,Vc2, color='b', label='Radau, h = 0.001',linestyle = 'dotted')
plt.legend()
plt.title('Comparison of each method')
plt.xlabel('Time (s)')
plt.ylabel('Volume (L)')
```

Out[149…

```
Text(0, 0.5, 'Volume (L)')
```



Question 2

A)

Analyzing the reactions starting at K1, the ratio for K1/K1r is about 1, meaning that the rates of reaction are roughly equal between A and B. Seeing that K2 is also much greater than K1, all of B will be turn to C very quickly. B is an intermediate and its levels will likely remain low. No amount of A will directly form C, but it is unclear if C will form A as K3 is not provided. This will affect how the system will behave at steady state.

The system will take the form of the following equations:

$$ \frac{d[A]}{dt} = K_{1r}[B] - (K_{3r}+K_1)[A] + K_3[C] $$

$$ \frac{d[B]}{dt} = K_1[A] - (K_2+K_{1r})[B] $$
$$ \frac{d[C]}{dt} = K_2[B] + K_{3r}[A] - K_3[C] $$

In [154…

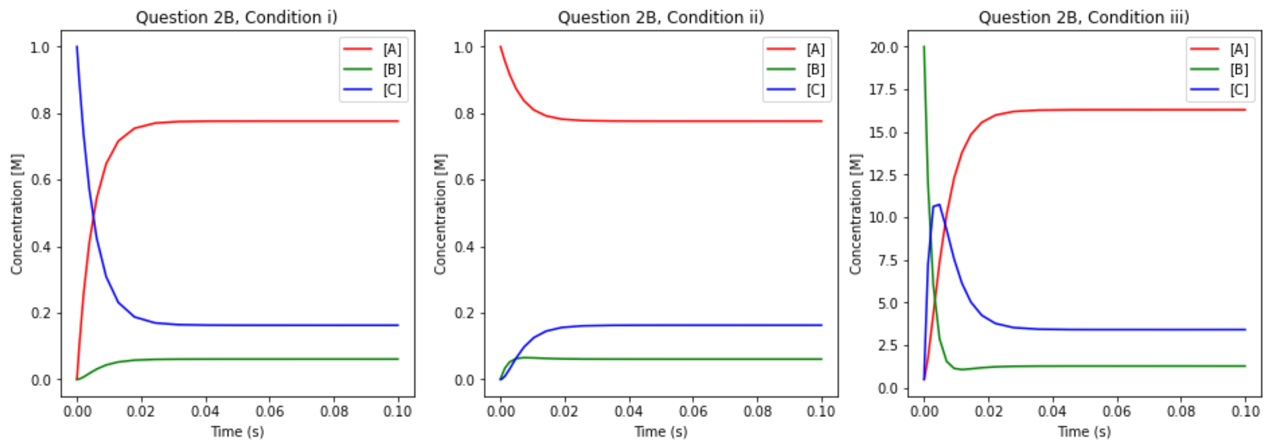
```
def conc(t,c): # Define function for equation, take in c - vector of all concentrations

     # Assign initial concentrations based on input
    ca = c[0]
    cb = c[1]
    cc = c[2]

     # Initialize reactions rates
    k1 = 33.7
    k1r = 28.7
    k2 = 400
    k3 = 150
    k3r = 0.02

    # Set up ODE to solve
    dAdt = k1r*cb - (k3r+k1)*ca + k3*cc

    dBdt = k1*ca - (k2+k1r)*cb
    dCdt = k2*cb + k3r*ca - k3*cc

    return [dAdt, dBdt, dCdt]

# Define initial values
c0i = np.array([0,0,1])
c0ii = np.array([1,0,0])
c0iii = np.array([0.5,20,0.5])

# Solve IVP based on initial values
sol2bi = solve_ivp(conc,[0,0.1],c0i)
sol2bii = solve_ivp(conc,[0,0.1],c0ii)
sol2biii = solve_ivp(conc,[0,0.1],c0iii)
```

In [155…

```
# Plot results
fig = plt.subplots(1,3, figsize = (16,5))



plt.subplot(131)
plt.plot(sol2bi.t, sol2bi.y[0], color='r', label='[A]')
plt.plot(sol2bi.t, sol2bi.y[1], color='g', label='[B]')
plt.plot(sol2bi.t, sol2bi.y[2], color='b', label='[C]')
plt.legend()
plt.title('Question 2B, Condition i)')
plt.xlabel('Time (s)')
plt.ylabel('Concentration [M]')

plt.subplot(132)
plt.plot(sol2bii.t, sol2bii.y[0], color='r', label='[A]')
plt.plot(sol2bii.t, sol2bii.y[1], color='g', label='[B]')
plt.plot(sol2bii.t, sol2bii.y[2], color='b', label='[C]')
plt.legend()
plt.title('Question 2B, Condition ii)')
plt.xlabel('Time (s)')
plt.ylabel('Concentration [M]')

plt.subplot(133)
plt.plot(sol2biii.t, sol2biii.y[0], color='r', label='[A]')
plt.plot(sol2biii.t, sol2biii.y[1], color='g', label='[B]')
plt.plot(sol2biii.t, sol2biii.y[2], color='b', label='[C]')
plt.legend()
plt.title('Question 2B, Condition iii)')
plt.xlabel('Time (s)')
plt.ylabel('Concentration [M]')
```

Out[155…

```
Text(0, 0.5, 'Concentration [M]')
```



C)

i) To reduce the model as much as possible, I will eliminate the step from A to B since the ratio between the forward and reverse reactions is almost 1, an just form C from A. Also, since the rate of reaction k3r is so small, it can be ignored giving the following system:

$$\require{mhchem} \ce{A <=> C} $$

    <nb>

Where $K_2$ will represent the reaction from A to C and $K_3$ represents the reverse reaction.

ii)

In [157…

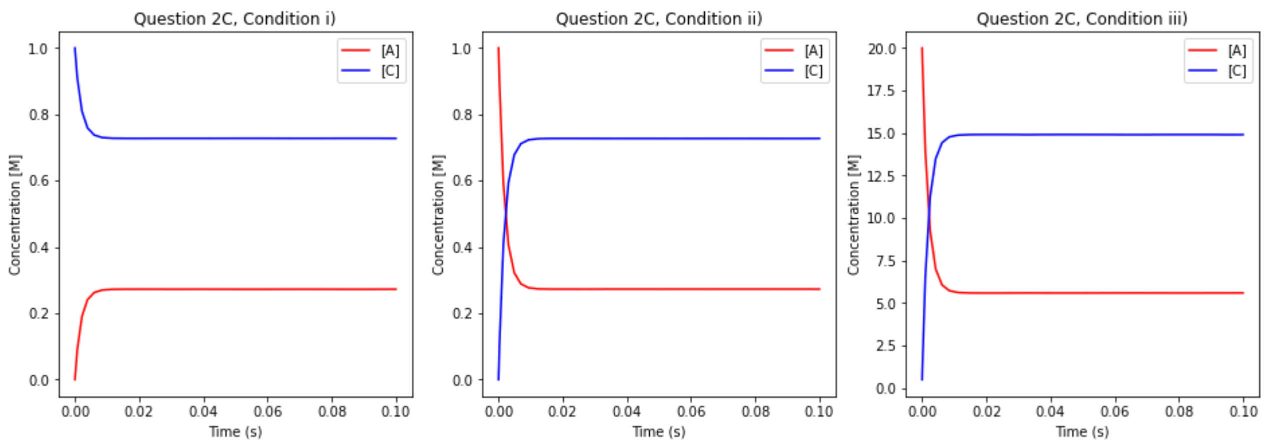
```
def concC(t,c): # Define function for equation, take in c - vector of all concentration

    # Assign initial concentrations based on input
    ca = c[0]
    cc = c[1]

    # Initialize reactions rates
    k2 = 400
    k3 = 150

    # Set up ODE to solve
    dAdt = -k2*ca + k3*cc
    dCdt = k2*ca - k3*cc

    return [dAdt, dCdt]

# Define initial values
c0ci = np.array([0,1])
c0cii = np.array([1,0])
c0ciii = np.array([20,0.5])

# Solve IVP based on initial values
sol2ci = solve_ivp(concC,[0,0.1],c0ci)
sol2cii = solve_ivp(concC,[0,0.1],c0cii)
sol2ciii = solve_ivp(concC,[0,0.1],c0ciii)
```

In [158…

```python
# Plot results

fig, axs = plt.subplots(1,3, figsize = (16,5))

plt.subplot(131)
plt.plot(sol2ci.t, sol2ci.y[0], color='r', label='[A]')
plt.plot(sol2ci.t, sol2ci.y[1], color='b', label='[C]')
plt.legend()
plt.title('Question 2C, Condition i)')
plt.xlabel('Time (s)')
plt.ylabel('Concentration [M]')

plt.subplot(132)
plt.plot(sol2cii.t, sol2cii.y[0], color='r', label='[A]')
plt.plot(sol2cii.t, sol2cii.y[1], color='b', label='[C]')
plt.legend()
plt.title('Question 2C, Condition ii)')
plt.xlabel('Time (s)')
plt.ylabel('Concentration [M]')

plt.subplot(133)
plt.plot(sol2ciii.t, sol2ciii.y[0], color='r', label='[A]')
plt.plot(sol2ciii.t, sol2ciii.y[1], color='b', label='[C]')
plt.legend()
plt.title('Question 2C, Condition iii)')
plt.xlabel('Time (s)')
plt.ylabel('Concentration [M]')
```

Out[158…

```
Text(0, 0.5, 'Concentration [M]')
```



The results of the reduced model are essentially opposite of each other in terms of [A] and [C]. Since $K_2$ is much larger than $K_3$ this solution makes sense since the concentration of C is favored. Comparing to the full model, it is apparent that B limits the amount of C that is formed, and without it we see the unregulated production of C. This means that my assumptions were not adequate in reducing the system since I have opposite values for the concentrations that are occuring. Both systems still reach steady state in a similar timeframe (<0.02 seconds) and the values of the concentration at steady state are in the same ranges, ignoring the mismatch between A and C. This at least means that I am not that far off in my solution, and reducing the model to a point that still includes B should be my next step.

```
In [ ]:
```