Introduction

I have programmed a simulator of a heated liquid water tank in python based on a mathematical model of the temperature variation of the given system:

$$c\rho VT'(t) = P(t-\tau) + c\rho F[T_{in}(t) - T(t)] + U[T_{env}(t) - T(t)]$$

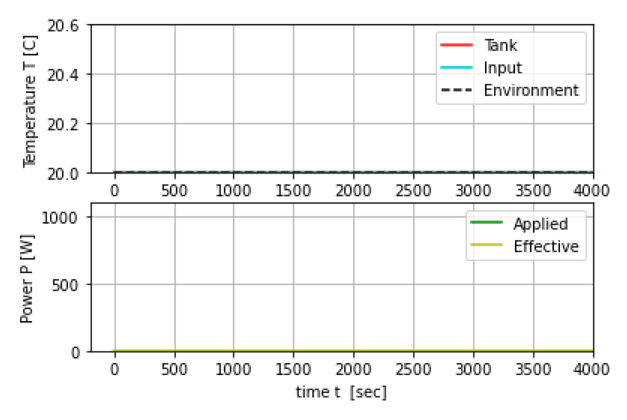
where,
c= Specific heat capacity of liquid $J/(kg.K)$
ρ = Density of liquid kg/m^3
V= Liquid volume in tank m^3
P= Supplied power W
τ= Time delay in the temperature response s
F= Liquid Flow m^3/s
U= Heat transfer coefficient of tank W/K
T_{in}= Temperature of liquid inflow ^oC
T_{env} = Environmental temperature ^oC
T= Temperature of liquid ^oC

I used the Euler forward method for solving the model in discrete form. The basic form of Euler forward method is:
$T_{k+1} = T_k + Ts * T'(k)$
where,
Ts= Simulation time-step
T_k= Current temperature of liquid in time t_k
T_{k+1}= Next Temperature of liquid in time t_{k+1}
T'_k= Time derivative of temperature

The simulation program including all the details with appropriate comments and the output is given below:

Task 1: Implementation

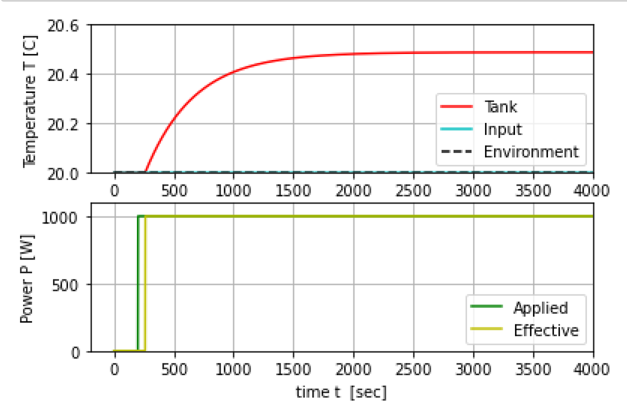
In [6]:
```python
#%% Import packages
import matplotlib.pyplot as plt
import numpy as np


#%% Model constants and parameters
F = 0.25e-3          # {m^3/s} Liquid Flow
c = 4200             # {J/(kg.K)} Specific heat capacity of liquid
rho = 1000           # {kg/m^3} Density of liquid
V = 0.2              # {m^3} Liquid volume in tank
U = 1000             # {W/K} Heat transfer coefficient of tank
T_env = 20           # {degree celcius} Environmental temperature
T_in = 20            # {degree celcius} Temperature of liquid flow
t_d = 60             # {seconds} Time delay in the temperature response



#%% Simulation time settings
Ts = 1         #{s}
t_start = 0
t_stop = 4000
N_sim = int((t_stop-t_start)/Ts) + 1

#%% Pre-allocation of arrays for plotting
t_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
T_in_array = np.zeros(N_sim) + T_in
T_env_array = np.zeros(N_sim) + T_env
P_array = np.zeros(N_sim)
effective_P = np.zeros(N_sim)

#%% Initialization
T_k = 20
P_k = 0
T_min = 20
T_max = 30

#%% Pre-allocation of arrays for time delay
Nd = int(round(t_d/Ts)) + 1
delay_array = np.zeros(Nd) + P_k


#%% Simulation loop
for k in range(0,N_sim):
    t_k = k * Ts        # Time

    P_in_k = 0

    # Time delay:
    P_k = delay_array[-1]
    delay_array[1:] = delay_array[0:-1]
    delay_array[0] = P_in_k

    #  Time derivative:
    dT_dt_k = (1/(c*rho*V))*P_k + (F/V)*(T_in-T_k) + (U/(c*rho*V))*(T_env-T_k)

    # State updates using the Euler method:
    T_kp1 = T_k + dT_dt_k * Ts
    T_kp1 = np.clip(T_kp1,T_min,T_max)
    # Writing values to arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_kp1
    P_array[k] = P_in_k
    effective_P[k] = P_k

    # Time-shift
    T_k = T_kp1

#%% Plotting
plt.close('all')
plt.figure(1)
plt.subplot(2,1,1)
plt.plot(t_array,T_array,'r',t_array,T_in_array,'c',t_array,T_env_array,'k--')
plt.xlabel('time t');plt.ylabel('Temperature T [C]')
plt.legend(['Tank','Input','Environment','Location','southeast'])
plt.axis([-200,4000,20,20.6])
plt.grid()
plt.subplot(2,1,2)
plt.plot(t_array,P_array,'g',t_array,effective_P,'y')
plt.xlabel('time t  [sec]');plt.ylabel('Power P [W]')
plt.legend(['Applied','Effective'])
plt.axis([-200,4000,0,1100])
plt.grid()
```



Here, initially we have not applied any input power P so the output temperature T also remains constant at 20^oC along with T_{in} and T_{env}.

Task 2: Simulation

In [7]:
```python
#%% Simulation loop
for k in range(0,N_sim):
    t_k = k * Ts        # Time

    # Selecting inputs:
    if t_k > 200:
        P_in_k = 1000
    else:
        P_in_k = 0

    # Time delay:
    P_k = delay_array[-1]
    delay_array[1:] = delay_array[0:-1]
    delay_array[0] = P_in_k

    #  Time derivative:
    dT_dt_k = (1/(c*rho*V))*P_k + (F/V)*(T_in-T_k) + (U/(c*rho*V))*(T_env-T_k)

    # State updates using the Euler method:
    T_kp1 = T_k + dT_dt_k * Ts
    T_kp1 = np.clip(T_kp1,T_min,T_max)
    # Writing values to arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_kp1
    P_array[k] = P_in_k
    effective_P[k] = P_k

    # Time-shift
    T_k = T_kp1

#%% Plotting
plt.close('all')
plt.figure(1)
plt.subplot(2,1,1)
plt.plot(t_array,T_array,'r',t_array,T_in_array,'c',t_array,T_env_array,'k--')
plt.xlabel('time t');plt.ylabel('Temperature T [C]')
plt.legend(['Tank','Input','Environment','Location','southeast'])
plt.axis([-200,4000,20,20.6])
plt.grid()
plt.subplot(2,1,2)
plt.plot(t_array,P_array,'g',t_array,effective_P,'y')
plt.xlabel('time t  [sec]');plt.ylabel('Power P [W]')
plt.legend(['Applied','Effective'])
plt.axis([-200,4000,0,1100])
plt.grid()
```



In the next step, we have applied a unit step input power P_in_k of magnitude 1000W after 200 secs which is shown by the green line in the plot. But, we can see that due to delay in power applied, which is shown by the yellow line in the plot, the output temperature starts to change only after delay time of 60 seconds.

This is a first order Time-constant system. The step response of a standard first order system is:

$y(t) = KU(1 - e^{-\frac{t}{T}})$ ...(0), where, KU is gain and T is the time constant. We need to first convert the given model to this form for transient and steady state analysis.

The model of the temperature variation of the given system is:

$c\rho VT'(t) = P(t-\tau) + c\rho F[T_{in}(t) - T(t)] + U[T_{env}(t) - T(t)]$ ....(1)

We will now calculate the transfer functions from P to T , T_{in} to T, and T_{env} to T. Taking laplace transform of (1) gives:

$c\rho V[sT(s) - T_0] = P(s)e^{-\tau s} + c\rho F[T_{in}(s) - T(s)] + U[T_{env}(s) - T(s)]$ ....(2)

Setting initial value to zero. i.e. $T_0 = 0$ and further simplifying (2) we get:

$T(s)[(\frac{c\rho V}{c\rho F + U})s + 1] = P(s)e^{-\tau s} + (c\rho F)T_{in}(s) + UT_{env}(s)$ ....(3)

Keeping the given values of the contants and parameters in (3) we get:

$T(s) = (\frac{4.878*10^{-4}}{409.75s+1})P(s)e^{-60s} + (\frac{0.5122}{409.75s+1})T_{in}(s) + (\frac{0.4878}{409.75s+1})T_{env}(s)$ ....(4)

We have, $P(s) = \frac{1000}{s}$, $T_{in}(s)$ = $\frac{20}{s}$, and $T_{env}(s)$ = $\frac{20}{s}$, then (4) becomes:

$T(s) = \frac{0.4878e^{-60s}}{(409.75s+1)s} + \frac{20}{(409.75s+1)s}$ ....(5)

Taking the inverse laplace transform on (5),we get,

$T(t) = 0.4878(1 - e^{-\frac{t-60}{409.75}}) + 20(1 - e^{-\frac{t}{409.75}})$ ....(6)

which is the step response of the temperature variation of the given system.

Now,lets find the steady state value i.e. the value at t = ∞

In [8]:
```python
t_steady = np.inf
T_steady = 0.4878*(1- np.exp(-((t_steady-60)/409.75))) + 20*(1- np.exp(-(t_steady/409.75)))
print('The steady state value is:',T_steady, 'degree Celcius.')
```

```
The steady state value is: 20.4878 degree Celcius.
```

The calculated model steady state value is in line with the steady state value as seen from the output plot.

We have, from (0) and (6), we have model Time constant = 409.75 seconds.

Also, time constant T is the time at which the output gets to 63% of steady state value. i.e. 63% of (20.4878 - 20) + 20 = 20.307314. To find the time at which the output temperature is nearly 20.307314. Let's search T_array which contains the output values.

In [9]:
```python
g = 0
for i in range(0,N_sim):
    if (20.307<T_array[i]<20.308):
        g += 1
        print(f'\nTemperature{g}:{T_array[i]:.5f} degree Celcius      Time{g}: {i-260} secon
ds')
```

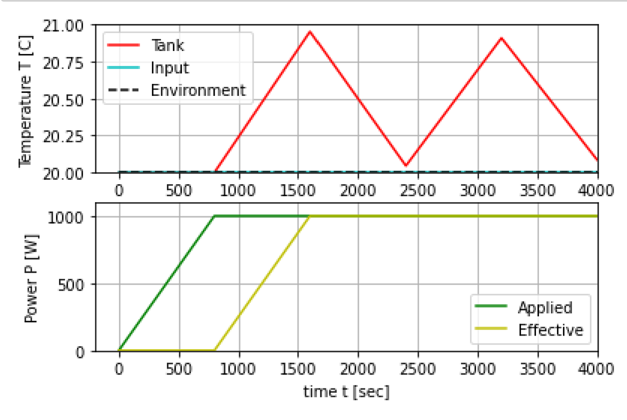
```
Temperature1:20.30736 degree Celcius      Time1: 408 seconds

Temperature2:20.30780 degree Celcius      Time2: 409 seconds
```

From above results, it is clear that the time constant from response is almost same as from the model.

Therefore, we can conclude that the simulator behaves correctly, i.e. according to the given model both dynamically (as seen from the transient responses) and in steady-state.

Task 3: Stability of the simulator

In [10]:
```python
#%% Simulation time settings
Ts = 800         #{s}
t_start = 0
t_stop = 4000
N_sim = int((t_stop-t_start)/Ts) + 1

#%% Pre-allocation of arrays for plotting
t_array = np.zeros(N_sim)
T_array = np.zeros(N_sim)
T_in_array = np.zeros(N_sim) + T_in
T_env_array = np.zeros(N_sim) + T_env
P_array = np.zeros(N_sim)
effective_P = np.zeros(N_sim)

#%% Initialization
T_k = 20
P_k = 0
T_min = 20
T_max = 30

#%% Pre-allocation of arrays for time delay
Nd = int(round(t_d/Ts)) + 1
delay_array = np.zeros(Nd) + P_k


#%% Simulation loop
for k in range(0,N_sim):
    t_k = k * Ts        # Time

    # Selecting inputs:
    if t_k > 200:
        P_in_k = 1000
    else:
        P_in_k = 0

    # Time delay:
    P_k = delay_array[-1]
    delay_array[1:] = delay_array[0:-1]
    delay_array[0] = P_in_k

    #  Time derivative:
    dT_dt_k = (1/(c*rho*V))*P_k + (F/V)*(T_in-T_k) + (U/(c*rho*V))*(T_env-T_k)

    # State updates using the Euler method:
    T_kp1 = T_k + dT_dt_k * Ts
    T_kp1 = np.clip(T_kp1,T_min,T_max)
    # Writing values to arrays for plotting:
    t_array[k] = t_k
    T_array[k] = T_kp1
    P_array[k] = P_in_k
    effective_P[k] = P_k

    # Time-shift
    T_k = T_kp1

#%% Plotting
plt.close('all')
plt.figure(1)
plt.subplot(2,1,1)
plt.plot(t_array,T_array,'r',t_array,T_in_array,'c',t_array,T_env_array,'k--')
plt.xlabel('time t');plt.ylabel('Temperature T [C]')
plt.legend(['Tank','Input','Environment','Location','southeast'])
plt.axis([-200,4000,20,21])
plt.grid()
plt.subplot(2,1,2)
plt.plot(t_array,P_array,'g',t_array,effective_P,'y')
plt.xlabel('time t [sec]');plt.ylabel('Power P [W]')
plt.legend(['Applied','Effective'])
plt.axis([-200,4000,0,1100])
plt.grid()
```



Here, I have selected simulator time step as 800 seconds which is a very large value.

Hence, from the above output, it is clear that the simulator becomes numerically inaccurate and unstable, if we select a (too) large simulator time step.