Problem 1:

In [ ]:
```
# importing libraries
from sympy import *
import numpy as np
import math
import matplotlib.pyplot as plt
```

The probability density function(PDF):

In [ ]:
```
t=symbols('t')
λ = symbols('λ')

pdf = (2*t)*exp((-(t**2))/(λ**2))/λ**2
pdf
```

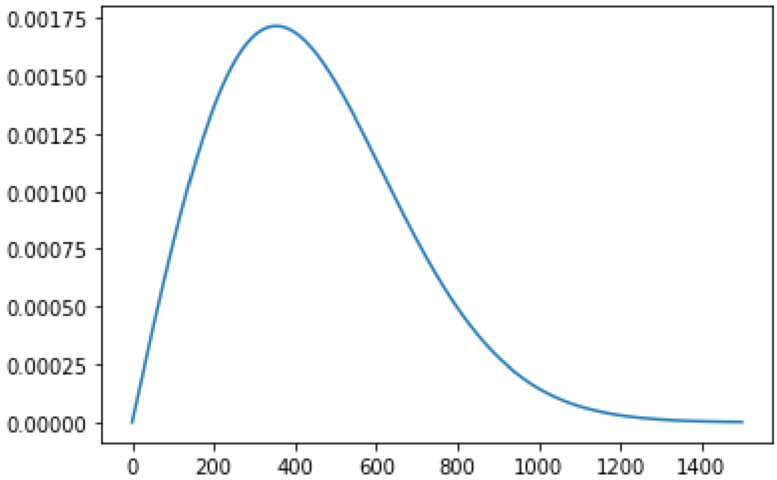
Out[ ]:

$$\frac{2te^{-\frac{t^2}{\lambda^2}}}{\lambda^2}$$

In [ ]:
```
λ = 500 # λ is a parameter, with specifically λ = 500days.
pdf = (2*t)*exp((-(t**2))/(λ**2))/λ**2 # with only t as a variable.
```

The PDF is plotted below:

In [ ]:
```
def PDF(t, λ = λ):

    if t >= 0:
        return (2*t)*np.exp((-(t**2))/(λ**2))/λ**2
    else:
        return 0

arr = np.array([])
for t0 in range(1500):
    arr = np.append(arr,PDF(t0))

plt.plot(arr);
```



a. Confirming that this is a valid PDF:

In [ ]:
```
# checking if it is non-negative everywhere:
print(f'Number of Nigative Values = {(arr<0).sum()}')
```

```
Number of Nigative Values = 0
```

In [ ]:
```
# checking if the area under the curve equals 1:
print("area under the curve =", integrate(pdf, [t,0,oo]))
```

```
area under the curve = 1
```

b. How long after installation should we do preventative maintenance if we wish to have the probability of unexpected failure be less than 1%, 10%, 50%, and 99%?

In [ ]:
```
def time_after_installation(target):

    for idxt, _ in enumerate(arr):

        narr = arr[:idxt]

        if np.trapz(narr) >= target:

            return idxt-1

days = []

for p in [0.01,0.1,0.5,0.99]:

    ndays = time_after_installation(p)
    days.append(ndays)
    print(f'To have a prbability of unexpected failure be less than {int(p*100)}%,
```

```
To have a prbability of unexpected failure be less than 1%, you should do preventa
tive maintenance at day: 51
To have a prbability of unexpected failure be less than 10%, you should do prevent
ative maintenance at day: 163
To have a prbability of unexpected failure be less than 50%, you should do prevent
ative maintenance at day: 417
To have a prbability of unexpected failure be less than 99%, you should do prevent
ative maintenance at day: 1073
```

c. What is the expected lifetime for this pump? What is the probability of failure before the expected lifetime?

In [ ]:
```
elt = integrate(t*pdf, [t,0,oo])
print(f'The expected lifetime is: {round(elt)} days')
```

```
The expected lifetime is: 443 days
```

In [ ]:
```
pf = round(integrate(pdf,[t,0,elt]),4)
print(f'The probability of failure before the expected lifetime is: {pf*100}%')
```

```
The probability of failure before the expected lifetime is: 54.41%
```

d. What is the variance of the pump’s lifetime? What is the range of the lifetime that falls within

one standard deviation of the expected value?

In [ ]:
```
vlt = integrate(pdf*( t  -elt )**2 , [t,0, oo])
print(f'The variance of the pump’s lifetime is: {round(vlt,2)}')
print(f'The standard deviation of the pump’s lifetime is: {round(sqrt(vlt),2)}')
```

```
The variance of the pump’s lifetime is: 53650.46
The standard deviation of the pump’s lifetime is: 231.63
```

Getting the CDF before writing a program that generates samples of t from its distribution:

In [ ]:
```
cdf = integrate(pdf,[t, 0,t])
cdf
```

Out[ ]:

$$1 - e^{-\frac{t^2}{250000}}$$

e. Write a program that generates samples of t from its distribution.

In [ ]:
```
def avg_rnng_cst(Tm ,n_samples = 10**6, Cr = 250, Cm= 50):

    smpl_arr = rnng_cst_arr = np.zeros((n_samples,))

    for s in range(n_samples):

        unfrm = np.random.uniform()
        smpl_arr[s] = math.log(-1/(unfrm-1))*1000

        if smpl_arr[s] <= Tm:
            rnng_cst_arr[s] = Cr/smpl_arr[s]
        else:
            rnng_cst_arr[s] = Cm/Tm

    avg_R = round(np.mean(rnng_cst_arr),4)
    avg_smpl = round(np.mean(smpl_arr),4)
    var_smpl = round(np.var(smpl_arr),4)

    print(f'Average cost for Tm = {Tm} was: {avg_R}$')
    print(f'Sample average was: {avg_smpl}')
    print(f'Sample variance was: {var_smpl}')
```

In [ ]:
```
for Tm in [1,10,100,1000,10000]:
    avg_rnng_cst(Tm)
    print("-"*44)
```

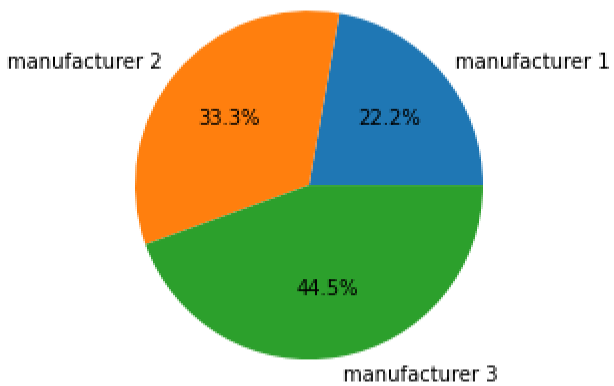
```
Average cost for Tm = 1 was: 51.71$
Sample average was: 51.71
Sample variance was: 83929.3115
--------------------------------------------
Average cost for Tm = 10 was: 8.7695$
Sample average was: 8.7695
Sample variance was: 2649397.2758
--------------------------------------------
Average cost for Tm = 100 was: 3.2955$
Sample average was: 3.2955
Sample variance was: 51276.5657
--------------------------------------------
Average cost for Tm = 1000 was: 4.1171$
Sample average was: 4.1171
Sample variance was: 601540.5287
--------------------------------------------
Average cost for Tm = 10000 was: 17.8046$
Sample average was: 17.8046
Sample variance was: 196437751.6095
--------------------------------------------
```

Problem 2:

In [ ]:
```
def transformer_prob_of_out_of_spec(n_transformers, prob_of_out_of_spec):
    pos = 0
    for t in range(1, n_transformers+1):
        pos += prob_of_out_of_spec**t
    return pos
```

In [ ]:
```
m1 = transformer_prob_of_out_of_spec(8, 2/1000)
m2 = transformer_prob_of_out_of_spec(5, 3/1000)
m3 = transformer_prob_of_out_of_spec(20, 4/1000)
```

In [ ]:
```
ms = np.array([m1,m2,m3])
mylabels = ["manufacturer 1", "manufacturer 2", "manufacturer 3"]

plt.pie(ms, labels = mylabels, autopct='%1.1f%%')
plt.show()
```



The probability that it is from manufacturer 1 is 22.2%

Problem 4:

In [ ]:
```
# the Kalman filter for 10 steps.
```

```
n_steps = 10
# At time k = 0, E[r(0)] = 20...
Er0 = 20
# with uncertainty Var[r(0)] = 25.
uncertainty_Vr0 = 25
# d(k) = 10 for all k ≥ 0.
dk = 10
# The consumer is predicted to use a supply m = 7
m = 7
# since d(k) = 10 for all k ≥ 0 and The consumer is predicted to use a supply m = 7
# then uf will always be:
uf = dk - m
# process uncertainty is Var [v(k)] = 9.
process_uncertainty = 9
# sensor uncertainty is Var [w(k)] = 25.
sensor_uncertainty = 25
# We receive the following sequence of measurements z(k):
measurements = [17.8, 22.6, 30.2, 37.3, 46.2, 49.5, 44.6, 50.3, 56.3, 51.6]
```

In [ ]:

```
water_volume = np.zeros([n_steps+1]) # to store the actual volume of water.
water_volume[0] = Er0 # since E[r(0)] = 20 at time k = 0.

uncertainty = np.zeros([n_steps+1]) # to provide the associated uncertainty for my
uncertainty[0] = uncertainty_Vr0 # since Var[r(0)] = 25 at time k = 0.
```

In [ ]:

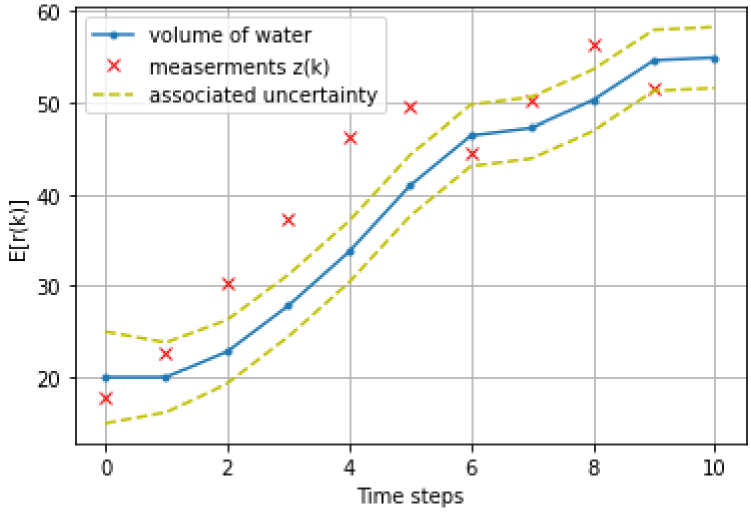
```
d = np.mat([[1]]) # dynamic

mm = np.mat([[1]]) # measurement model

mnv = np.mat([[25]]) # measurement noise variance
```

In [ ]:

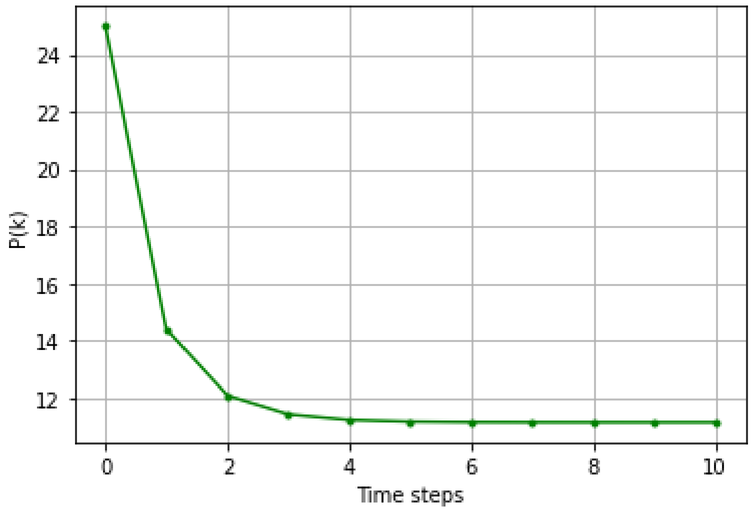
```
for k in range(1,n_steps+1):

    Kp = uf + d*Er0
    K_uncertainty = sensor_uncertainty * (d**2) + process_uncertainty

    measurement = measurements[k-1]

    K = K_uncertainty @ mm.T @ np.linalg.inv(mm @ K_uncertainty @ mm.T + mnv)
    Er0 = Kp + K @ (measurement - mm @ Kp)
    sensor_uncertainty = (np.eye(1) - K @ mm) @ K_uncertainty @ (np.eye(1) - K @ mm

    water_volume[k] = Er0[0]

    uncertainty[k] = sensor_uncertainty[0]
```

In [ ]:

```
plt.plot(water_volume,'.-',label="volume of water")
plt.plot(measurements,'rx',label="measerments z(k)")
plt.plot(water_volume+np.sqrt(uncertainty),'y--',label="associated uncertainty")
plt.plot(water_volume-np.sqrt(uncertainty),'y--')
plt.ylabel('E[r(k)]')
plt.xlabel('Time steps')
plt.legend()
plt.grid(True)
```



In [ ]:
```
plt.plot(uncertainty,'g.-')
plt.ylabel('P(k)')
plt.xlabel('Time steps')
plt.grid(True)
```



In [ ]:
```
d = np.mat([[1,-1],[0,1]]) # dynamics

mm = np.mat([[1,0]]) # measurement model

mnv = np.mat([[25]]) # measurement noise variance

ii = np.eye(2)
```

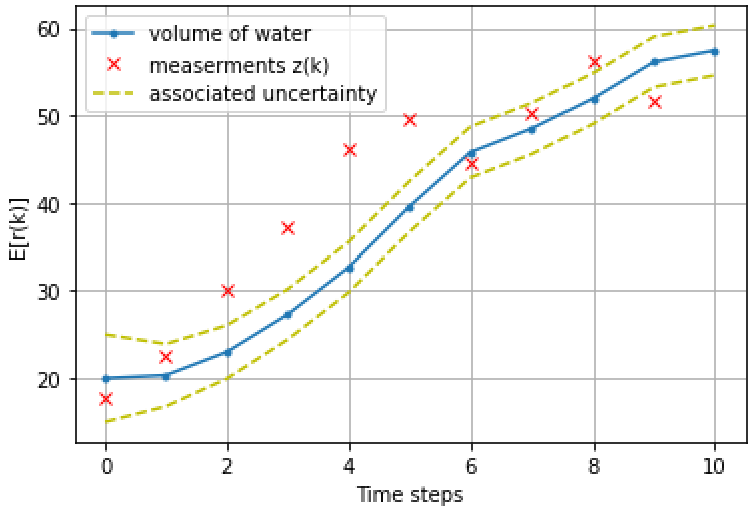
In [ ]:
```
# At time k = 0, E[r(0)] = 20, E[c(0)] = 7...
Er0 = np.mat([[20],[7]])

uf = np.mat([[10],[0]])
# Var [n(k)] = 0.1
process_uncertainty = np.mat([[0,0],[0,0.1]])
# sensor uncertainty is Var [w(k)] = 25, Var [c(0)] = 1
sensor_uncertainty = np.mat([[25,0],[0,1]])
```

In [ ]:
```
water_volume = np.zeros([n_steps+1,2])

water_volume[0,0] = Er0[0,0]
water_volume[0,1] = Er0[1,0]

uncertainty = np.zeros([n_steps+1,2])
```

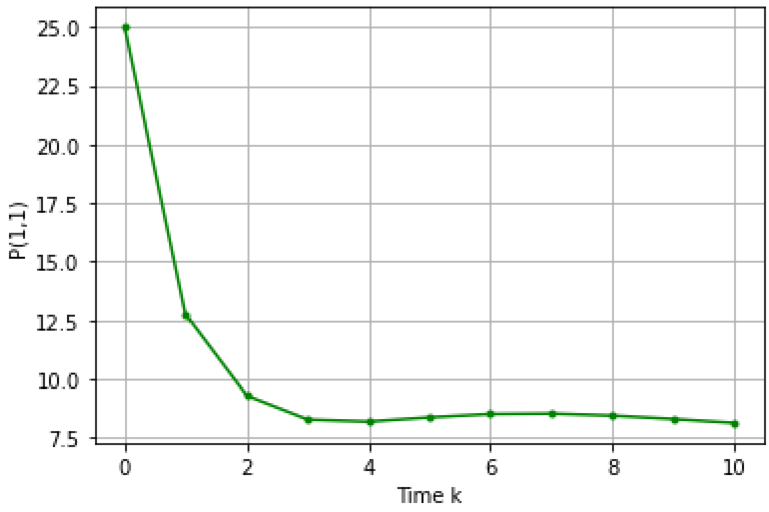
```
uncertainty[0,0] = sensor_uncertainty[0,0]
uncertainty[0,1] = sensor_uncertainty[1,1]
```

In [ ]:
```
for k in range(1,n_steps+1):

    # Kalman filter prediction:
    Kp = d @ Er0 + uf
    # Kalman filter prediction uncertainty:
    K_uncertainty = d @ sensor_uncertainty @ d.T + process_uncertainty

    measurement = measurements[k-1]

    K = K_uncertainty @ mm.T @ np.linalg.inv(mm @ K_uncertainty @ mm.T + mnv)
    Er0 = Kp + K @ (measurement - mm @ Kp)
    sensor_uncertainty = (ii-K @ mm)@ K_uncertainty @(ii-K @ mm).T + K @ mnv @ K.T

    # store the variables for plotting:
    water_volume[k,0] = Er0[0,0]
    water_volume[k,1] = Er0[1,0]

    uncertainty[k,0] = sensor_uncertainty[0,0]
    uncertainty[k,1] = sensor_uncertainty[1,1]
```

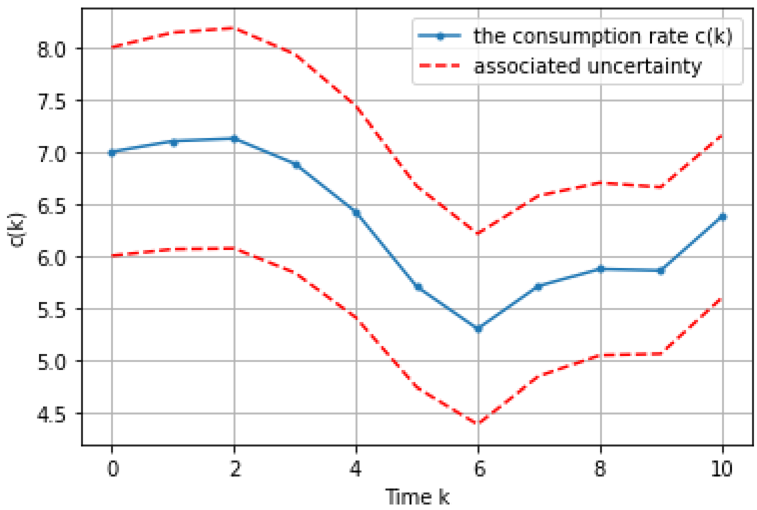
In [ ]:
```
plt.plot(water_volume[:,0],'.-',label="volume of water")
plt.plot(measurements,'rx',label="measerments z(k)")
plt.plot(water_volume[:,0]+np.sqrt(uncertainty[:,0]),'y--',label="associated uncer
plt.plot(water_volume[:,0]-np.sqrt(uncertainty[:,0]),'y--')
plt.ylabel('E[r(k)]')
plt.xlabel('Time steps')
plt.legend()
plt.grid(True)
```



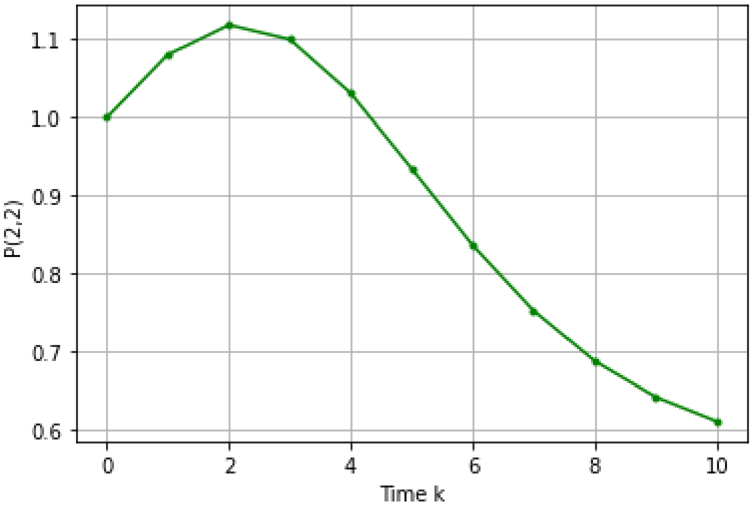
In [ ]:
```
plt.plot(uncertainty[:,0],'g.-',label="P(k)(1,1)")
plt.xlabel('Time k')
plt.ylabel('P(1,1)')
plt.grid(True)
```



In [ ]:
```
plt.plot(water_volume[:,1],'.-',label="the consumption rate c(k)")
plt.plot(water_volume[:,1]+np.sqrt(uncertainty[:,1]),'r--',label="associated uncer
plt.plot(water_volume[:,1]-np.sqrt(uncertainty[:,1]),'r--')

plt.xlabel('Time k')
plt.ylabel('c(k)')
plt.grid(True)
plt.legend();
```



In [ ]:
```
plt.plot(uncertainty[:,1],'g.-',label="P(k)(2,2)")
plt.xlabel('Time k')
plt.ylabel('P(2,2)')
plt.grid(True)
```



In [ ]:
```
α = 0.3
d = np.matrix([[1-2*α, α, α, 0, -1, 0, 0, 0],
               [α, 1-2*α, α, 0, 0, -1, 0, 0],
               [α, α, 1-3*α, α, 0, 0, -1, 0],
               [0, 0, α, 1-α, 0, 0, 0, -1],
               [0, 0, 0, 0, 1, 0, 0, 0],
               [0, 0, 0, 0, 0, 1, 0, 0],
               [0, 0, 0, 0, 0, 0, 1, 0],
               [0,0,0,0,0,0,0,1]])

process_uncertainty=np.diag([0,0,0,0,0.1,0.1,0.1,0.1])

mm = np.eye(4,8)

mnv = np.eye(4)*25
```

In [ ]:
```
z1 = np.array([59.3, 72, 64.4, 83.6, 84.9, 94.3, 84, 86.6, 89, 89.1])
z2 = np.array([39.1, 38.4, 36.2, 43.4, 50.5, 56.3, 40.3, 58.5, 55.4, 59.6])
z3 = np.array([31.1, 31.2, 41.6, 44.4, 41, 41.9, 39.2, 46.3, 43.3, 45.3])
z4 = np.array([38.6, 38, 32.6, 18, 29.4, 23.3, 11, 14.6, 18.4, 20.5])

list_of_measurements = [z1,z2,z3,z4]
measurements = np.mat(list_of_measurements)
```

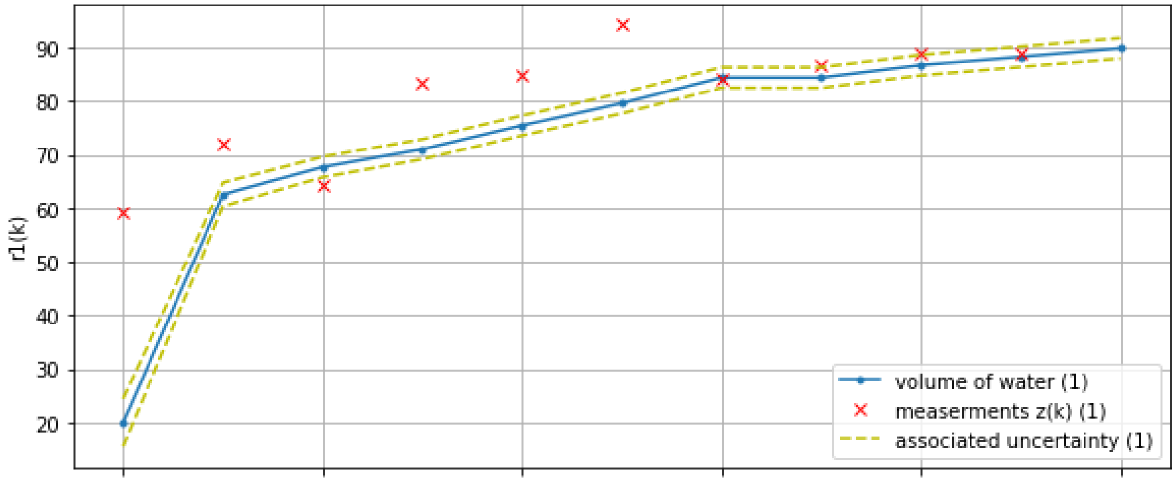
In [ ]:
```
Er0 = np.mat([[20],[40],[60],[20],[7],[7],[7],[7]])

sensor_uncertainty= np.diag([20,20,20,20,1,1,1,1])

uf = [[30],[0],[0],[0],[0],[0],[0],[0]]
```

In [ ]:
```
water_volume = np.zeros([n_steps+1,8])

uncertainty = np.zeros([n_steps+1,8])

for i in range(8):
    water_volume[0,i] = Er0[i,0]
    uncertainty[0,i] = sensor_uncertainty[i,i]
```

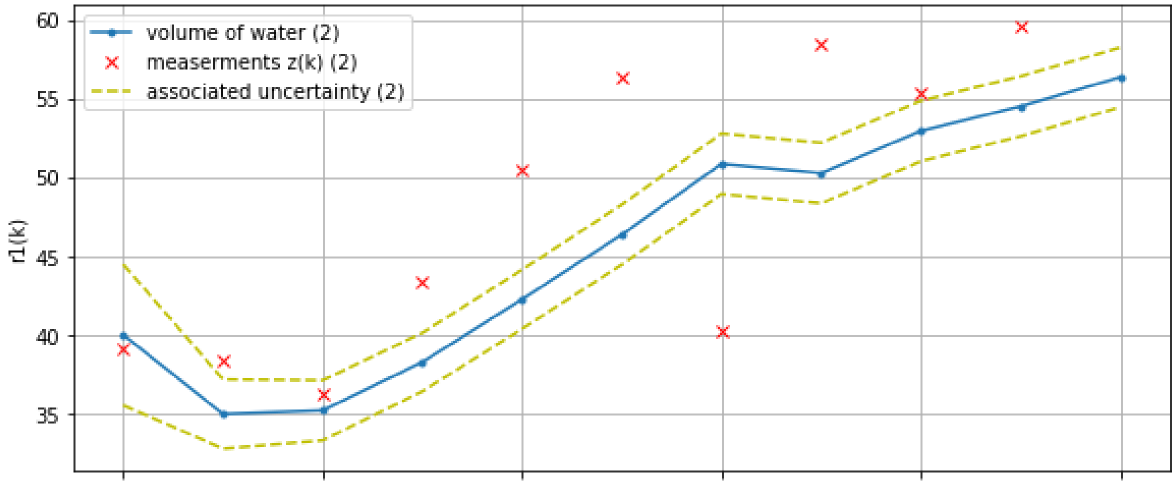
In [ ]:
```
for k in np.arange(1,n_steps+1):

    Kp = d @ Er0 + uf
    K_uncertainty = d @ sensor_uncertainty @ d.T + process_uncertainty

    measurement = measurements[:,k-1]
```

```
    K = K_uncertainty @ mm.T @ np.linalg.inv(mm @ K_uncertainty @ mm.T + mnv)
    Er0 = Kp + K @ (measurement - mm @ Kp)
    sensor_uncertainty = (np.eye(8) - K @ mm) @ K_uncertainty @ (np.eye(8) - K @ mm

    for i in range(8):
        water_volume[k,i] = Er0[i,0]
        uncertainty[k,i] = sensor_uncertainty[i,i]
```

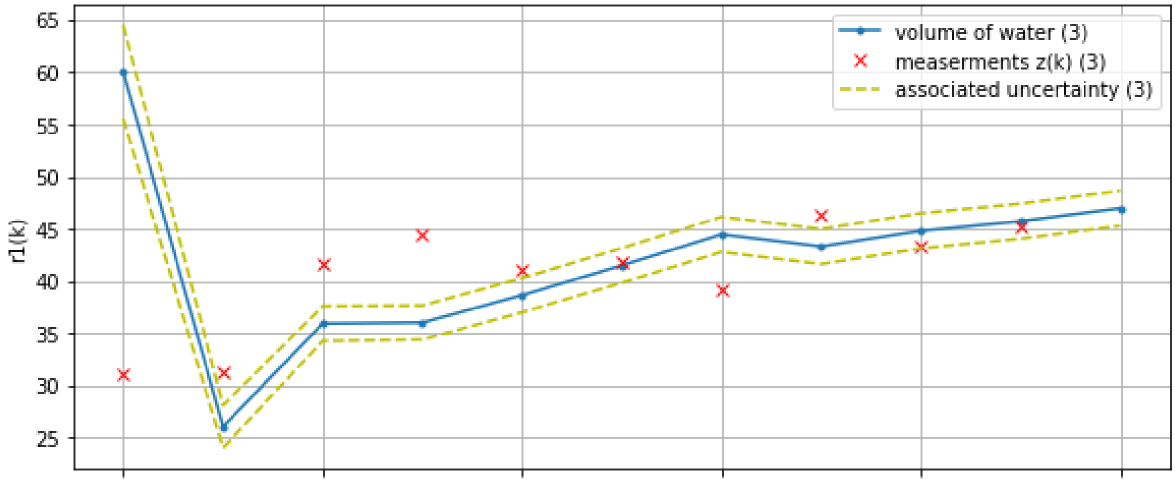
In [ ]:
```
fig, ax = plt.subplots(4,1,sharex=True)
fig.set_size_inches(10,20)

for n in range(4):

    ax[n].plot(water_volume[:,n],'.-',label=f"volume of water ({n+1})")
    ax[n].plot(list_of_measurements[n],'rx',label=f"measerments z(k) ({n+1})")
    ax[n].plot(water_volume[:,n]+np.sqrt(uncertainty[:,n]),'y--',label=f"associated
    ax[n].plot(water_volume[:,n]-np.sqrt(uncertainty[:,n]),'y--',)

    ax[n].set_ylabel('r1(k)')
    ax[n].legend()
    ax[n].grid(True)
```



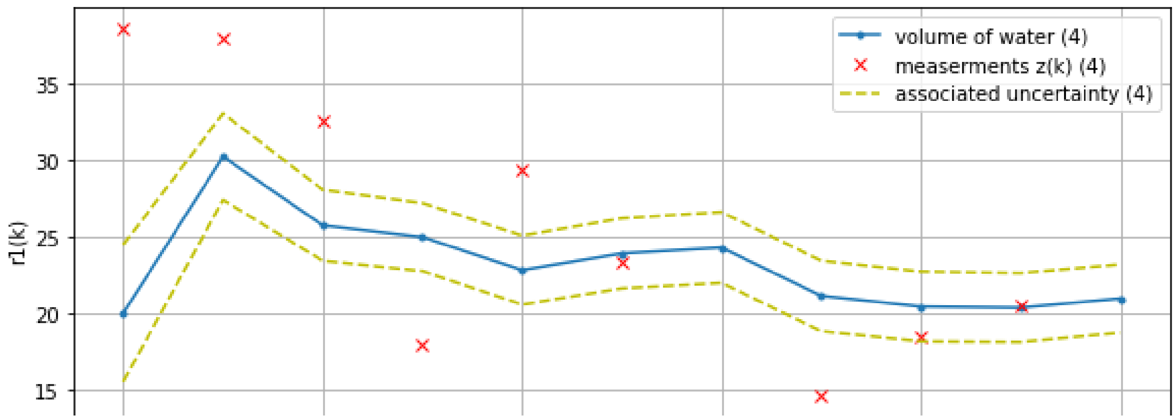
90
80
70
60
50
40
30
20
r1(k)
volume of water (1)
measerments z(k) (1)
associated uncertainty (1)



60
55
50
45
40
35
r1(k)
volume of water (2)
measerments z(k) (2)
associated uncertainty (2)

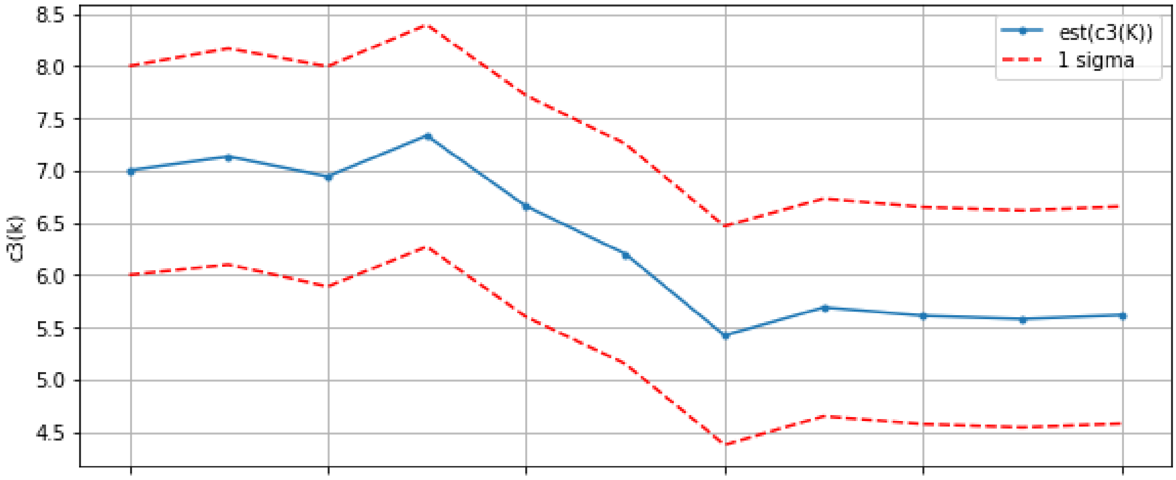


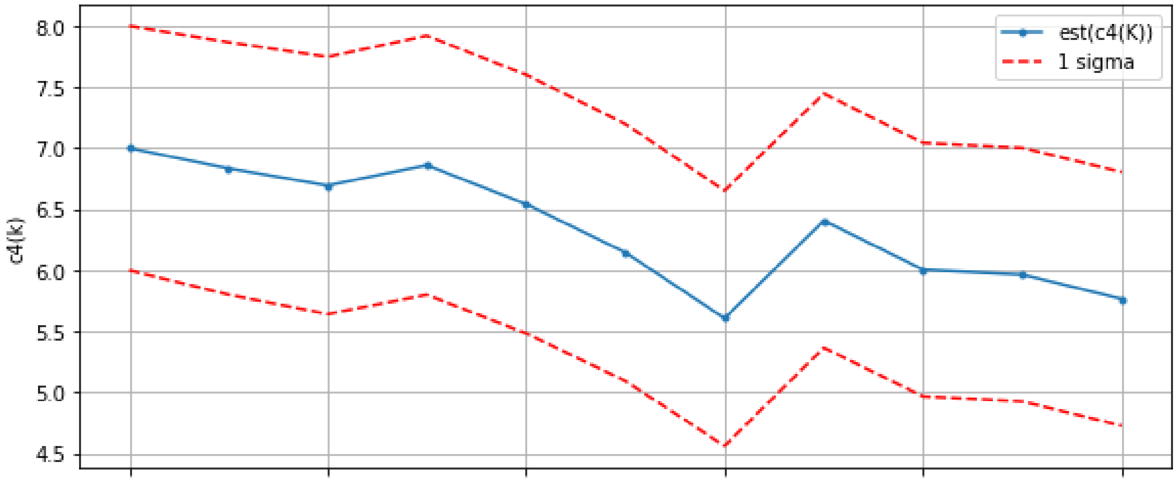
65
60
55
50
45
40
35
30
25
r1(k)
volume of water (3)
measerments z(k) (3)
associated uncertainty (3)

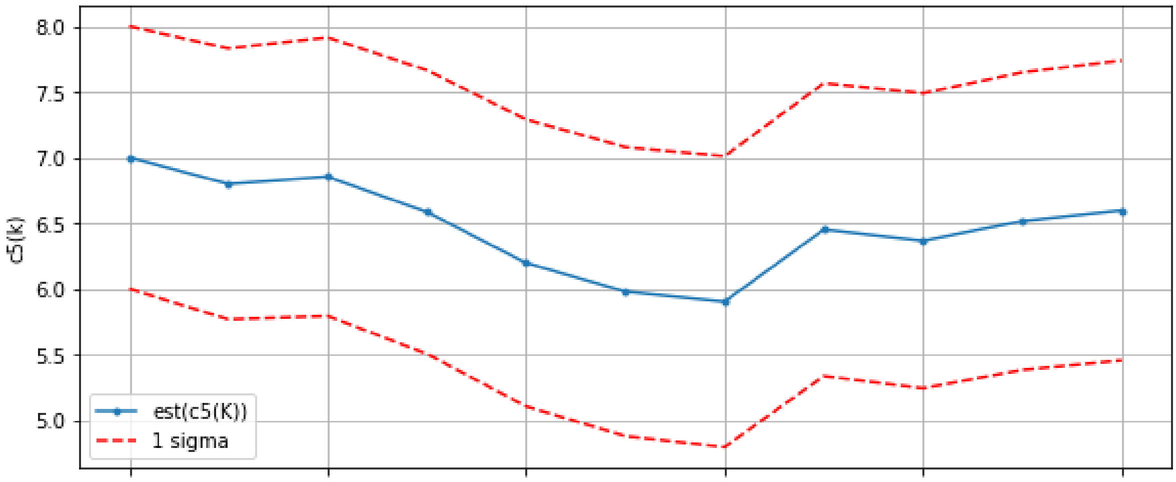


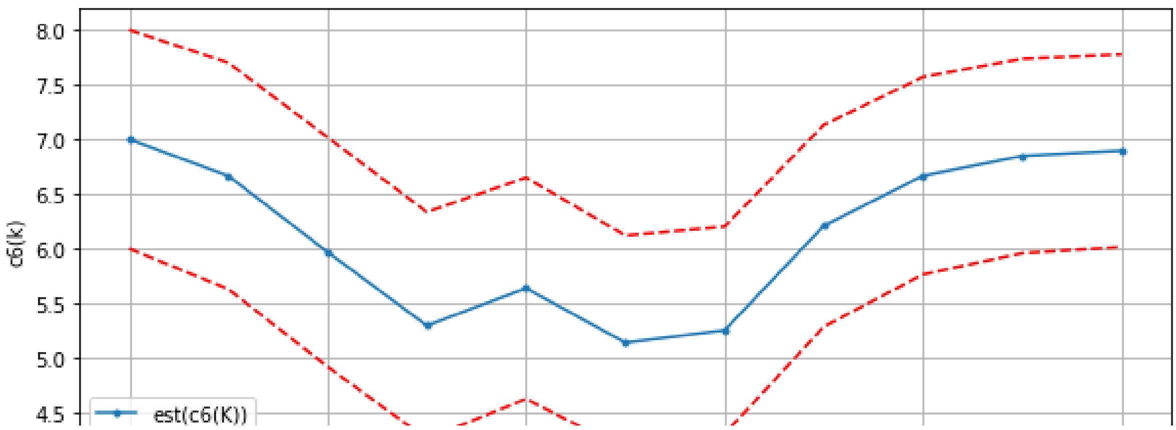
35
30
25
20
15
r1(k)
volume of water (4)
measerments z(k) (4)
associated uncertainty (4)

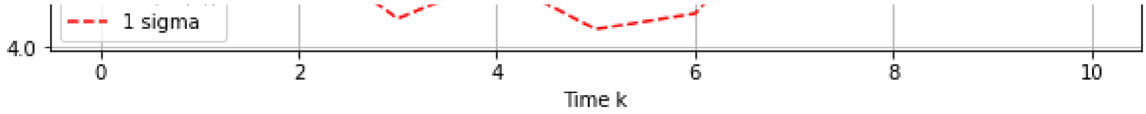
In [ ]:
```
fig, ax = plt.subplots(4,1,sharex=True)
fig.set_size_inches(10,20)

for n in range(4):

    ax[n].plot(water_volume[:,n+4],'.-',label=f"est(c{n+3}(K))")
    ax[n].plot(water_volume[:,n+4]+np.sqrt(uncertainty[:,n+4]),'r--',label="1 sigma
    ax[n].plot(water_volume[:,n+4]-np.sqrt(uncertainty[:,n+4]),'r--',)

    ax[3].set_xlabel('Time k')
    ax[n].set_ylabel(f'c{n+3}(k)')
    ax[n].legend()
    ax[n].grid(True)
```


c3(k)
8.5
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
est(c3(K))
1 sigma


c4(k)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
est(c4(K))
1 sigma

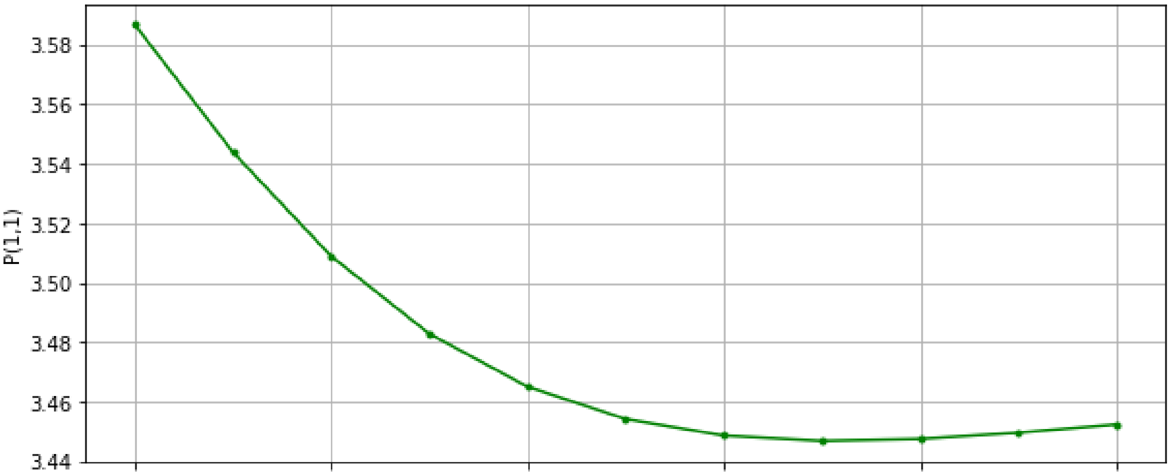

c5(k)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
est(c5(K))
1 sigma

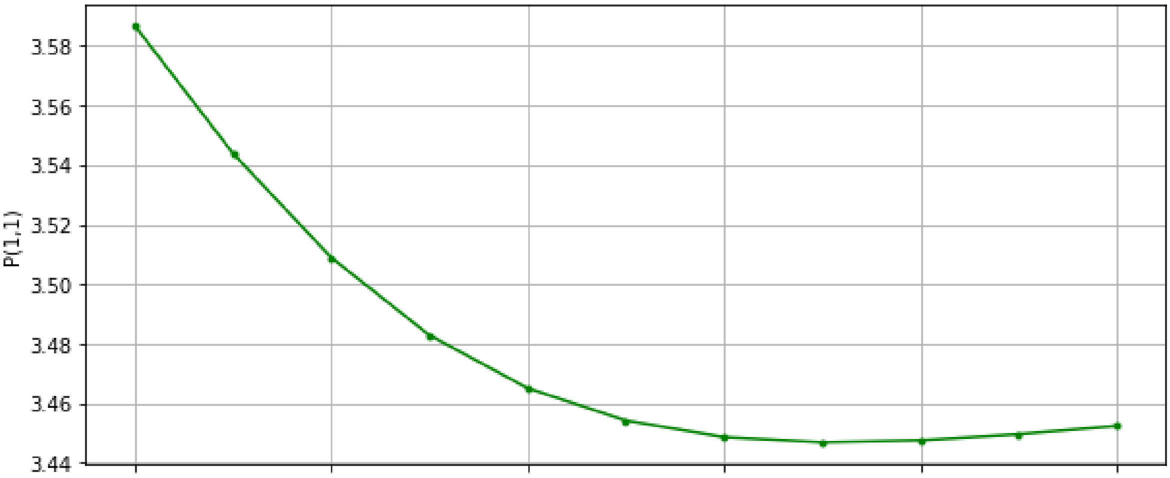

c6(k)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
est(c6(K))

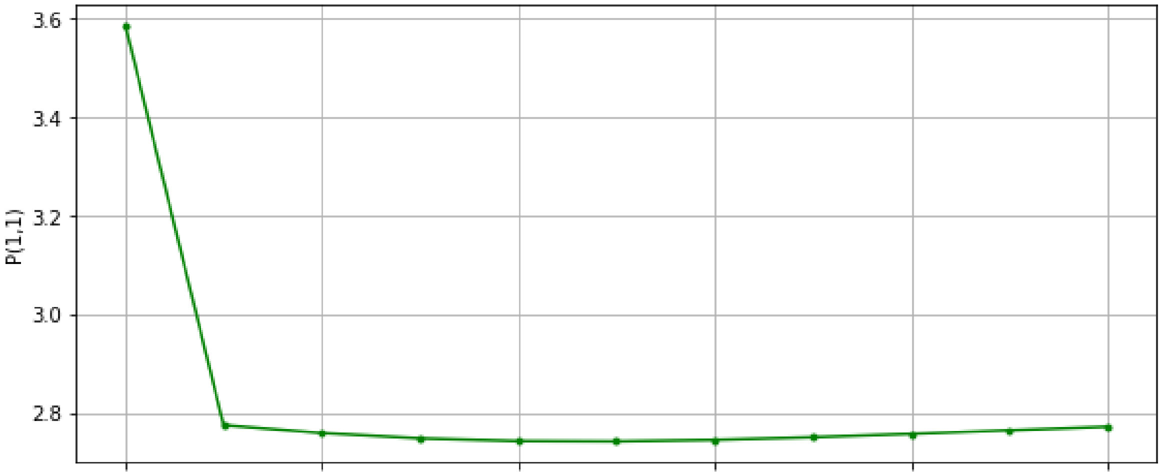


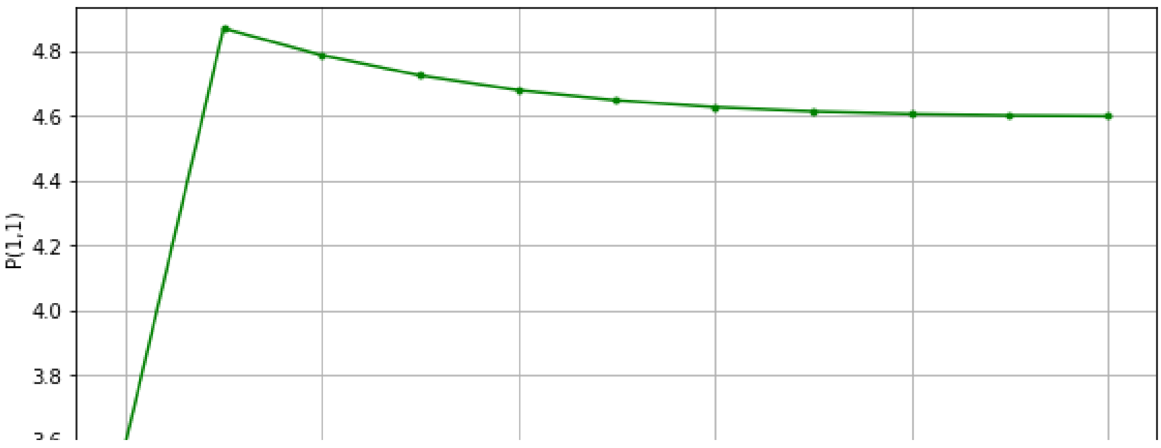
In [ ]:
```
uncertainties = [np.zeros([n_steps+1,1]),np.zeros([n_steps+1,1]),np.zeros([n_steps

for i in range(4):

    uncertainties[i][0,0] = sensor_uncertainty[0,0]

for s in np.arange(1,n_steps+1):

    K_uncertainty = d @ sensor_uncertainty @ d.T + process_uncertainty

    K = K_uncertainty @ mm.T @ np.linalg.inv(mm @ K_uncertainty @ mm.T + mnv)
    sensor_uncertainty = (np.eye(8) - K @ mm) @ K_uncertainty @ (np.eye(8) - K @ mm

    for i in range(4):
        uncertainties[i][s,0] = sensor_uncertainty[i,i]
```

In [ ]:
```
fig, ax = plt.subplots(4,1,sharex=True)
fig.set_size_inches(10,20)

for n in range(4):

    ax[n].plot(uncertainties[n][:,0],'g.-',label="P(1,1)")
    ax[n].set_ylabel('P(1,1)')
    ax[n].grid(True)
```


P(1,1)
3.58
3.56
3.54
3.52
3.50
3.48
3.46
3.44


P(1,1)
3.58
3.56
3.54
3.52
3.50
3.48
3.46
3.44


P(1,1)
3.6
3.4
3.2
3.0
2.8


P(1,1)
4.8
4.6
4.4
4.2
4.0
3.8

In [ ]:
```
for i in range(4):
    print(1+i,':')
    print(uncertainties[i][:,0])
```

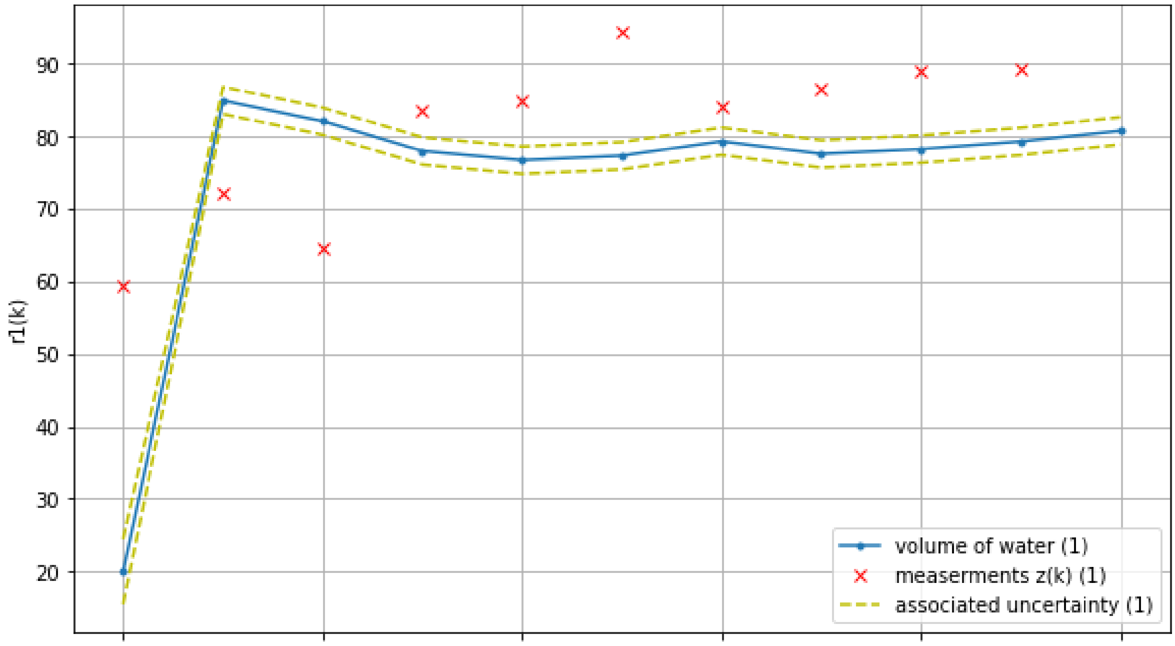
```
1 :
[3.58678078 3.54391255 3.50896257 3.4828954  3.46511245 3.45425204
 3.44870439 3.4469224  3.44758705 3.44967182 3.45244104]
2 :
[3.58678078 3.54391255 3.50896257 3.4828954  3.46511245 3.45425204
 3.44870439 3.4469224  3.44758705 3.44967182 3.45244104]
3 :
[3.58678078 2.77486303 2.75857296 2.74780805 2.74249816 2.74184656
 2.74474987 2.75007506 2.75682296 2.76420467 2.77165648]
4 :
[3.58678078 4.86927961 4.78708804 4.72452577 4.67915383 4.64770437
 4.62687624 4.61374186 4.60591418 4.60157046 4.59939703]
```

In [ ]:
```
mm = np.eye(3,8) # measurement model

mnv = np.eye(3)*25 # measurement noise variance
```

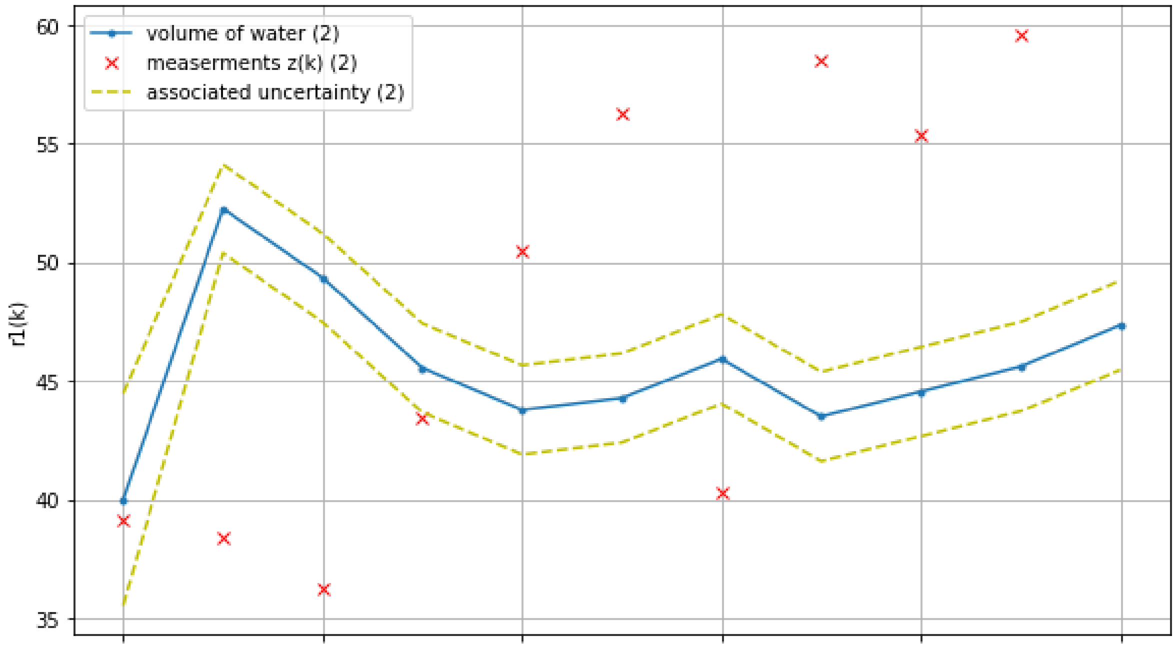
In [ ]:
```
list_of_measurements = [z1,z2,z4]
measurements = np.mat(list_of_measurements)
```

In [ ]:
```
for k in np.arange(1,n_steps+1):

    Kp = d @ Er0 + uf
    K_uncertainty = d @ sensor_uncertainty @ d.T + process_uncertainty

    measurement = measurements[:,k-1]

    K = K_uncertainty @ mm.T @ np.linalg.inv(mm @ K_uncertainty @ mm.T + mnv)
    Er0 = Kp + K @ (measurement - mm @ Kp)
    sensor_uncertainty = (np.eye(8) - K @ mm) @ K_uncertainty @ (np.eye(8) - K @ mm

    for i in range(8):
        water_volume[k,i] = Er0[i,0]
        uncertainty[k,i] = sensor_uncertainty[i,i]
```

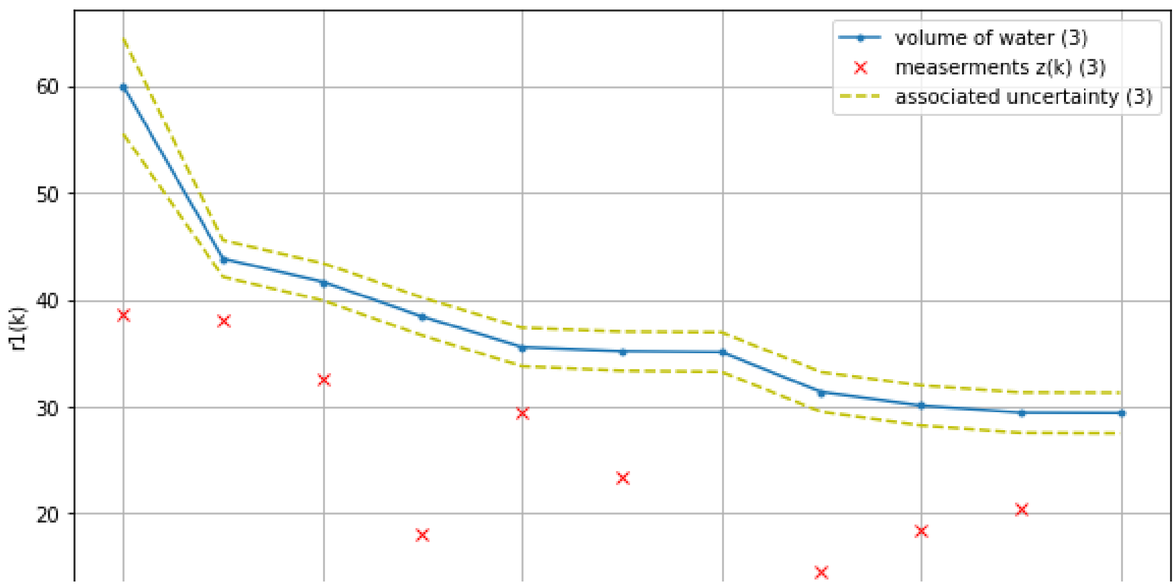
In [ ]:
```
fig, ax = plt.subplots(3,1,sharex=True)
fig.set_size_inches(10,20)

for n in range(3):

    ax[n].plot(water_volume[:,n],'.-',label=f"volume of water ({n+1})")
    ax[n].plot(list_of_measurements[n],'rx',label=f"measerments z(k) ({n+1})")
    ax[n].plot(water_volume[:,n]+np.sqrt(uncertainty[:,n]),'y--',label=f"associate
    ax[n].plot(water_volume[:,n]-np.sqrt(uncertainty[:,n]),'y--',)

    ax[n].set_ylabel('r1(k)')
    ax[n].legend()
    ax[n].grid(True)
```



r1(k)
90
80
70
60
50
40
30
20
volume of water (1)
measerments z(k) (1)
associated uncertainty (1)

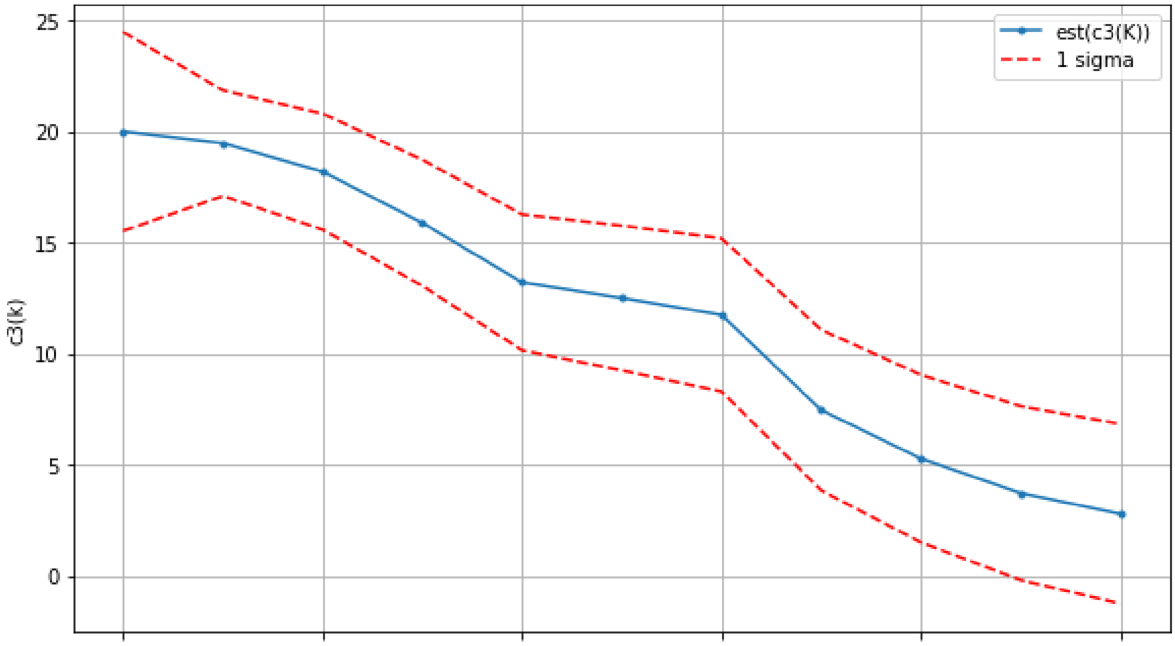


r1(k)
60
55
50
45
40
35
volume of water (2)
measerments z(k) (2)
associated uncertainty (2)

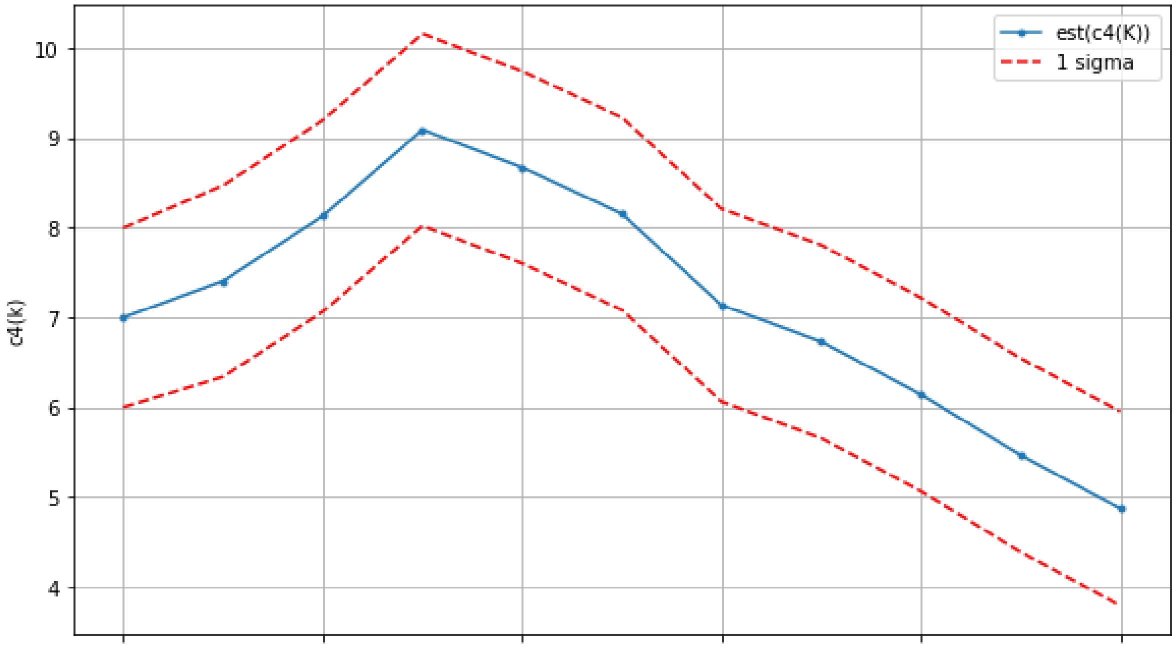


r1(k)
60
50
40
30
20
volume of water (3)
measerments z(k) (3)
associated uncertainty (3)

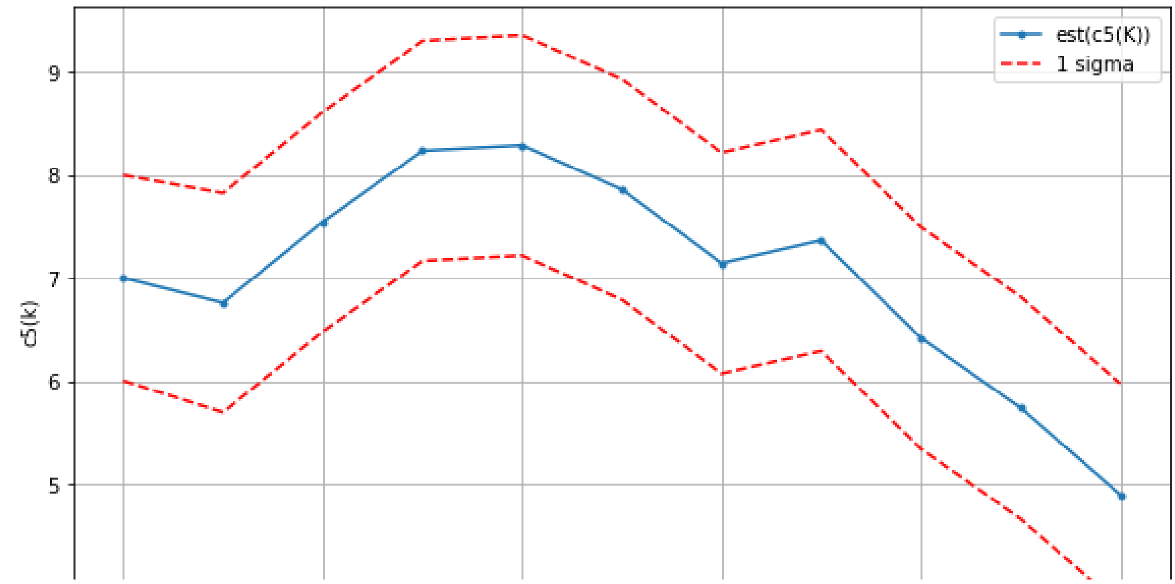
In [ ]:
```
fig, ax = plt.subplots(3,1,sharex=True)
fig.set_size_inches(10,20)

for n in range(3):

    ax[n].plot(water_volume[:,n+3],'.-',label=f"est(c{n+3}(K))")
    ax[n].plot(water_volume[:,n+3]+np.sqrt(uncertainty[:,n+3]),'r--',label="1 sigma
    ax[n].plot(water_volume[:,n+3]-np.sqrt(uncertainty[:,n+3]),'r--',)

    ax[2].set_xlabel('Time k')
    ax[n].set_ylabel(f'c{n+3}(k)')
    ax[n].legend()
    ax[n].grid(True)
```



est(c3(K))
1 sigma
c3(k)
25
20
15
10
5
0



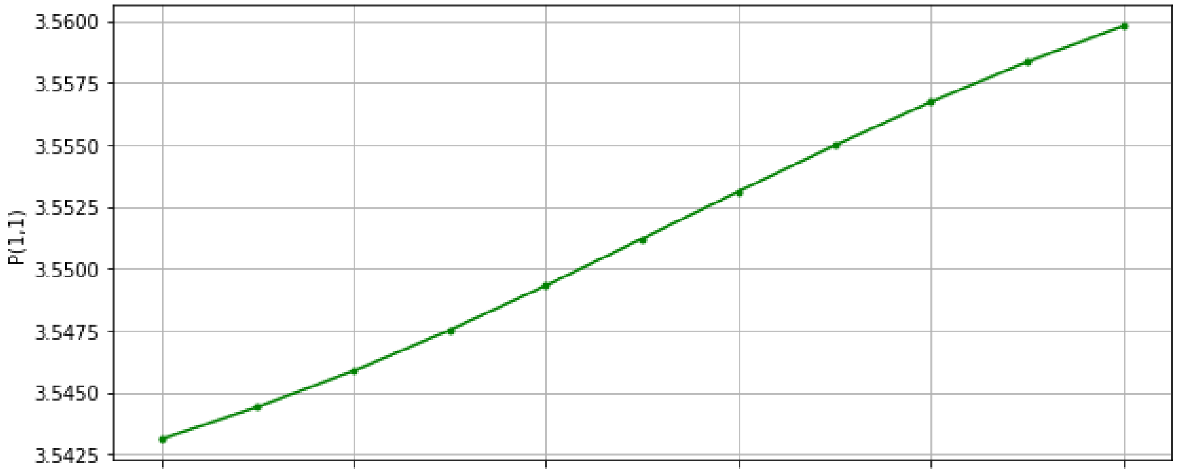
est(c4(K))
1 sigma
c4(k)
10
9
8
7
6
5
4

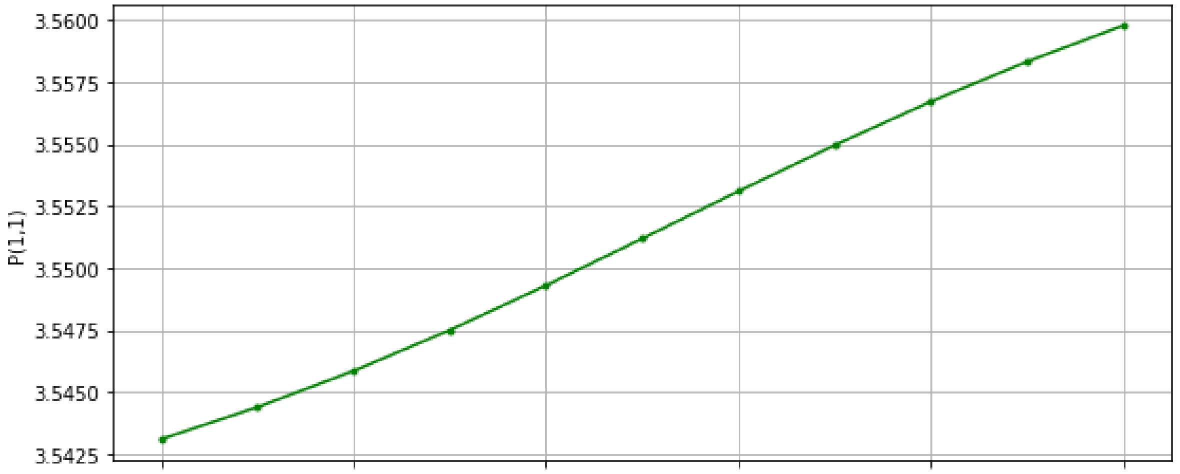


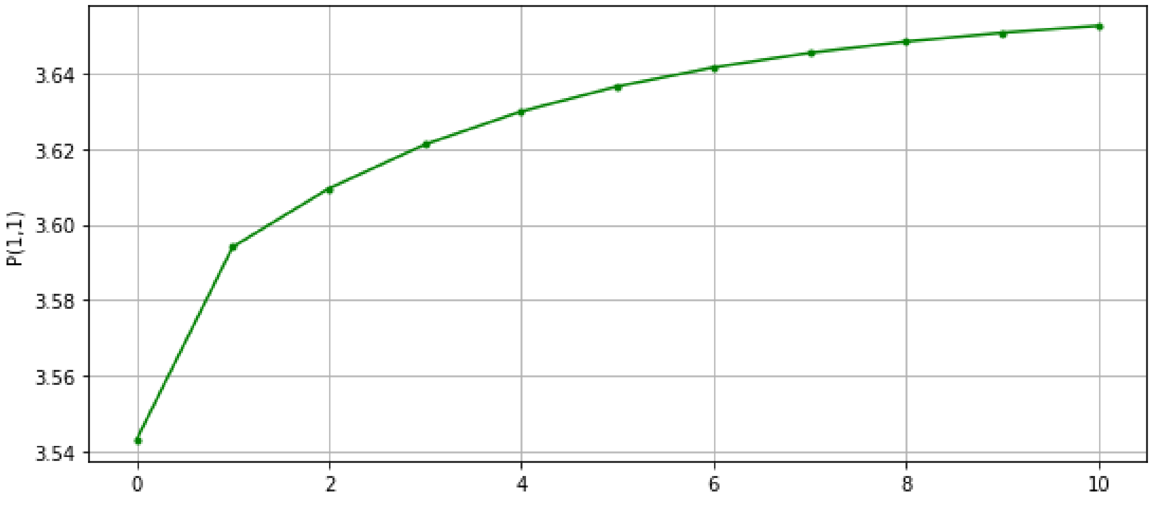
est(c5(K))
1 sigma
c5(k)
9
8
7
6
5

In [ ]:
```
uncertainties = [np.zeros([n_steps+1,1]),np.zeros([n_steps+1,1]),np.zeros([n_steps

for i in range(3):
    uncertainties[i][0,0] = sensor_uncertainty[0,0]

for s in np.arange(1,n_steps+1):

    K_uncertainty = d @ sensor_uncertainty @ d.T + process_uncertainty

    K = K_uncertainty @ mm.T @ np.linalg.inv(mm @ K_uncertainty @ mm.T + mnv)
    sensor_uncertainty = (np.eye(8) - K @ mm) @ K_uncertainty @ (np.eye(8) - K @ mm

    for i in range(3):
        uncertainties[i][s,0] = sensor_uncertainty[i,i]
```

In [ ]:
```
fig, ax = plt.subplots(3,1,sharex=True)
fig.set_size_inches(10,15)

for n in range(3):

    ax[n].plot(uncertainties[n][:,0],'g.-',label="P(1,1)")

    ax[n].set_ylabel('P(1,1)')
    ax[n].grid(True)
```







In [ ]:
```
for i in range(3):
    print(1+i,':')
    print(uncertainties[i][:,0])
```

```
1 :
[3.54310441 3.54439725 3.54585696 3.54750479 3.5493059  3.55119673
 3.55310543 3.55496532 3.55672244 3.5583385  3.55979079]
2 :
[3.54310441 3.54439725 3.54585696 3.54750479 3.5493059  3.55119673
 3.55310543 3.55496532 3.55672244 3.5583385  3.55979079]
3 :
[3.54310441 3.594111   3.60960955 3.62125136 3.63000331 3.63660853
 3.64162738 3.64547634 3.64846159 3.65080658 3.65267365]
```

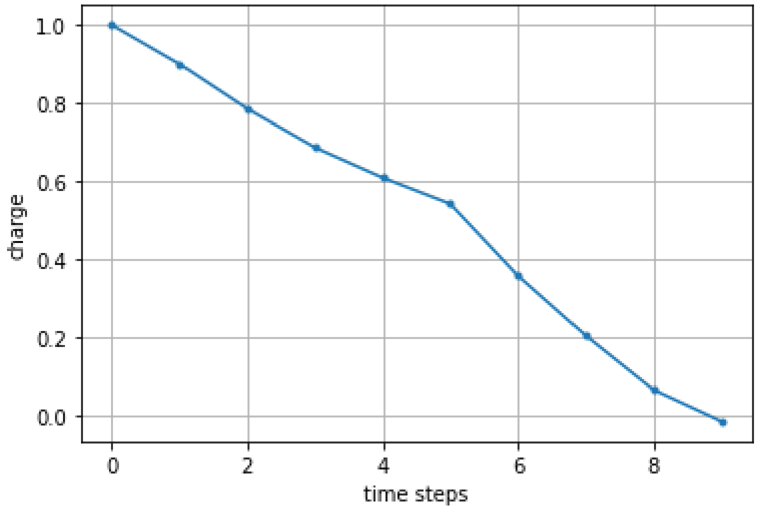
Problem 5:

In [ ]:
```
n_steps = 9

d = 1 # dynamic

# j0 = 0.1, σv = 0.05, and σw = 0.1
σv2 = 0.05**2
σw2 = 0.1**2

q0 = 1  # q(0) = 1
uncertainty_Vr0 = 0

true_state = np.random.normal(1,0)
```

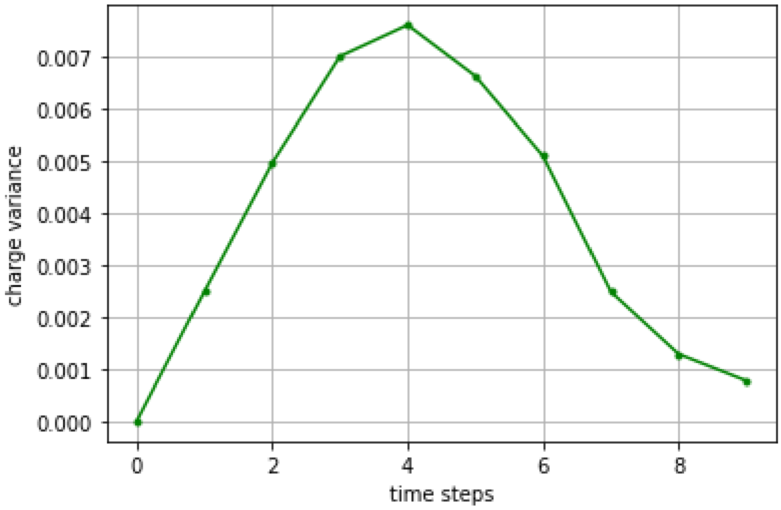
In [ ]:
```
charges = np.zeros([n_steps+1,1]) # to store the charges.
charges[0,0] = q0

uncertainty = np.zeros([n_steps+1,1])
uncertainty[0,0] = uncertainty_Vr0

measurments_ = np.zeros([n_steps+1,1]) # to store measurments.
```

In [ ]:
```
for k in np.arange(1,n_steps+1):

    noise = np.random.normal(0, 0.05)

    true_state = true_state - noise - 0.1

    w = np.random.normal(0, 0.1)

    measurments_[k,0] = 4 + ((true_state)-1)**3 + w


    Kp = q0 - 0.1
    K_uncertainty = d*uncertainty_Vr0*d + σv2

    H = 3*(Kp-1)**2
    K = K_uncertainty*H*1/(H*K_uncertainty*H + σw2)

    AVGs = 4 + ((Kp)-1)**3

    q0 = Kp + K*(measurments_[k,0]-AVGs)
    uncertainty_Vr0 = (1 - K*H)*K_uncertainty*(1 - K*H) + K*σw2*K

    charges[k,0] = q0
    uncertainty[k,0] = uncertainty_Vr0
```

In [ ]:
```
plt.plot(charges[:,0],'.-')
plt.xlabel('time steps')
plt.ylabel('charge')
plt.grid(True)
```



In [ ]:
```
plt.plot(uncertainty[:,0],'g.-')
plt.xlabel('time steps')
plt.ylabel('charge variance')
plt.grid(True)
```

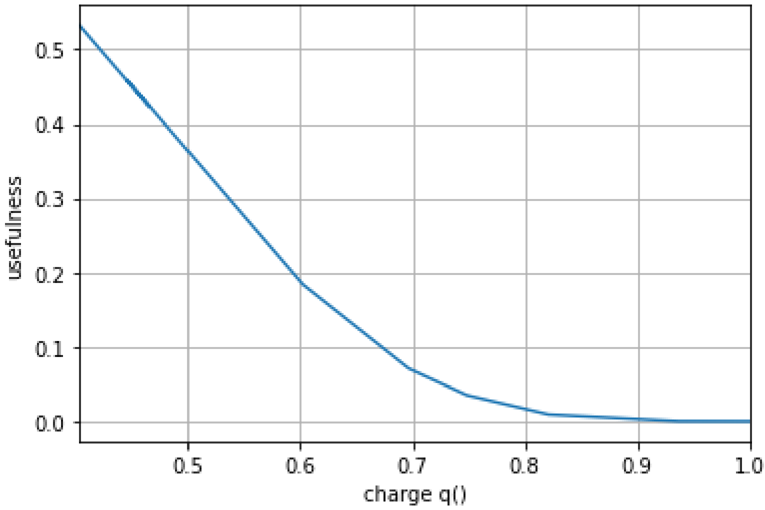


In [ ]:
```
def q(x, u, v):
    return  x - u - v

q0 = 1
σw = u = 0.1

time_steps = np.arange(n_steps+1)
Xs = np.zeros(n_steps+1)
Hs = np.zeros(n_steps+1)
Us = np.zeros(n_steps+1)

Xs[0] = q0
Hs[0] = (q0-1)**2
Us[0]=(Hs[0]**2*0.1)/(Hs[0]**2*0.1+σw)
```

In [ ]:
```
for k in range(n_steps):
        v = np.random.normal(0,0.05)
        Xs[k+1] = q(Xs[k], u, v)
        Hs[k+1] = 3*(Xs[k+1]-1)**2
        Us[k+1]=(Hs[k+1]**2*0.1)/(Hs[k+1]**2*0.1+σw)
```

In [ ]:
```
plt.plot(Xs, Us)
plt.xlim([Xs[-1], Xs[0]])
plt.xlabel('charge q()')
```

```
plt.ylabel('usefulness')
plt.grid(True)
```

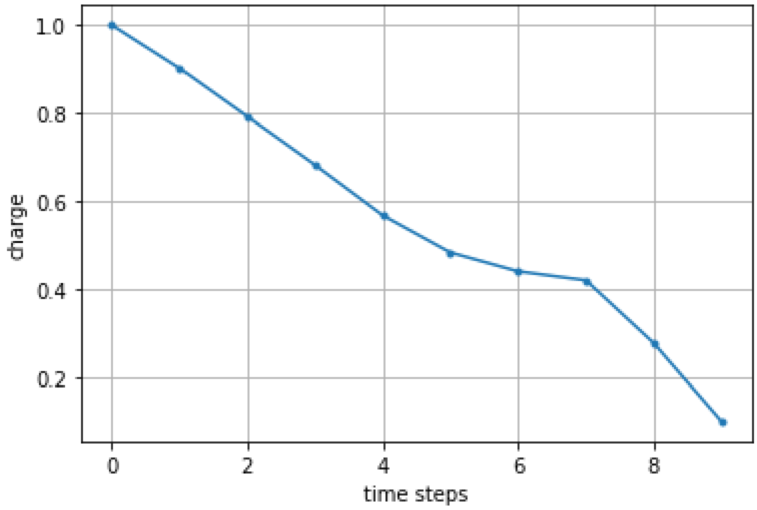


In [ ]:
```
q0 = 1  # q(0) = 1
uncertainty_Vr0 = 0

states = np.zeros([n_steps+1,1]) # to store the history of the true states.
true_state = np.random.normal(1,0)
states[0,0] = true_state
```

In [ ]:
```
measurments = [4.21, 3.83, 3.92, 3.89, 3.88, 3.89, 3.91, 3.57, 3.21]
```

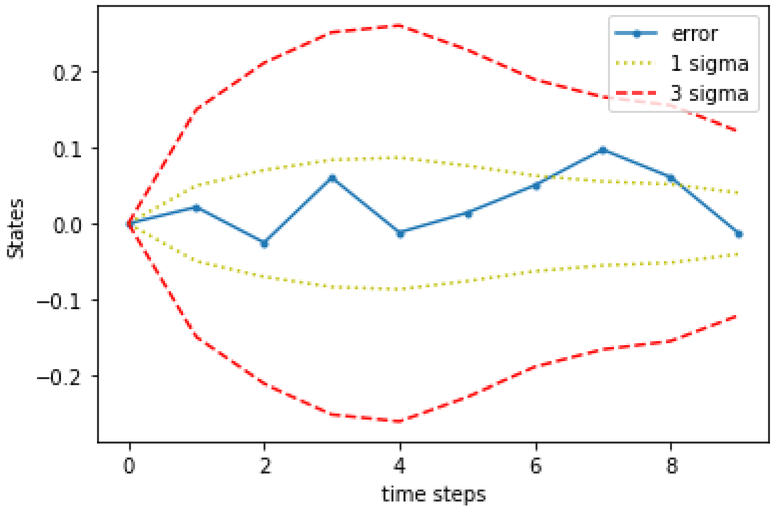
In [ ]:
```
for k in np.arange(1,n_steps+1):

    noise = np.random.normal(0, 0.05)

    true_state = true_state - noise - 0.1

    w = np.random.normal(0, 0.1)

    measurments_[k,0] = measurments[k-1]

    Kp = q0 - 0.1
    K_uncertainty = d*uncertainty_Vr0*d + σv2

    H = 3*(Kp-1)**2
    K = K_uncertainty*H*1/(H*K_uncertainty*H + σw2)

    AVGs = 4 + ((Kp)-1)**3

    q0 = Kp + K*(measurments_[k,0]-AVGs)
    uncertainty_Vr0 = (1 - K*H)*K_uncertainty*(1 - K*H) + K*σw2*K

    charges[k,0] = q0
    uncertainty[k,0] = uncertainty_Vr0
    states[k,0] = true_state
```

In [ ]:
```
plt.plot(charges[:,0],'.-')
plt.xlabel('time steps')
plt.ylabel('charge')
plt.grid(True)
```

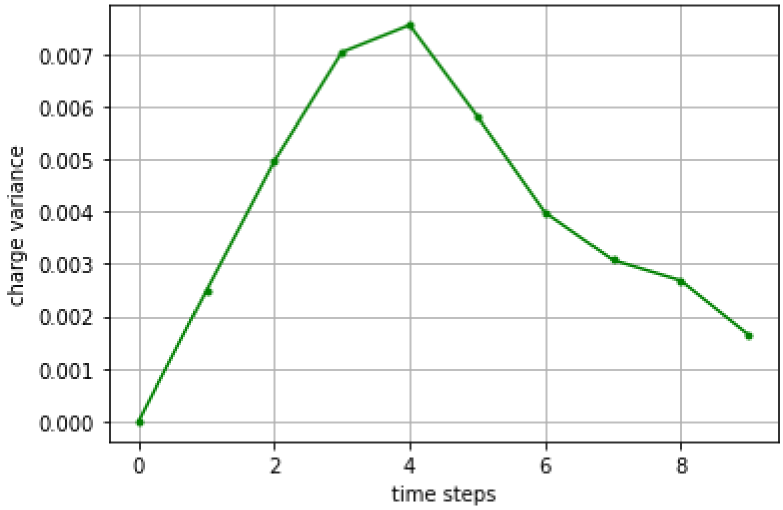


In [ ]:
```
plt.plot(charges[:,0]-states[:,0],'.-',label="error")
plt.plot(np.sqrt(uncertainty[:,0]),'y:',label="1 sigma")
plt.plot(-np.sqrt(uncertainty[:,0]),'y:',)
plt.plot(3*np.sqrt(uncertainty[:,0]),'r--',label="3 sigma")
plt.plot(-3*np.sqrt(uncertainty[:,0]),'r--',)
plt.xlabel('time steps')
plt.ylabel('States')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x22e976cd580>
```



In [ ]:
```
plt.plot(uncertainty[:,0],'g.-')
plt.xlabel('time steps')
plt.ylabel('charge variance')
plt.grid(True)
```



In [ ]:
```
# since...
j0=0.1
sigma=0.05
# then...
mean_q=1-9*j0
var_q=9*sigma**2

print(f'the mean if I did not have the voltage measurements: {round(mean_q,4)}')
print(f'the variance if I did not have the voltage measurements: {round(var_q,4)}'
```

```
the mean if I did not have the voltage measurements: 0.1
the variance if I did not have the voltage measurements: 0.0225
```

In [ ]:
```
plt.plot(measurments_[:,0],'-',label="z")
plt.plot([0] + measurments,'yo',label="z_m")

plt.xlabel('time steps')
plt.ylabel('measurements')
plt.legend();
```