In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from scipy.interpolate import interp1d
from scipy.optimize import fsolve
from simple_pid import PID  # pip install simple_pid
```

In [2]:
```
# fully open gate height estimated from diagram on source 4
print('Gate dimensions at "empty" dam (30m) and fully open gate:')
GateHeight = (503-476)/2
print('   Gate Height =',round(GateHeight,2),'m')
GateFlowArea = 3.84405**2 # m^2 so that flow rate maxes at 358.5 m^3/s
GateWidth = GateFlowArea/GateHeight
print('   Gate Width =',round(GateWidth,4),'m')
def FlowRate(h,Δz):
    # h  = height of opening of the gate valve
    # Δz = dam depth can range from 30 to 220 m
    # returns the flow rate (q)
    width = GateWidth #m
    A = h*width
    g = 9.81 # m/s^2
    v = np.sqrt(2*g*Δz) # m/s - velocity of water
    q = A*v # flow rate through gate maxes at 358.5 m^3/s
    return q
```

```
Gate dimensions at "empty" dam (30m) and fully open gate:
   Gate Height = 13.5 m
   Gate Width = 1.0946 m
   Max Flow rate = 358.5 m^3/s
```

In [3]:
```
def PowerOut(h,Δz):
    # This function give the power output given the height of the
    #     gate's opening, and depth of the dam
    # returns power output in MW

    q = FlowRate(h,Δz)
    g = 9.81 # m/s^2

    # efficiency found using max rated flowrate (358.5 m^3/s)
    # and rated output (640 MW)
    η = .8286

    ρ = 998.29 # kg/m^3 - density of water at 20°C
    W_dam = ρ*g*q*Δz # Energy from dam hydrostatic pressure
    W_head = 0#ρ*g*q*194 # Energy from 194 m head between gate and tur
    return η*(W_dam+W_head) / 10**6

print('At maximum gate opening and "empty" dam (30m):')
```

```
At maximum gate opening and "empty" dam (30m):
   Max power output = 87.27 MW
```

In [4]:
```
def FindOperatingHeights(Δz):
```

```
    # Need to change the gate height depending on the Dam Level
    #   in order to maintain within operating zones for turbine

    def HiMx(h):
        return PowerOut(h,Δz)-640.0
    HighMax = fsolve(HiMx,8)[0]
    if HighMax > 13.5:
        HighMax = 13.5
    if HighMax < 0.0:
        HighMax = 0.0

    def HiLo(h):
        return PowerOut(h,Δz)-550.0
    HighLow = fsolve(HiLo,8)[0]
    if HighLow > 13.5:
        HighLow = 13.5
    if HighLow < 0:
        HighLow = 0.0

    def LoMx(h):
        return PowerOut(h,Δz)-350.0
    LowMax = fsolve(LoMx,3)[0]
    if LowMax > 13.5:
        LowMax = 13.5
    if LowMax < 0:
        LowMax = 0.0

    def LoLo(h):
        return PowerOut(h,Δz)-0.0
    LowLow = fsolve(LoLo,3)[0]
    if LowLow > 13.5:
        LowLow = 13.5
    if LowLow < 0:
        LowLow = 0.0

```

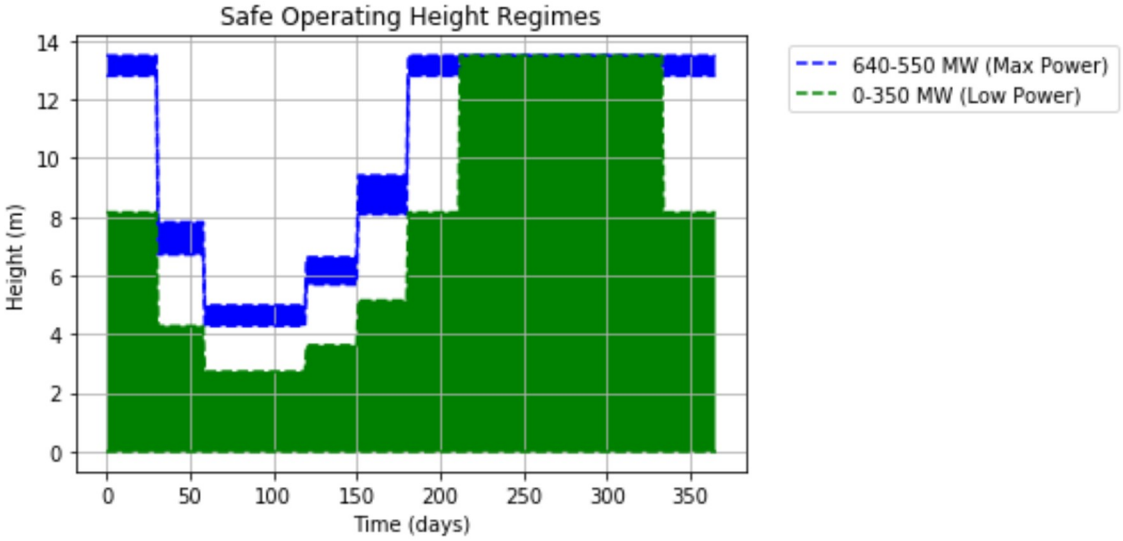
In [5]:

```
# Step function of dam level
ndays = 365
ns = ndays
#DamLevel = np.array([106,163,220,220,182,144,106,68,30,30,68,106])
DamLevel = np.ones(ndays+1)
DamLevel[0:31] *= 106
DamLevel[31:59] *= 163
DamLevel[59:90] *= 220
DamLevel[90:120] *= 220
DamLevel[120:151] *= 182
DamLevel[151:181] *= 144
DamLevel[181:212] *= 106
DamLevel[212:243] *= 68
DamLevel[243:273] *= 30
DamLevel[273:304] *= 30
DamLevel[304:334] *= 68
DamLevel[334:366] *= 106

# Find Operating Limits to graph
TopMax = np.ones(len(DamLevel))
```

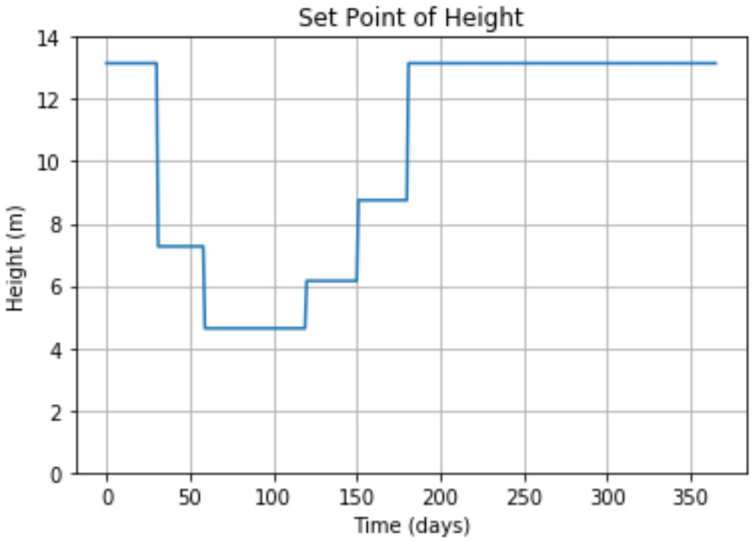
```
TopLow = np.ones(len(DamLevel))
BotMax = np.ones(len(DamLevel))
BotLow = np.ones(len(DamLevel))
for i in range(len(DamLevel)):
    #print(i)
    z = DamLevel[i]
    values = FindOperatingHeights(z)
    TopMax[i] = values[0]
    TopLow[i] = values[1]
    BotMax[i] = values[2]
    BotLow[i] = values[3]

ts = np.linspace(0,ndays,ndays+1) # Each day of the year

print('The dam level changes throughout the year')
plt.grid()
plt.title('Dam Level Throughout the year')
plt.plot(ts,DamLevel,label='Dam Level')
plt.xlabel('Time (days)')
plt.ylabel('Height (m)')
plt.show()

print('Based on the energy balance, the colored zones are the height t
plt.grid()
plt.title('Safe Operating Height Regimes')
plt.plot(ts,TopMax,'--b',label='640-550 MW (Max Power)')
plt.plot(ts,TopLow,'--b')
plt.fill(np.append(ts, ts[::-1]), np.append(TopMax, TopLow[::-1]), 'b'
plt.plot(ts,BotMax,'--g',label='0-350 MW (Low Power)')
plt.plot(ts,BotLow,'--g')
plt.fill(np.append(ts, ts[::-1]), np.append(BotMax, BotLow[::-1]), 'g'
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
plt.xlabel('Time (days)')
plt.ylabel('Height (m)')
plt.show()

# Generate simulated process data
setPoint = (TopMax+TopLow)/2
for i in range (len(TopMax)):
    if setPoint[i] == TopMax[i]:
        setPoint[i] = setPoint[i-1]
#tsnew = np.linspace(0,ndays,2*ndays)
#newHeightSim = spline(ts,avgHeight,tsnew)
#process = avgHeight#newHeightSim + .01*np.sin(tsnew)

print('We want to operate in the max power regime.  So the set point i
plt.grid()
plt.title('Set Point of Height')
plt.plot(ts,setPoint)
plt.xlabel('Time (days)')
plt.ylabel('Height (m)')
plt.ylim(0,14)
plt.show()
```

```
The dam level changes throughout the year
```

Based on the energy balance, the colored zones are the height the gate can be open in safe operating conditions



We want to operate in the max power regime. So the set point is averaged between the top and bottom values of the max power regime



In [6]:

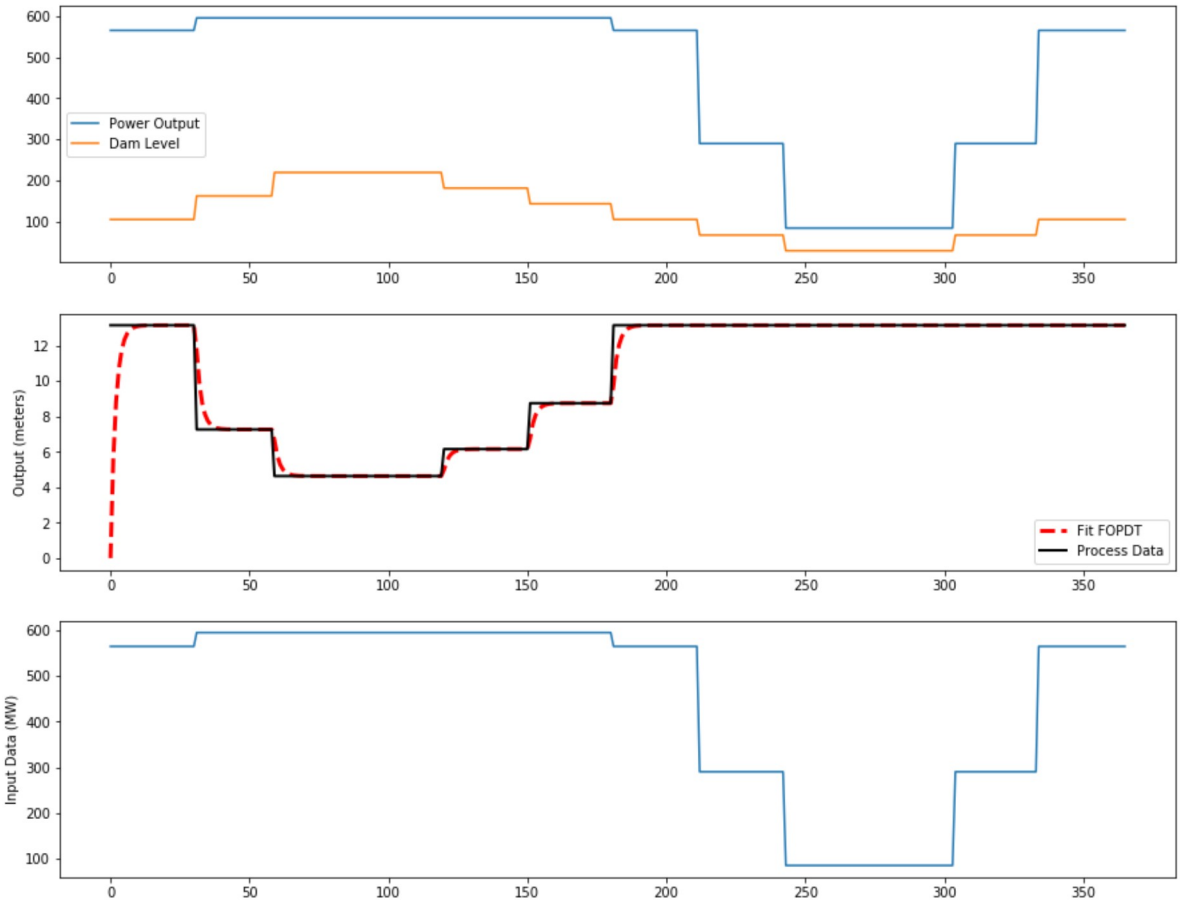
```
# define process model (to generate process data)
def process(y,t,u,Kp,taup):
    # arguments
    #  y[n] = outputs - MW
    #  t    = time
    #  dz   = Dam Level
    #  u    = input value
    #  Kp   = process gain
    #  taup = process time constant

    n = 1 #  n    = order of the system

    # equations for higher order system
    dydt = np.zeros(n)
    # calculate derivative
```

```
    dydt[0] = (-y[0] + Kp * u)/(taup/n)
    for i in range(1,n):
        dydt[i] = (y[i] + y[i-1])/(taup/n)
    return dydt

# define first-order plus dead-time approximation
def fopdt(y,t,uf,Km,taum,thetam):
    # arguments
    #  y      = output
    #  t      = time
    #  uf     = input linear function (for time shift)
    #  Km     = model gain
    #  taum   = model time constant
    #  thetam = model time constant
    # time-shift u
    try:
        if (t-thetam) <= 0:
            um = uf(0.0)
        else:
            um = uf(t-thetam)
    except:
        #print('Error with time extrapolation: ' + str(t))
        um = 0
    # calculate derivative
    dydt = (-y + Km * um)/taum
    return dydt

# specify number of steps
ns = ndays #40
# define time points
t = np.linspace(0,ns,ns+1)
delta_t = t[1]-t[0]
# define input vector - set point data
u = setPoint#np.zeros(ns+1)
#u[5:] = 1.0
# create linear interpolation of the u data versus time
uf = interp1d(t,u)

# use this function or replace yp with real process data
def sim_process_data():
    # higher order process
    n=1        # order was 10 in source code
    Kp=2.7     # gain
    taup=5.0   # time constant
    # storage for predictions or data
    yp = np.zeros(ns+1)  # process
    yp1 = np.zeros(ns+1)
    for i in range(ns+1):
        yp1[i] = PowerOut(setPoint[i],DamLevel[i])
    for i in range(1,ns+1):
        if i==1:
            yp0 = yp[0]
        ts = [delta_t*(i-1),delta_t*i]
        y = odeint(process,yp0,ts,args=(u[i],Kp,taup))
        yp0 = y[-1]
        yp[i] = y[1][n-1]
```

```
        #print(len(yp))
    return yp,yp1
yp = sim_process_data()[0]/2.693307151511013#2.4755
yp1 = sim_process_data()[1]
print('Simulated Height Output based on the Power set point data and
plt.figure(figsize=(15,12))
plt.subplot(3,1,1)
plt.plot(ts,yp1,label='Power Output')
plt.plot(ts,DamLevel,label='Dam Level')
plt.legend()

# simulate FOPDT model with x=[Km,taum,thetam]
def sim_model(Km,taum,thetam):
    # input arguments
    #Km
    #taum
    #thetam
    # storage for model values
    ym = np.zeros(ns+1)  # model
    # initial condition
    ym[0] = 0
    # loop through time steps
    for i in range(1,ns+1):
        ts = [delta_t*(i-1),delta_t*i]
        y1 = odeint(fopdt,ym[i-1],ts,args=(uf,Km,taum,thetam))
        ym[i] = y1[-1]
    return ym

# calculate model with updated parameters
Km = 1.0
taum = 1.8
thetam = 0.0
ym = sim_model(Km,taum,thetam)



# plot results
plt.subplot(3,1,2)
plt.plot(t,ym,'r--',linewidth=3,label='Fit FOPDT')
#plt.plot(t,yp,'k-',linewidth=2,label='Process Data')
plt.plot(t,u,'k-',linewidth=2,label='Process Data')
plt.ylabel('Output (meters)')
plt.legend(loc='best')
#plt.show()
#plt.subplot(2,1,2)

plt.subplot(3,1,3)
plt.plot(ts,yp1)
#plt.plot(t,u,'bx-',linewidth=2)
#plt.plot(t,uf(t),'r--',linewidth=3)
#plt.legend(['Measured','Interpolated'],loc='best')
plt.ylabel('Input Data (MW)')
plt.savefig('FOPDT.png')
plt.show()

#print('IAE was minimized manually by altering Km, taum, and thetam.'
```

```
#print('Minimum IAE at Km = 1.0, taum = 3.8, theam = 0.0')
#print('However this would lead to constant movement of the gate and
print('\nThe process values would be:\nKp =',Km,'\ntaup =',taum,'\nth
# Integral Absolute Error
iae = 0
for i in range(len(ym)):
    iae += np.abs(ym[i]-yp[i])
print('IAE =',iae)
```

```
Simulated Height Output based on the Power set point data and dam leve
l
```




```
The process values would be:
Kp = 1.0
taup = 1.8
thetap = 0.0
IAE = 91.55624744417176
```

In [7]:

```
# PID Controller parameters
Kp = Km
taup = taum
thetap = thetam

# Moderate IMC Tuning correlations from FOPDT parameters
tauc = np.max([1.0*Kp,0.8*thetap])
Kc = 1/Kp * (taup+0.5*thetap) / (tauc+0.5*thetap)
tauI = taup+0.5*thetap
tauD = taup*thetap/(2*taup+thetap)
print('Kc =',Kc,'\ntauI =',tauI,'\ntauD =',tauD)
```

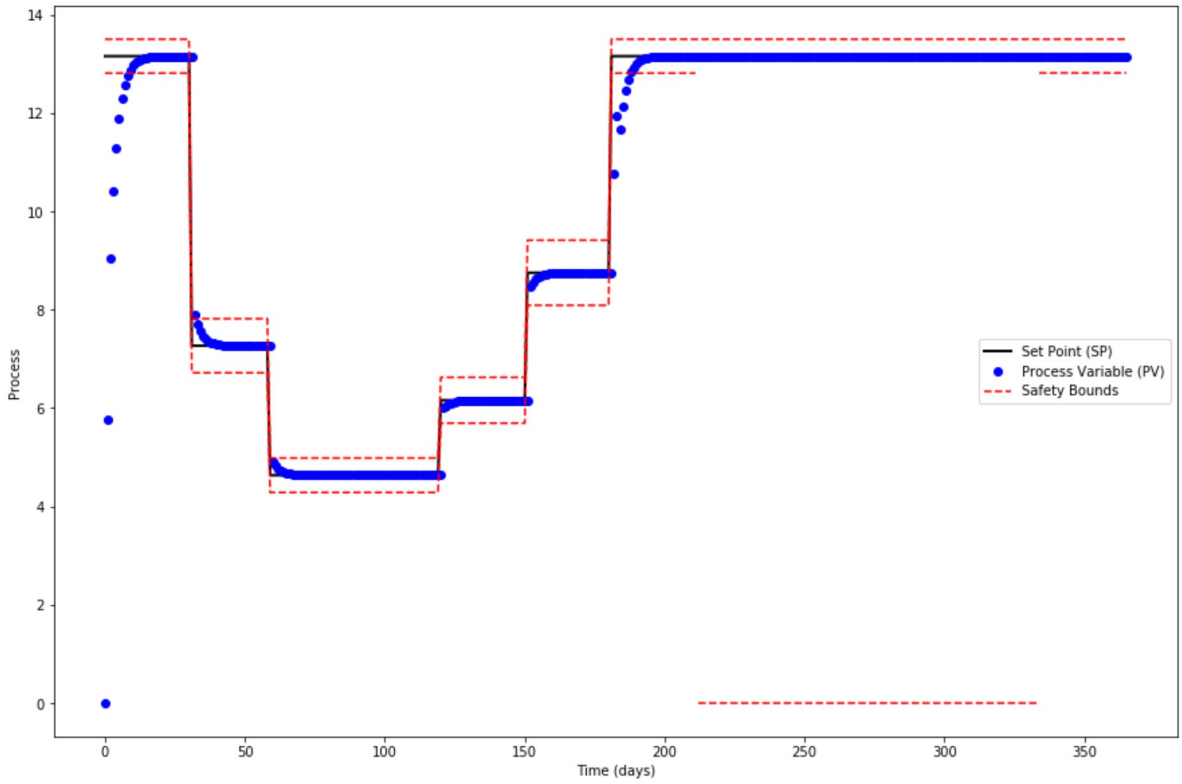
```
Kc = 1.8
tauI = 1.8
tauD = 0.0
```

In [8]:

```
# storage for recording values
op = np.zeros(ns+1)  # controller output
pv = np.zeros(ns+1)  # process variable
e = np.zeros(ns+1)   # error
ie = np.zeros(ns+1)  # integral of the error
dpv = np.zeros(ns+1) # derivative of the pv
P = np.zeros(ns+1)   # proportional
I = np.zeros(ns+1)   # integral
D = np.zeros(ns+1)   # derivative
sp = setPoint  # set point

# PID (starting point)
Kc = 1.8
tauI = 1.8
tauD = 0.0

# PID (tuning)
Kc = Kc * 0.75
tauI = tauI / 1.0
tauD = 0.0

# Upper and Lower limits on OP
op_hi = 13.5
op_lo = 0.0

# loop through time steps
for i in range(0,ns):
    e[i] = sp[i] - pv[i]
    if i >= 1:  # calculate starting on second cycle
        dpv[i] = (pv[i]-pv[i-1])/delta_t
        ie[i] = ie[i-1] + e[i] * delta_t
    P[i] = Kc * e[i]
    I[i] = Kc/tauI * ie[i]
    D[i] = - Kc * tauD * dpv[i]
    op[i] = op[0] + P[i] + I[i] + D[i]
    if op[i] > op_hi:  # check upper limit
        op[i] = op_hi
        ie[i] = ie[i] - e[i] * delta_t # anti-reset windup
    if op[i] < op_lo:  # check lower limit
        op[i] = op_lo
        ie[i] = ie[i] - e[i] * delta_t # anti-reset windup
    y = odeint(process,pv[i],[0,delta_t],args=(op[i],Kp,taup))
    pv[i+1] = y[-1]
op[ns] = op[ns-1]
ie[ns] = ie[ns-1]
P[ns] = P[ns-1]
I[ns] = I[ns-1]
D[ns] = D[ns-1]

# plot results
plt.figure(1)
plt.figure(figsize=(15,10))
```

```
plt.plot(t,sp,'k-',linewidth=2,label='Set Point (SP)')
plt.plot(t,pv,'bo',linewidth=2, label='Process Variable (PV)')
plt.ylabel('Process')
#plt.ylim([-0.1,12])
plt.xlabel('Time (days)')

plt.plot(ts,TopMax,'--r',label='Safety Bounds')
plt.plot(ts[0:212],TopLow[0:212],'--r')
plt.plot(ts[212:334],BotLow[212:334],'--r')
plt.plot(ts[334:ndays+1],TopLow[334:ndays+1],'--r')
plt.legend(loc='best')
plt.savefig('projectPID(untuned).png')
```

```
<Figure size 432x288 with 0 Axes>
```



While tuning we had to reduce the gain because of oscillations that occurred and caused the system to occasionally spend excess time outside the safety zone in more areas than the large jump at t = 212 days.

In [ ]: