1. Consider the following system:

$$x(k) = x(k-1) + v(k-1)$$
$$z(k) = x(k) + w(k)$$

where $x(0), v(\cdot), w(\cdot)$ are independent, and each is drawn from a uniform distribution in the range $[-1, 1]$.

At $k=1$ you make the observation $z(1) = 1$.

(a) Solve the pdf of the state $x(1)$ conditioned on $z(1) = 1$.

- Since uniform dist: $f(x(0)) = \frac{1}{1-(-1)} = 1/2$

- Question is asking to find pdf of state w/ non-Gaussian noise and initial condition. Therefore we use optimal state estimator.

- Start with TPT, conditioning on $x(k-1)$:

$$f(x(k) \mid z(1:k-1)) = \int_{x(k-1) \in \mathcal{X}} f(x(k) \mid x(k-1), z(1:k-1)) f(x(k-1) \mid z(1:k-1)) \, dx(k-1)$$

- Since $x(k)$ is conditionally independent of $z(1:k-1)$ given $x(k-1)$

$$f(x(k) \mid x(k-1), z(1:k-1)) = f(x(k) \mid x(k-1))$$

- This gives us our prior update step.

$$f(x(k) \mid z(1:k-1)) = \int_{x(k-1) \in \mathcal{X}} f(x(k) \mid x(k-1)) f(x(k-1) \mid z(1:k-1)) \, dx(k-1)$$

$$f(x(1) \mid z(1:0)) = \int f(x(1) \mid x(0)) f(x(0) \mid z(1:0)) \, dx(0)$$
$$= \int f(x(1) \mid x(0)) f(x(0)) \, dx(0)$$

* Omit $z(1:0)$ since that measurement doesn't exist

$$f(x(1)) = \int_{x(0) \in \mathcal{X}} f(x(1) \mid x(0)) (1/2) \, dx(0)$$

- Since $x(0)$ and $v(0)$ are independent and $x(1) = x(0) + v(0)$

$\rightarrow f(x(1) \mid x(0)) = 1/2$ if $x(1) - x(0) \in [-1, 1]$

or if $x(0) \in [x(1) - 1, x(1) + 1]$

- Now we plug in values for $f(x(1))$ to find its pdf shape

- $f(x(1)=-1) = \int_{x(0)\in\mathcal{X}} f(x(1)=-1 \mid x(0))\,(1/2)\,dx(0)$

$= \int_{x(0)\in\mathcal{X}} (1/2)(1/2)\,dx(0)$ for $x(0)\in[-2,0]$

$= \int_{-1}^{0} (1/2)(1/2)\,dx(0) = 1/4$

- $f(x(1)=0) = \int_{x(0)\in\mathcal{X}} f(x(1)=0 \mid x(0))\,(1/2)\,dx(0)$

$= \int_{x(0)\in\mathcal{X}} (1/2)(1/2)\,dx(0)$ for $x(0)\in[-1,1]$

$= \int_{-1}^{1} (1/2)(1/2)\,dx(0) = 1/2$

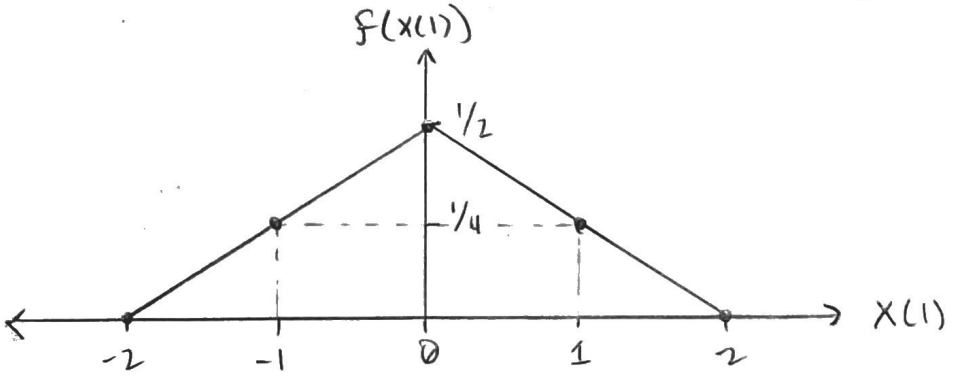
- $f(x(1)=-2) = \int_{x(0)\in\mathcal{X}} f(x(1)=-2 \mid x(0))\,(1/2)\,dx(0)$

$\star$ $x(0)$'s bounds dont coincide w/ $[-3,-1]$

$\therefore f(x(1)=-2) = 0$

- Similarly: $f(x(1)=1) = 1/4$ and $f(x(1)=2) = 0$

- It follows that $f(x(1))$'s pdf is triangular as below:



- Using $y=mx+b$, we can find the pdf equation for $x(1)\in[-2,0]$ and $x(1)\in[0,2]$

($f(x(1)) = (\text{rise}/\text{run})\,x(1) + y$ intercept

- For $x(1) \in [-2, 0]$,

$f(x(1)) = (1/2/2)\, x(1) + 1/2 = 1/2 + x(1)/4$

- For $x(1) \in [0, 2]$

$f(x(1)) = (-1/2/2)\, x(1) + 1/2 = 1/2 - x(1)/4$

$$\therefore f(x(1) \mid z(1:0)) = f(x(1)) = \begin{cases} 1/2 + \frac{x(1)}{4} & \text{if } x(1) \in [-2, 0] \\ 1/2 - \frac{x(1)}{4} & \text{if } x(1) \in [0, 2] \\ 0 & \text{else} \end{cases}$$

Next we use measurement update:

$$f(x(1) \mid z(1:1)) = \frac{f(z(1) \mid x(1))\, f(x(1) \mid z(1:0))}{\int_{\bar{x}(1) \in \mathcal{X}} f(z(1) \mid \bar{x}(1))\, f(\bar{x}(1) \mid z(1:0))\, d\bar{x}(1)}$$

- Since x and w are independent,

$$f(w) = \begin{cases} 1/2 & \text{if } x(k) \in [z(k) \pm w(k)] \\ 0 & \text{else} \end{cases}$$

- We know $z(1) = 1$,

$$\therefore f(z(1) = 1 \mid x(1)) = \begin{cases} 1/2 & \text{if } x(1) \in [0\ 2] \\ 0 & \text{else} \end{cases}$$

$$\therefore f(x(1) \mid z(1:1)) = \begin{cases} \frac{1/2\,(1/2 - x(1)/4)}{\text{denom}} & \text{if } x(1) \in [0, 2] \\ 0 & \text{else} \end{cases}$$

* Note that we only keep bounds of $f(x(1) \mid z(1:0))$ that coincided w/ $f(z(1) = 1 \mid x(1))$

• Now we solve for denominator,

$$\int_{-2}^{2} f(z(1)=1 \mid \bar{x}(1))\, f(\bar{x}(1) \mid z(1:0))\, d\bar{x}(1)$$

$$= 0 + \int_{0}^{2} (1/2)(1/2 - \tfrac{\bar{x}(1)}{4})\, d\bar{x}(1) = \int_{0}^{2} 1/4 - \tfrac{\bar{x}(1)}{8}\, d\bar{x}(1)$$

$$= \left[\frac{\bar{x}(1)}{4} - \frac{\bar{x}(1)^2}{16}\right]_0^2 = \frac{2}{4} - \frac{4}{16} = \frac{1}{4}$$

Finally,

$$f(x(1) \mid z(1)) = \frac{1/2\,(1/2 - x(1)/4)}{1/4} \quad \text{if } x(1) \in [0,2]$$

$$f(x(1) \mid z(1)) = \begin{cases} 1 - \frac{x(1)}{2} & \text{if } x(1) \in [0,2] \\ 0 & \text{else} \end{cases}$$

(b) Compute the minimum Mean Squared error estimate for $x(1)$.

$$\hat{x}^{MMSE} = E[x(1) \mid z(1)] = \int_{x(1)} x(1)\, f(x(1) \mid z(1))\, dx(1)$$

$$= \int_{0}^{2} x(1)\left(1 - \frac{x(1)}{2}\right) dx(1) = \int_{0}^{2} x(1) - \frac{x(1)^2}{2}\, dx(1)$$

$$= \left[\frac{x(1)^2}{2} - \frac{x(1)^3}{6}\right]_0^2 = \frac{4}{2} - \frac{8}{6} = \boxed{2/3}$$

(C) Compute the Maximum A Posteriori estimate for $x(1)$

$$\hat{x}^{MAP} = \hat{x} \text{ s.t } f(\hat{x} \mid x(1)) = \max_x (f(x \mid z)) = 1$$

$$\boxed{\hat{x}^{MAP} = 0} \quad \text{for} \quad f(\hat{x}^{MAP} \mid z(1))$$

Problem 5.

(a-i) Write down the model equations for the simplified problem.

- We are given: $c(k) = M + v(k)$, $z(k) = r(k) + w(k)$
- Define system state as $r(k)$, where the process noise is $v(k-1)$ as defined in the problem.

$$r(k) = r(k-1) - M + d(k-1) + v(k-1)$$

($r(k)$, $r(k-1)$: system state; $v(k-1)$: process noise)

- Define measurement as $z(k)$, where measurement noise is $w(k)$ as defined in the problem.

$$z(k) = r(k) + w(k)$$

($z(k)$: measurement; $r(k)$: state; $w(k)$: measurement noise)

(b) We now extend the previous part by also estimating the consumption, $c(k)$, where $c(k) = c(k-1) + n(k-1)$, where $n(k-1)$ is a zero-mean random variable, which is white and independent of all other quantities.

(b-i) Write down the model equations for this, noting that now your state has dimension 2.

$$\begin{bmatrix} r(k) \\ c(k) \end{bmatrix} = \begin{bmatrix} r(k-1) - c(k-1) + d(k-1) \\ c(k-1) + n(k-1) \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r(k-1) \\ c(k-1) \end{bmatrix} + \begin{bmatrix} d(k-1) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ n(k-1) \end{bmatrix}$$

($:= A$; input: $u(k-1)$; process noise: $v(k-1)$; system state)

- Our measurement $z(k)$ is defined in terms of measurment noise, $w(k)$,

$$z(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} r(k) \\ c(k) \end{bmatrix} + w(k)$$

($z(k)$: measurement; $:= H$; state; measurement noise)

(C). Now we extend the system to a more complex network with four reservoirs, and four consumers,

(ci). Write down the model equations for this problem. Make explicit what is the system state, the measurement, the process noise, and the measurement noise. Write the solution as a linear problem, and clearly give terms of the matrices A, H, etc.

- From the problem statement we can deduce the following equations,

$$r_1(k) = r_1(k-1) + d(k-1) - c_1(k-1) + \alpha\left(r_2(k-1) - r_1(k-1)\right) + \alpha\left(r_3(k-1) - r_1(k-1)\right)$$

$$r_2(k) = r_2(k-1) - c_2(k-1) + \alpha\left(r_1(k-1) - r_2(k-1)\right) + \alpha\left(r_3(k-1) - r_2(k-1)\right)$$

$$r_3(k) = r_3(k-1) - c_3(k-1) + \alpha\left(r_1(k-1) - r_3(k-1)\right) + \alpha\left(r_2(k-1) - r_3(k-1)\right) + \alpha\left(r_4(k-1) - r_3(k-1)\right)$$

$$r_4(k) = r_4(k-1) - c_4(k-1) + \alpha\left(r_3(k-1) - r_4(k-1)\right)$$

- $c_1(k) = c_1(k-1) + n_1(k-1)$
  $c_2(k) = c_2(k-1) + n_2(k-1)$
  $c_3(k) = c_3(k-1) + n_3(k-1)$
  $c_4(k) = c_4(k-1) + n_4(k-1)$

- $z_1(k) = r_1(k) + w_1(k)$
  $z_2(k) = r_2(k) + w_2(k)$
  $z_3(k) = r_3(k) + w_3(k)$
  $z_4(k) = r_4(k) + w_4(k)$

- Putting these equations into matrix form, and defining our state vector in $\mathbb{R}^8$, $x(k)$

$$\begin{bmatrix} r_1(k) \\ r_2(k) \\ r_3(k) \\ r_4(k) \\ c_1(k) \\ c_2(k) \\ c_3(k) \\ c_4(k) \end{bmatrix} = \begin{bmatrix} r_1(k-1)(1-2\alpha) + \alpha r_2(k-1) + \alpha r_3(k-1) - c_1(k-1) \\ \alpha r_1(k-1) + r_2(k-1)(1-2\alpha) + \alpha r_3(k-1) - c_2(k-1) \\ \alpha r_1(k-1) + \alpha r_2(k-1) + r_3(k-1)(1-3\alpha) - c_3(k-1) \\ \alpha r_3(k-1) + r_4(k-1)(1-\alpha) - c_4(k-1) \\ c_1(k-1) \\ c_2(k-1) \\ c_3(k-1) \\ c_4(k-1) \end{bmatrix} + \begin{bmatrix} d(k-1) \\ 0 \\ 0 \\ 0 \\ n_1(k-1) \\ n_2(k-1) \\ n_3(k-1) \\ n_4(k-1) \end{bmatrix}$$

State: $x(k)$

$$x(k) = \begin{bmatrix} 1-2\alpha & \alpha & \alpha & 0 & -1 & 0 & 0 & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & 0 & -1 & 0 & 0 \\ \alpha & \alpha & 1-3\alpha & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & \alpha & 1-\alpha & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x(k-1) + \begin{bmatrix} d(k-1) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ n_1(k-1) \\ n_2(k-1) \\ n_3(k-1) \\ n_4(k-1) \end{bmatrix}$$

State

$:= A$

input: $u(k-1)$

process noise $v(k-1)$

(c, cont)

• $Z(k)$ will be 4x1, with $x(k)$ being 8x1, H will be 4x8

$$Z(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} x(k) + \begin{bmatrix} w_1(k) \\ w_2(k) \\ w_3(k) \\ w_4(k) \end{bmatrix}$$

$Z(k)$: measurement; $:= H$; $x(k)$: state; $w(k)$:= measurement noise

(ciii) After sensor 3 fails

$$Z(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} x(k) + \begin{bmatrix} w_1(k) \\ w_2(k) \\ w_3(k) \end{bmatrix}$$

$Z(k)$: measurement; $:= H$; $x(k)$: state; $w(k)$:= measurement noise

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

''' This problem involves the implementation of a time-varying Kalman filter
 for a system with Gaussian process and measurement noises '''

# (a) Compute the PDF of y(1) given the observation z(1) = 2.22

# Define global vars

# Time-invariant linear system as given in problem statement
N = 2
A = np.array([[0.8, 0.6], [-0.6, 0.8]])
H = np.array([[1, 0]])

# Time-invariant process and measurement noise covariances given as identity
V, W = np.eye(N), np.eye(1)

# Process and measurement noise are zero mean, gaussian, and independant

'''Function that returns value corresponding to Gaussian dist matching
 required mean/variance for process and measurement noises.
 Note that mean = 0 and var = I for both v(k) and w(k)'''


def r_normal(Ex, Var): return np.random.normal(Ex, Var, 1)

# Define measurement and time update for Kalman Filter implementation


def time_update(xm, Pm):
    xp = A @ xm
    Pp = A @ Pm @ A.T + V
    return xp, Pp


def meas_update(xp, Pp, z):
    K = Pp @ H.T @ np.linalg.inv(H @ Pp @ H.T + W)
    xm = xp + K @ (z - H @ xp)
    Pm = (np.eye(N) - K @ H) @ Pp @ (np.eye(N) - K @ H).T + K @ W @ K.T
    return xm, Pm


# Define system propagation


def sym_sys(x_true):
    # Expecation and Var of v_k, w_k = 0 and I respectively
    v_k = np.array([[r_normal(0, 1)[0], r_normal(0, 1)[0]]]).T  # Process noise
    w_k = np.array([[r_normal(0, 1)[0]]])  # Scalar measurement noise
    x_true = A @ x_true + v_k
    z = H @ x_true + w_k
    return x_true, z


# Initialize estimate and covariance of state (at k = 0)
xm_0, Pm_0 = np.zeros((2, 1)), np.array([[3, 0], [0, 1]])

# First simulate prediction and measurement update steps of KF forward to k = 1
xp, Pp = time_update(xm_0, Pm_0)
xm, Pm = meas_update(xp, Pp, 2.22)

'''
Since output y(k) is an affine linear transformation of GRV x(k), y(k) itself
is a GRV with E[y] = [1 1]*E[x(k)] = [1 1]*xm and
Var[y] = E[(y-E[y])(y-E[y]).T] = [1 1]*Var[x]*[1 1].T = [1 1]*Pm(k)*[1 1].T
(see property 1 of GRVs from chapter 8 class notes)
'''


def E_y(xm): return np.array([[1, 1]]) @ xm


def Var_y(Pm): return np.array([[1, 1]]) @ Pm @ np.array([[1, 1]]).T


E_y_val = E_y(xm)
Var_y_val = Var_y(Pm)

print(
    'y(1) is normally distributed with mean: '
    + repr(round(E_y_val[0, 0], 4))
    + ' and variance: '
    + repr(round(Var_y_val[0, 0], 4))
    )
```

```
y(1) is normally distributed with mean: 1.2034 and variance: 2.8224
```

In [3]:
```
'''
(b) At which time step k (for k <= 10) do you have least knowledge about y(k)?
For which do you have the most knowledge? Interpret "little knowledge of
something" as "our estimate of something has large variance".
'''

T_f = 11  # Simulation Timesteps (0 included)

# Initialize variance storage array with first components at k = 0
Var_y_vec = np.zeros(T_f)
Var_y_vec[0] = np.diagonal(Var_y(Pm_0))

# Intialize xm, Pm
xm, Pm = xm_0, Pm_0

# Initalize true state
x_true = np.array([[r_normal(0, 3)[0], r_normal(0, 1)[0]]]).T

for k in range(1, T_f):  # Note that estimates for k = 0 is initilized above

    # Simulate system and measurement
    x_true, z = sym_sys(x_true)

    # Kalman filter estimation: time update
    xp, Pp = time_update(xm, Pm)

    # Kalman filter estimation: measurement update
    xm, Pm = meas_update(xp, Pp, z)

    # Compute variance of y(k)
    Var_y_vec[k] = Var_y(Pm)

# Find k values for min and max variance of y(k)
k_var_max = np.argmax(Var_y_vec)
k_var_min = np.argmin(Var_y_vec)

for i, val in np.ndenumerate(Var_y_vec):
    print(
        'Variance of y(k) at timestep: ' + repr(i[0]) + ' is: '
        + repr(round(val, 4)))

print(
    'We have the least knowledge about y(k) at timestep: '
    + repr(k_var_max)
    + ' and we have the most knowledge about y(k) at timestep: '
    + repr(k_var_min)
    )
```

```
Variance of y(k) at timestep: 0 is: 4.0
Variance of y(k) at timestep: 1 is: 2.8224
Variance of y(k) at timestep: 2 is: 4.0778
Variance of y(k) at timestep: 3 is: 3.869
Variance of y(k) at timestep: 4 is: 3.5873
Variance of y(k) at timestep: 5 is: 3.5073
Variance of y(k) at timestep: 6 is: 3.5014
Variance of y(k) at timestep: 7 is: 3.5036
Variance of y(k) at timestep: 8 is: 3.5038
Variance of y(k) at timestep: 9 is: 3.5034
Variance of y(k) at timestep: 10 is: 3.5032
We have the least knowledge about y(k) at timestep: 2 and we have the most knowledge about y(k) at timestep: 1
```

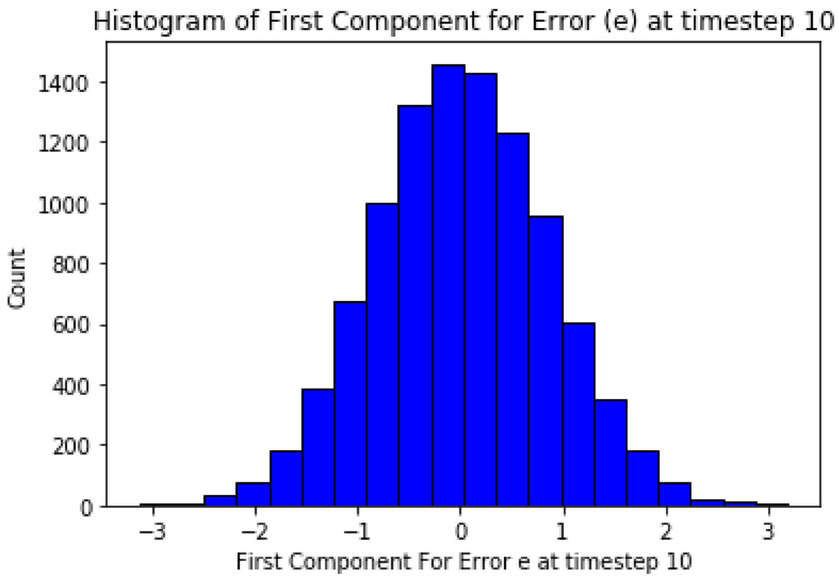
In [4]:
```
# (c) Compute the PDF of e(10)

E_e10_part_c = np.array([[0], [0]])
Var_e10_part_c = Pm

print(
    'With all random variables having expecations of zero,'
    ' and the KF being initialized with E[x0], it follows that the filter,'
    ' is unbiased, ie E[e] is: '
    + repr(E_e10_part_c)
    + ' and Var[e(k)] = E[(x(k) - xm(k))(x(k) - xm(k)).T] = Pm(k) =: '
    + repr(Var_e10_part_c)
    )
```

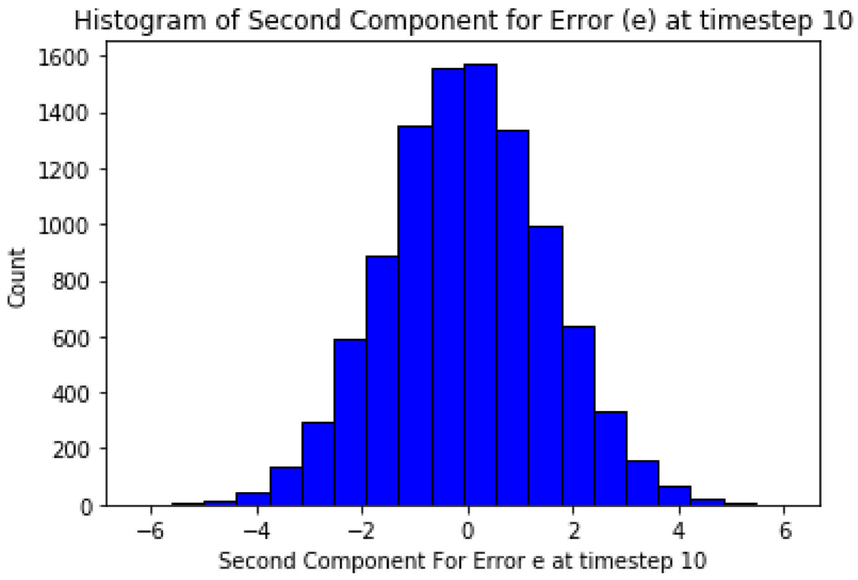
```
With all random variables having expecations of zero, and the KF being initialized with E[x0], it follows that the filter, is unbias
ed, ie E[e] is: array([[0],
       [0]]) and Var[e(k)] = E[(x(k) - xm(k))(x(k) - xm(k)).T] = Pm(k) =: array([[0.71578715, 0.2368531 ],
       [0.2368531 , 2.3137152 ]])
```

In [5]:
```
'''
(d) Implement a simulation of the system and a Kalman filter that produces an
estimate x(k). Execute this 10,000 times and plot a histogram of the resulting
values for e(10) (make two histograms,one for each component of e(10)).
 Comment on how this compares to your answer in c).
'''

sim_tot = 10000  # Number of simulations

# Intialize xm, Pm
xm, Pm = xm_0, Pm_0

# preallocate parameters
e = np.zeros([sim_tot, T_f, N])
x_est = np.zeros([sim_tot, T_f, N])  # zeros for initial x_est

for sim in range(0, sim_tot):

    # Initalize true state
    x_true = np.array([[r_normal(0, 3)[0], r_normal(0, 1)[0]]]).T

    e[sim, 0, :] = (x_true - xm).ravel()

    for k in range(1, T_f):  # Note that estimate for k = 0 is initialized above

        # Simulate system and measurement
        x_true, z = sym_sys(x_true)

        # Kalman filter estimation: time update
        xp, Pp = time_update(xm, Pm)

        # Kalman filter estimation: measurement update
        xm, Pm = meas_update(xp, Pp, z)

        # Store results for plotting
        x_est[sim, k, :] = xm.ravel()
        e[sim, k, :] = (x_true - xm).ravel()

#  Plots
num_bins = 20

plt.figure(0)
plt.hist(e[:, 10, 0], num_bins, facecolor='blue', edgecolor='black', alpha=1)
plt.xlabel('First Component For Error e at timestep 10')
plt.ylabel('Count')
plt.title(r'Histogram of First Component for Error (e) at timestep 10')

plt.figure(1)
plt.hist(e[:, 10, 1], num_bins, facecolor='blue', edgecolor='black', alpha=1)
plt.xlabel('Second Component For Error e at timestep 10')
plt.ylabel('Count')
plt.title(r'Histogram of Second Component for Error (e) at timestep 10')

'''
To compare the results from part (c) and part (d),
we compute the mean and variance of each e(10) component over all the
simulations.
'''

E_e_10_part_d = np.mean(e[:, 10, :], axis=0)
Var_e_10_part_d = np.var(e[:, 10, :], axis=0)

print(' The expectations for each component of e(10) over the 10,000 '
      'simulations from part (d) are: ' + repr(round(E_e_10_part_d[0], 4)) +
      ' and ' + repr(round(E_e_10_part_d[1], 4)) + ' respectively. Both of '
      'these values are relatively close to the values from part (c)'
      ' of: ' + repr(round(E_e10_part_c[0, 0], 4)) + ' and '
      + repr(E_e10_part_c[1, 0])
      )

print(' The variances for each component of e(10) over the 10,000 '
      'simulations from part (d) are: ' + repr(round(Var_e_10_part_d[0], 4))
      + ' and ' + repr(round(Var_e_10_part_d[1], 4)) + ' respectively. Both '
      'of these values are relatively close to the values from part (c)'
      ' of: ' + repr(round(Var_e10_part_c[0, 0], 4)) + ' and '
      + repr(round(Var_e10_part_c[1, 1], 4))
      )

plt.show()
```

The expectations for each component of e(10) over the 10,000 simulations from part (d) are: 0.0055 and -0.018 respectivelty. Both of these values are relatively close to the values from part (c) of: 0 and 0

The variances for each component of e(10) over the 10,000 simulations from part (d) are: 0.7142 and 2.3438 respectivelty. Both of these values are relatively close to the values from part (c) of: 0.7158 and 2.3137





In [ ]:

In [2]:
```
import numpy as np

'''
This problem involves the implementation of a time-varying Kalman filter
 for a simple (scalar) system with uniform process and measurement noises
'''

'''
Design a Kalman filter for the system given in Problem 1. What is the Kalman
filter estimate at time k = 1?
Comment on how this compares to the solution you got for Problem 1.
'''

# Define global vars

# Time-invariant linear system as given in problem statement
N = 1
A = 1
H = 1

# x(0), v(k), and w(k) are all uniformly distributed in the range [-1, 1]
a, b = -1, 1

# Time-invariant process and measurement noise covariances (uniform)
V, W = (b - a)**2/12, (b - a) ** 2/12

# Process and measurement noise are zero mean, white, and independant

''' Function that returns value corresponding to uniform dist matching
 required mean/variance for process and measurement noises '''


def r_uniform(): return np.random.uniform(a, b)

# Define measurement and time update for Kalman filter implementation


def time_update(xm, Pm):
    xp = A*xm
    Pp = A*Pm*A + V
    return xp, Pp


def meas_update(xp, Pp, z):
    K = Pp*H*(H*Pp*H + W)**-1
    xm = xp + K*(z - H*xp)
    Pm = (np.eye(N) - K*H)*Pp*(np.eye(N) - K*H) + K*W*K
    return xm, Pm


# Initialize estimate and covariance of state (at k = 0)
xm, Pm = 0, (b - a) ** 2/12

# Find KF estimate at time k = 1 given that z(1) = 1
z = 1

# Kalman filter estimation: time update
xp, Pp = time_update(xm, Pm)

# Kalman filter estimation: measurement update
xm, Pm = meas_update(xp, Pp, z)

print(
    'Kalman filter estimate at time k = 1 is: '
    + repr(np.round(xm, 4))
    )

print(
    'With Pm at time k = 1 being: '
    + repr(np.round(Pm[0, 0], 4))
    )

print('The KF estimate here of 2/3 coincides with the MMSE estimate from'
      ' problem 1. This makes sense intuitively because amongst the class'
      ' of linear and unbiased estimators, the Kalman filter optimally'
      ' minimizes the mean squared estimation error. The difference in'
      ' implementation between a standard KF and the optimal estimator from'
      ' problem 1 is that we do not compute the entire state pdf at each'
      ' timestep for a KF, only its MMSE estimate and variance.')
```

```
Kalman filter estimate at time k = 1 is: 0.6667
With Pm at time k = 1 being: 0.2222
The KF estimate here of 2/3 coincides with the MMSE estimate from problem 1. This makes sense intuitively because amongst the class
of linear and unbiased estimators, the Kalman filter optimally minimizes the mean squared estimation error. The difference in implem
entation between a standard KF and the optimal estimator from problem 1 is that we do not compute the entire state pdf at each times
tep for a KF, only its MMSE estimate and variance.
```

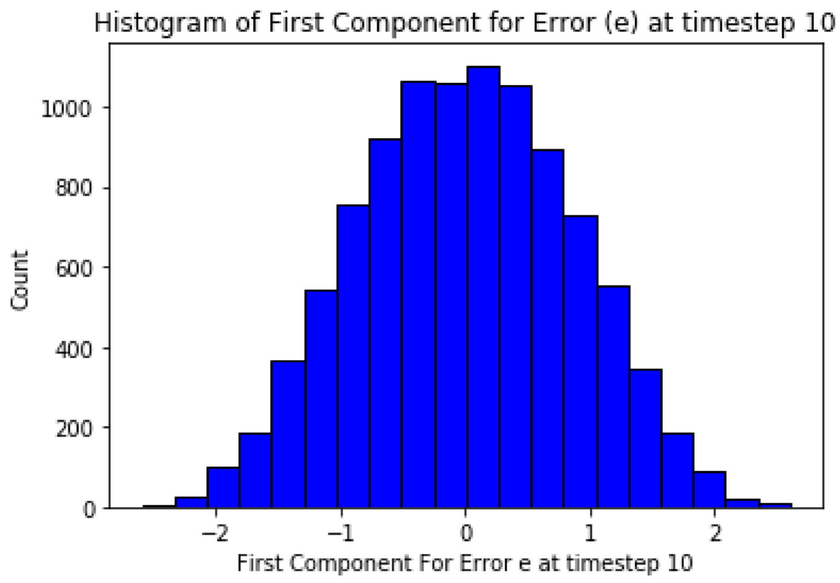
In [3]:
```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt

'''
This problem involves the implementation of a time-varying Kalman filter
for a system with uniform (ie non-Gaussian) process and measurement noises
'''

'''
Design an optimal linear estimator for this system (KF), and
implement a simulation of the system and a Kalman filter that produces an
estimate as a function of k. For the simulation, draw all of the random
variables from uniform distributions that match the required mean/variance.
Execute this 10,000 times and plot a histogram of the resulting values for
e(10) (make two histograms, one for each component of e(10)). Comment on
how this compares to your answers in Problem 2c) and Problem 2d). Also give
the ensemble average and variance for each component of e(10)
(that is, the average over the samples from the simulations for each
component). Compare that to the estimator's output, and briefly discuss.
'''

# Define global vars

# Time-invariant linear system as given in problem statement
N = 2
A = np.array([[0.8, 0.6], [-0.6, 0.8]])
H = np.array([[1, 0]])

# Time-invariant process and measurement noise covariances given as identity
V, W = np.eye(N), np.eye(1)

# Process and measurement noise are zero mean, white, and independant

'''
The problem asks us to draw all of the random variables from uniform
distributions matching the required mean/variance. In order to implement
this in python, we need the bounds of each uniform distribution. The variance
for a uniform distribution is: Var = (b - a)^2 / 12 where 'b' is the upper
bound of the distribution and 'a' is the lower bound. The mean for a uniform
distribution is then Mean = (a + b) / 2. We will solve for these bounds
using Sympy and the given problem info for x0, w(k), and v(k).
'''


def get_bounds(mean, var):
    a, b = sp.Symbol("a"), sp.Symbol("b")
    eqtns = (sp.Eq((a + b) / 2, mean), sp.Eq((b - a)**2 / 12, var))
    ans = sp.solve(eqtns, (a, b))
    return ans[0][0], ans[0][1]  # return first answer, 'b' being larger


# mean and variance for each component of v(k) and w(k) is 0 and 1 respectively
mean_vw, var_vw = 0, 1
a_vw, b_vw = get_bounds(mean_vw, var_vw)

# mean and variance for 1st component of x(k) is 0 and 3 respectively
mean_x1, var_x1 = 0, 3
a_x1, b_x1 = get_bounds(mean_x1, var_x1)

# mean and variance for 1st component of x(k) is 0 and 1 respectively
mean_x2, var_x2 = 0, 1
a_x2, b_x2 = get_bounds(mean_x2, var_x2)

'''
Next define function that returns value corresponding to uniform dist
matching required mean/variance for process and measurement noises
'''


def r_uniform(a, b): return np.random.uniform(a, b)

# Define measurement and time update for Kalman Filter implementation


def time_update(xm, Pm):
    xp = A @ xm
    Pp = A @ Pm @ A.transpose() + V
    return xp, Pp


def meas_update(xp, Pp, z):
    K = Pp @ H.transpose() @ np.linalg.inv(H @ Pp @ H.transpose() + W)
    xm = xp + K @ (z - H @ xp)
    Pm = (np.eye(N) - K @ H) @ Pp @ (np.eye(N) - K @ H).transpose()
    + K @ W @ K.transpose()
    return xm, Pm
```

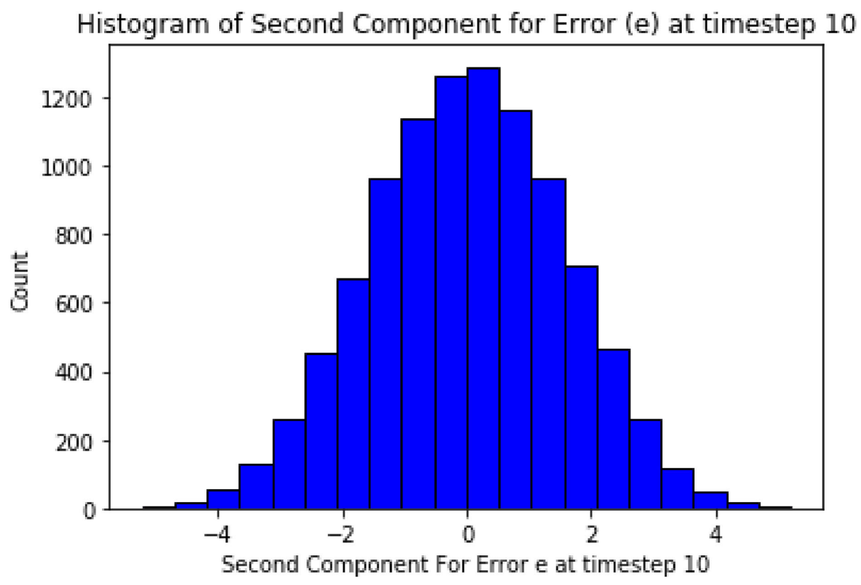
```python
# Define system propagation


def sym_sys(x_true):
    # a = -sqrt(3), b = sqrt(3) for uniform dist of v_k and w_k (see writing)
    v_k = np.array([[r_uniform(a_vw, b_vw)],
                    [r_uniform(a_vw, b_vw)]])  # Process noise
    w_k = np.array([r_uniform(a_vw, b_vw)])  # measurement noise
    x_true = A @ x_true + v_k
    z = H @ x_true + w_k
    return x_true, z


# Initialize estimate and covariance of state (at k = 0)
xm, Pm = np.zeros((2, 1)), np.array([[3, 0], [0, 1]])

T_f = 11  # Simulation Timesteps (0 included)
sim_tot = 10000  # Number of simulations

# preallocate parameters
e = np.zeros((sim_tot, T_f, N))
x_est = np.zeros((sim_tot, T_f, N))  # zeros for initial x_est

for sim in range(0, sim_tot):

    # Initalize true state, a = -3, b = 3 for uniform dist of x[0]
    x_true = np.array([[r_uniform(a_x1, b_x1)], [r_uniform(a_x2, b_x2)]])
    e[sim, 0, :] = (x_true - xm).ravel()

    for k in range(1, T_f):  # Note that estimate for k=0 is initialized above

        # Simulate system and measurement
        x_true, z = sym_sys(x_true)

        # Kalman filter estimation: time update
        xp, Pp = time_update(xm, Pm)

        # Kalman filter estimation: measurement update
        xm, Pm = meas_update(xp, Pp, z)

        # Store results for plotting
        x_est[sim, k, :] = xm.ravel()
        e[sim, k, :] = (x_true - xm).ravel()

print(
    'The results from these plots are similar to problem (2c) and (2d) in that'
    ' the averages for each component of e(10) are approximately centered'
    ' around zero. Additionally, the variances for each component of e(10) '
    ' are relatively similar for both problems at ~ 0.7 and ~2.3'
    ' respectively. It makes sense that each problem formulation yeilds'
    ' similar results, since the KF doesnt require RVs to have any particular'
    ' distribution. '
    )
```

```
The results from these plots are similar to problem (2c) and (2d) in that the averages for each component of e(10) are approximately centered around zero. Additionally, the variances for each component of e(10)  are relatively similar for both problems at ~ 0.7 and ~2.3 respectively. It makes sense that each problem formulation yeilds similar results, since the KF doesnt require RVs to have any particular distribution.
```

In [4]:
```
#  Plotting
num_bins = 20

plt.figure(0)
plt.hist(e[:, 10, 0], num_bins, facecolor='blue', edgecolor='black', alpha=1)
plt.xlabel('First Component For Error e at timestep 10')
plt.ylabel('Count')
plt.title(r'Histogram of First Component for Error (e) at timestep 10')

plt.figure(1)
plt.hist(e[:, 10, 1], num_bins, facecolor='blue', edgecolor='black', alpha=1)
plt.xlabel('Second Component For Error e at timestep 10')
plt.ylabel('Count')
plt.title(r'Histogram of Second Component for Error (e) at timestep 10')
```

Out[4]:
```
Text(0.5, 1.0, 'Histogram of Second Component for Error (e) at timestep 10')
```





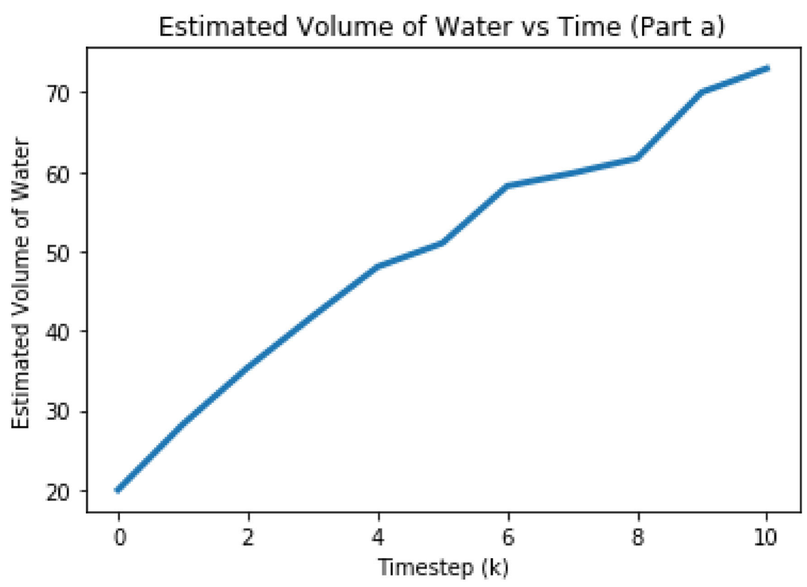
In [5]:
```
#  Display ensemble average, variance and est output for each comp of e(10)
print('Ensemble Average of First component for e(10): '
      + repr(round(np.mean(e[:, 10, 0]), 4)))
print('Ensemble Variance of First component for e(10): '
      + repr(round(np.var(e[:, 10, 0]), 4)))
print(
    'Ensemble Average of First component for Estimator Output at Timestep 10: '
    + repr(round(np.mean(x_est[:, 10, 0]), 4)) + '\n'
      )

print('Ensemble Average of Second component for e(10): '
      + repr(round(np.mean(e[:, 10, 1]), 4)))
print('Ensemble Variance of Second component for e(10): '
      + repr(round(np.var(e[:, 10, 1]), 4)))
print(
    'Ensemble Average of Second component for Estimator Output at Timestep 10: '
    + repr(round(np.mean(x_est[:, 10, 1]), 4))
      )

print(
    'The ensemble averages for each component of e(10) are relatively close to'
    ' zero, meaning that the estimator is approximately unbiased, which makes'
    ' sense. The variance of the second component for e(10) is higher than'
    ' that of the first component, meaning that the estimator is less'
    ' confident in its estimate of the second state.'
    )
```

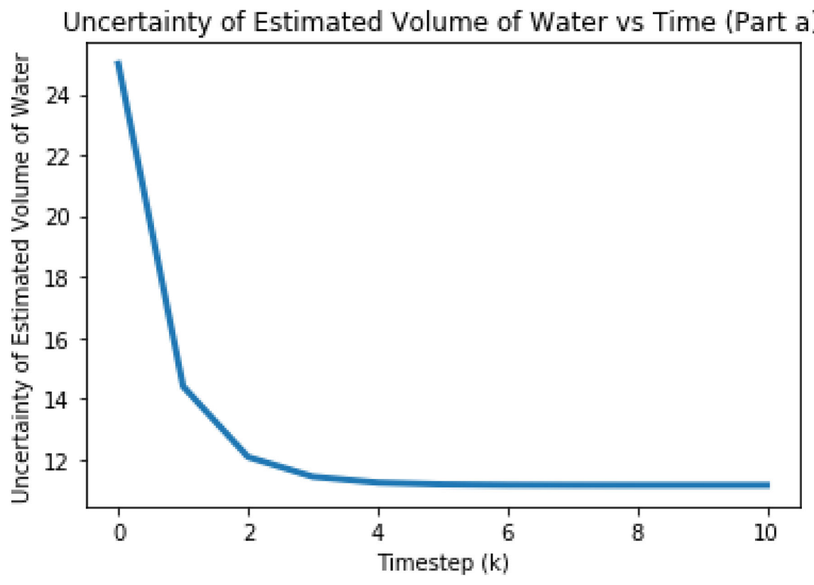
```
Ensemble Average of First component for e(10): -0.0023
Ensemble Variance of First component for e(10): 0.737
Ensemble Average of First component for Estimator Output at Timestep 10: 0.0037

Ensemble Average of Second component for e(10): -0.0037
Ensemble Variance of Second component for e(10): 2.3874
Ensemble Average of Second component for Estimator Output at Timestep 10: -0.0069
The ensemble averages for each component of e(10) are relatively close to zero, meaning that the estimator is approximately unbiase
d, which makes sense. The variance of the second component for e(10) is higher than that of the first component, meaning that the es
timator is less confident in its estimate of the second state.
```

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt

'''
In this problem we estimate the behavior of a
city's water network, where fresh water is supplied by
a desalination plant, and consumed at various points
in the network. The network contains various con-
sumers, and various reservoirs where water is locally
stored.
Our objective is to use noisy information about pro-
duction and consumption levels, and noisy measure-
ments of the amount of water in the reservoirs, to
estimate the state of the network. We will consider 3
networks of increasing complexity.
'''

'''
(a) We will first investigate a very simplified system, with a single
reservoir and single consumer (i.e. we only have one tank level r
to keep track of). We will model our consumers as c(k) = m + v(k),
where m is the typical consumption, and v(k) is a zero-mean uncertainty,
assumed white and independent of all other quantities. Using the Kalman filter,
estimate the actual volume of water r for various k. Also provide the associated
uncertainty for your estimate of r. Provide this using graphs.
'''

# Define global vars

# Time-invariant linear system deduced from problem statement
N = 1
A = 1
H = 1

# Time-invariant process and sensor uncertainity (variance)
V, W = 9, 25

# Process and measurement noise are zero mean, white, and independant

# define constant supply of water, d, and predicted customer use, m
d, m = 10, 7

# Define measurement and time update for Kalman filter implementation


def time_update_a(xm, Pm):
    xp = A*xm - m + d  # state update constants are appropriately added here
    Pp = A*Pm*A + V
    return xp, Pp


def meas_update_a(xp, Pp, z):
    K = Pp*H*(H*Pp*H + W)**-1
    xm = xp + K*(z - H*xp)
    Pm = (np.eye(N) - K*H)*Pp*(np.eye(N) - K*H) + K*W*K
    return xm, Pm


# Initialize estimate and covariance of state (at k = 0)
xm, Pm = 20, 25

T_f = 11  # Simulation Timesteps (0 included)

# preallocate parameters
x_est = np.zeros([T_f])
x_est[0] = xm
P_est = np.zeros([T_f])
P_est[0] = Pm

# Given measurements
z = np.array([32.1, 39.8, 45.9, 52.0, 51.0, 63.4, 58.1, 60.3, 76.6, 73.0])

for k in range(1, T_f):  # Note that estimate for k=0 is initialized above

    # Kalman filter estimation: time update
    xp, Pp = time_update_a(xm, Pm)

    # Kalman filter estimation: measurement update
    xm, Pm = meas_update_a(xp, Pp, z[k-1])

    # Store results for plotting
    x_est[k] = xm
    P_est[k] = Pm
```

In [16]:
```
#  Plotting
plt.figure(0)
plt.plot(range(0, T_f), x_est, linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated Volume of Water')
plt.title(r'Estimated Volume of Water vs Time (Part a)')

plt.figure(1)
plt.plot(range(0, T_f), P_est, linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Uncertainty of Estimated Volume of Water')
plt.title(r'Uncertainty of Estimated Volume of Water vs Time (Part a)')
```

Out[16]: Text(0.5, 1.0, 'Uncertainty of Estimated Volume of Water vs Time (Part a)')

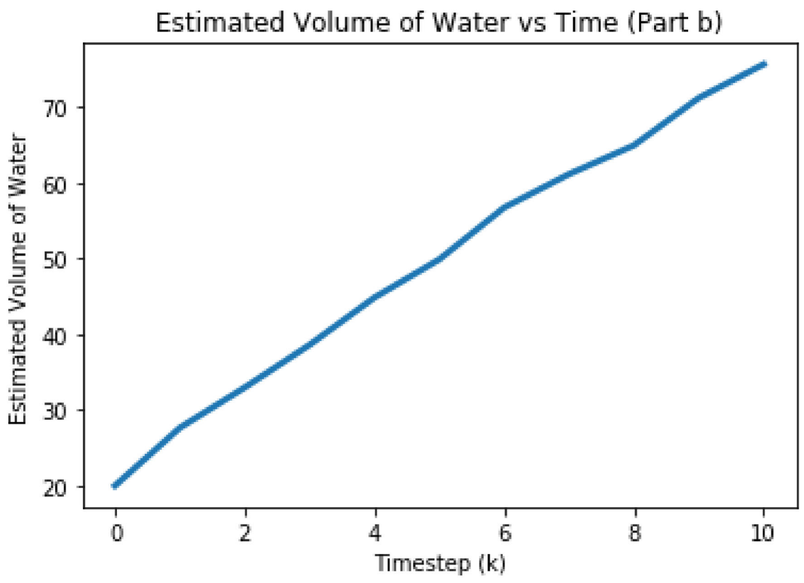


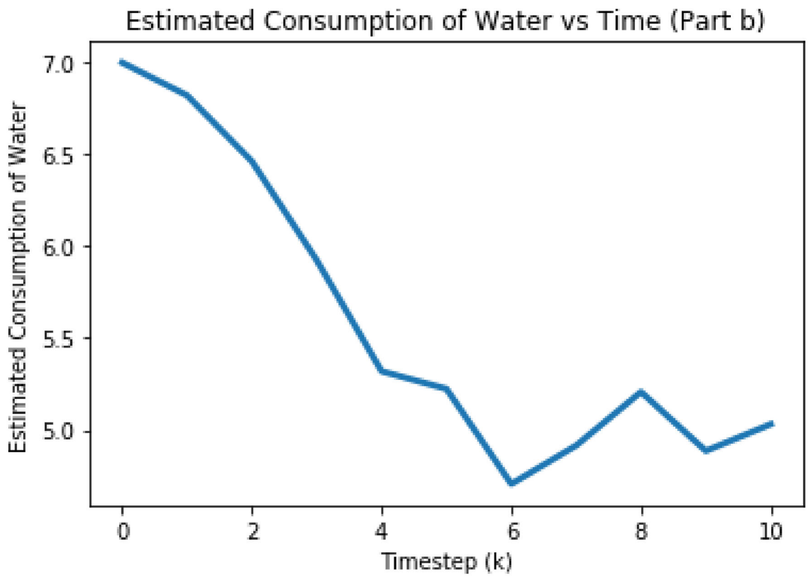


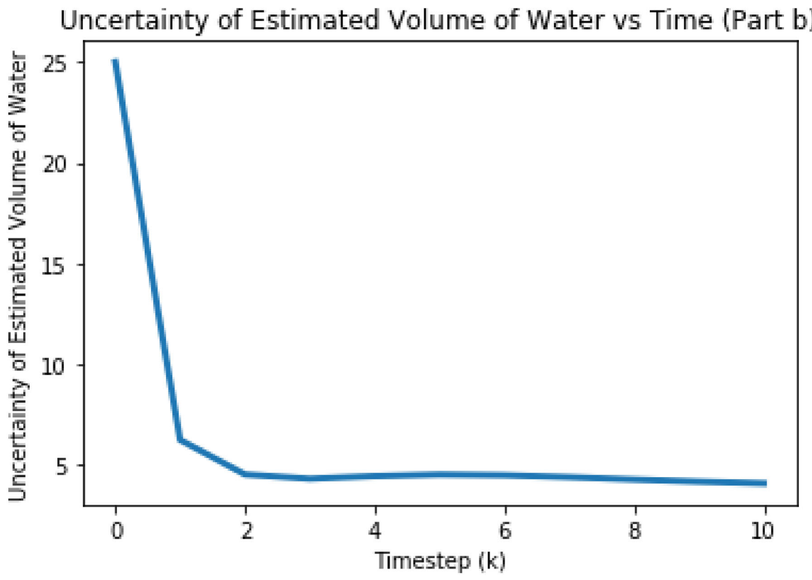
In [17]:
```
'''
(b) We now extend the previous part by also estimating the consumption, c(k).
We model the consumption as evolving as c(k) = c(k-1) + n(k-1), where n(k-1)
is a zero-mean random variable, which is white and independent of all other
quantities. Using the Kalman filter, estimate the actual volume of water r for
various k, and the consumption rate c(k). Also provide the associated
uncertainty for both. Provide this using graphs. How do your answers for r
compare to the previous case?
'''

# Define global vars

# Time-invariant linear system deduced from problem statement
N = 2
A = np.array([[1, -1], [0, 1]])
H = np.array([[1, 0]])

# Time-invariant process and sensor uncertainity (variance)
V, W = 0.1, 25

# Process and measurement noise are zero mean, white, and independant

# Constant supply of water, d, is defined above

# Define measurement and time update for Kalman filter implementation


def time_update_b(xm, Pm):
    # Known input, d added to system
    xp = A @ xm + np.array([[d], [0]])
    # Process noise only affects second component of state below
    Pp = A @ Pm @ A.T + np.array([[0, 0], [0, V]])
    return xp, Pp


def meas_update_b(xp, Pp, z):
    K = Pp @ H.T @ np.linalg.inv(H @ Pp @ H.T + W)
    xm = xp + K @ (z - H @ xp)
    Pm = (np.eye(N) - K @ H) @ Pp @ (np.eye(N) - K @ H).T
    + K * W * K.T
    return xm, Pm


# Initialize estimate and covariance of state (at k = 0)
xm, Pm = np.array([[20], [7]]), np.array([[25, 0], [0, 1]])

# preallocate parameters
x_est = np.zeros((T_f, N))
x_est[0] = xm.ravel()
P_est = np.zeros((T_f, N))
P_est[0] = Pm.diagonal()

# Given measurements are same as part (a)

for k in range(1, T_f):  # Note that estimate for k=0 is initialized above

    # Kalman filter estimation: time update
    xp, Pp = time_update_b(xm, Pm)

    # Kalman filter estimation: measurement update
    xm, Pm = meas_update_b(xp, Pp, z[k-1])

    # Store results for plotting
    x_est[k] = xm.ravel()
    P_est[k] = Pm.diagonal()
```

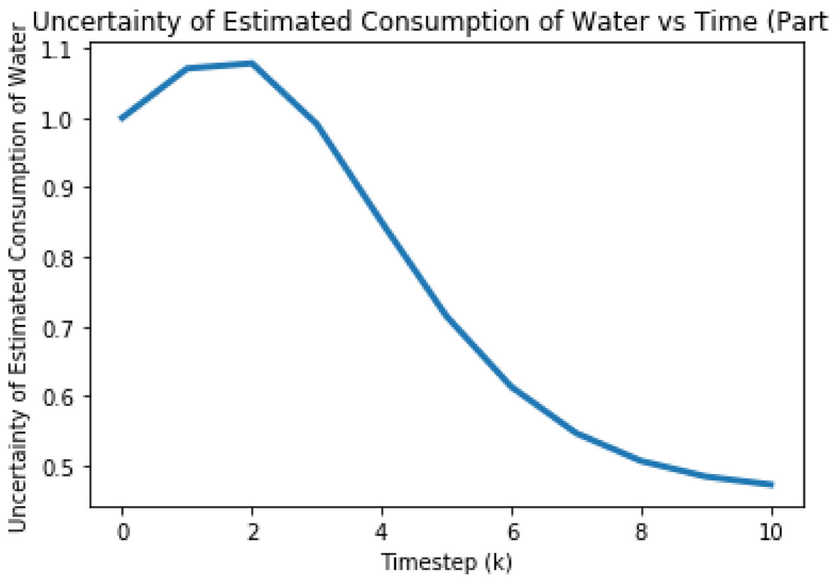
In [18]:
```
#  Plotting
plt.figure(2)
plt.plot(range(0, T_f), x_est[:, 0], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated Volume of Water')
plt.title(r'Estimated Volume of Water vs Time (Part b)')

plt.figure(3)
plt.plot(range(0, T_f), x_est[:, 1], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated Consumption of Water')
plt.title(r'Estimated Consumption of Water vs Time (Part b)')

plt.figure(4)
plt.plot(range(0, T_f), P_est[:, 0], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Uncertainty of Estimated Volume of Water')
plt.title(r'Uncertainty of Estimated Volume of Water vs Time (Part b)')

plt.figure(5)
plt.plot(range(0, T_f), P_est[:, 1], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Uncertainty of Estimated Consumption of Water')
plt.title(r'Uncertainty of Estimated Consumption of Water vs Time (Part b)')

print('The answers for r(k) in part (a) and part (b) are shaped similarily.'
      ' However, the steady state uncertainty from (b) was lower than that'
      ' of part (a) as shown in the graphs. Also, the graph for part (b) is'
      ' smoother than part (a).')
```

The answers for r(k) in part (a) and part (b) are shaped similarily. However, the steady state uncertainty from (b) was lower than t
hat of part (a) as shown in the graphs. Also, the graph for part (b) is smoother than part (a).







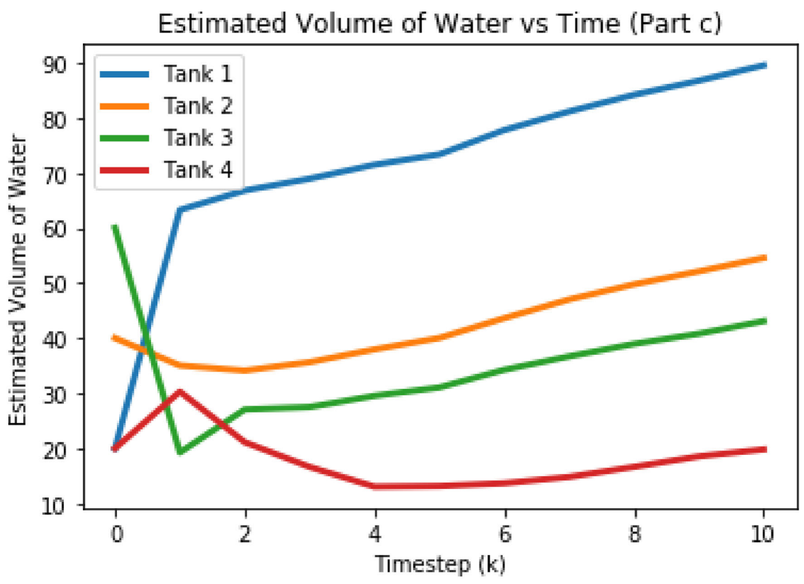


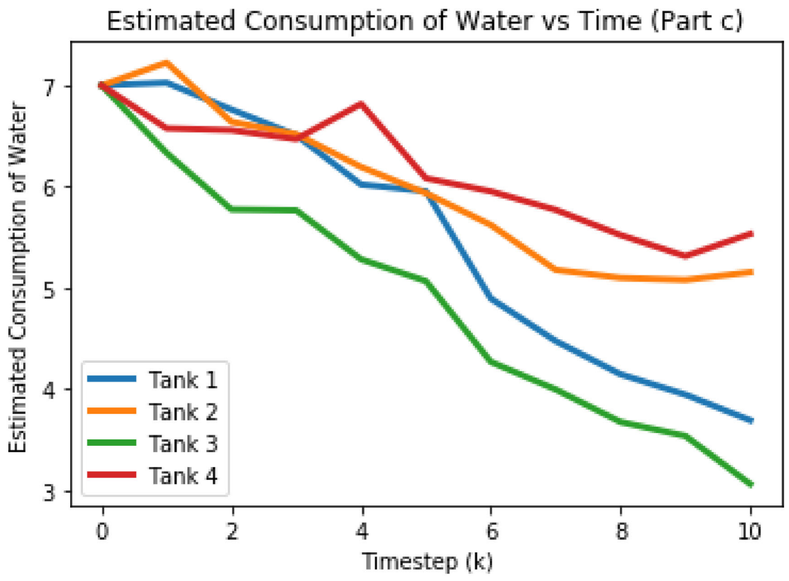
In [19]:

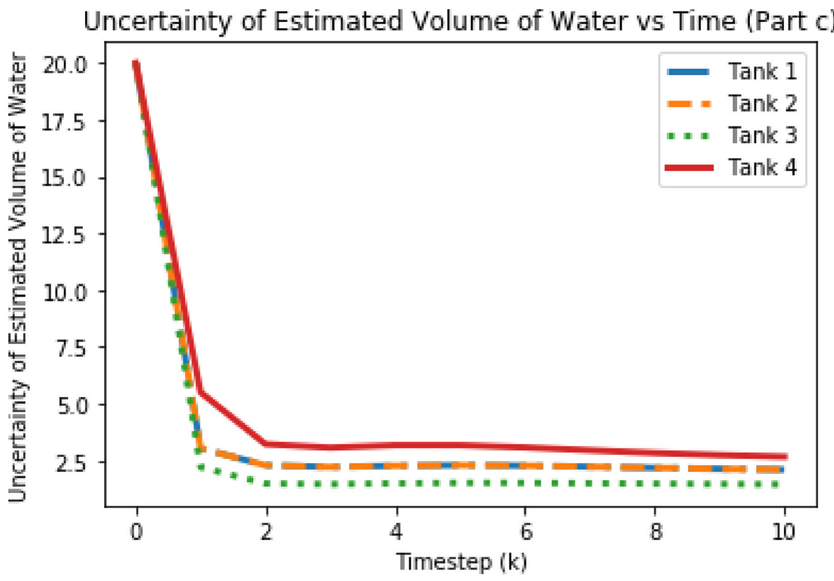
```
'''
(c) Now, we extend the system to a more complex network with four reservoirs,
and four consumers. Using the Kalman filter, estimate the actual volume of
water r(i) over various k for all tanks. Also provide the associated
uncertainty. Provide this using graphs. Notice that the estimate uncertainty
for tank 1 is lower than the solution you had for the simpler
network, why is this?
'''

# Define global vars

# Time-invariant linear system deduced from problem statement
N = 8
d = 30  # constant supply of water d(k)
alph = 0.3  # Flow balancing of 0.3

A = np.concatenate((
    np.concatenate((
        np.array([[1-2*alph, alph, alph, 0],
                  [alph, 1-2*alph, alph, 0],
                  [alph, alph, 1-3*alph, 0],
                  [0, 0, alph, 1-alph]]),
        -1*np.eye(4)), axis=1),
    np.concatenate((np.zeros((4, 4)), np.eye(4)), axis=1)),
    axis=0)

H = np.concatenate((np.eye(4), np.zeros((4, 4))), axis=1)

# Time-invariant process and measurement noise covariances
V = 0.1 * np.eye(N)
V[0:4, 0:4] = 0  # Adjust V so process noise only affects last four states
W = 25 * np.eye(4)  # z(k) is of size 4

# Process and measurement noise are zero mean, white, and independant

# Define measurement and time update for Kalman Filter implementation


def time_update_c(xm, Pm):
    uk = np.array([[d, 0, 0, 0, 0, 0, 0, 0]]).T  # add input 'd' to 1st state
    xp = A @ xm + uk
    Pp = A @ Pm @ A.T + V
    return xp, Pp


def meas_update_c(xp, Pp, z):
    K = Pp @ H.T @ np.linalg.inv(H @ Pp @ H.T + W)
    xm = xp + K @ (z - H @ xp)
    Pm = (np.eye(N) - K @ H) @ Pp @ (np.eye(N) - K @ H).T
    + K @ W @ K.T
    return xm, Pm


# Initialize estimate and covariance of state (at k = 0)
xm = np.array([[20, 40, 60, 20, 7, 7, 7, 7]]).T
Pm = np.diag([20, 20, 20, 20, 1, 1, 1, 1])

# preallocate parameters
x_est = np.zeros((T_f, N))
x_est[0] = xm.ravel()
P_est = np.zeros((T_f, N))
P_est[0] = Pm.diagonal()

# Given measurements
z = np.array([[62.6, 70.3, 73.5, 77.2, 73.2, 94.2, 87.4, 89.7, 90.4, 94.2],
              [29.4, 44.8, 37.3, 40.1, 44.1, 43.8, 53.8, 49.9, 51.9, 52.1],
              [35.9, 38.8, 25.9, 39.0, 31.2, 46.9, 39.6, 44.0, 42.5, 54.2],
              [40.9, 21.9, 18.0, 08.8, 23.9, 17.2, 18.6, 22.0, 22.9, 17.2]])

for k in range(1, T_f):  # Note that estimate for k=0 is initialized above

    # Kalman filter estimation: time update
    xp, Pp = time_update_c(xm, Pm)

    # Kalman filter estimation: measurement update
    # (ensuring z(k) is 4x1 after slicing)
    xm, Pm = meas_update_c(xp, Pp, z[0:4, k-1].reshape(4, 1))

    # Store results for plotting
    x_est[k] = xm.ravel()
    P_est[k] = Pm.diagonal()
```

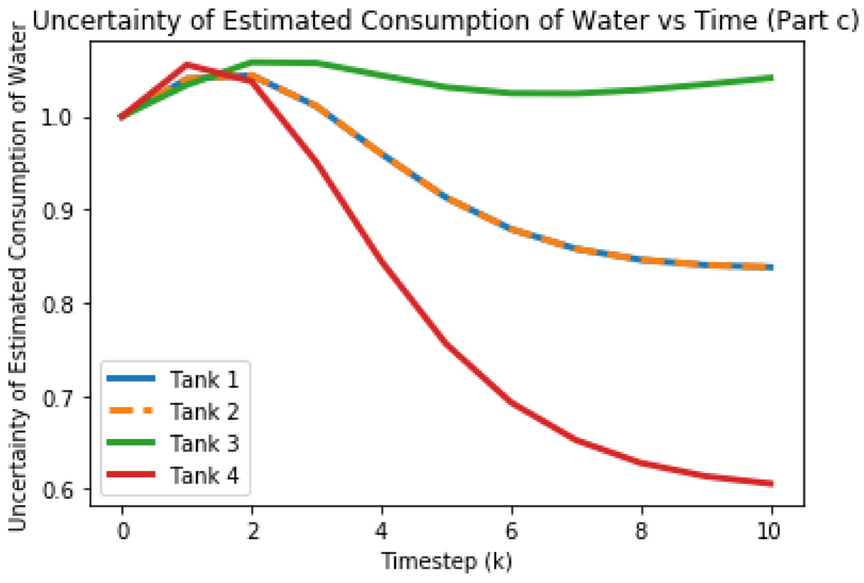
In [20]:
```
#  Plotting
plt.figure(6)
plt.plot(range(0, T_f), x_est[:, 0:4], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated Volume of Water')
plt.title(r'Estimated Volume of Water vs Time (Part c)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="upper left")

plt.figure(7)
plt.plot(range(0, T_f), x_est[:, 4:9], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated Consumption of Water')
plt.title(r'Estimated Consumption of Water vs Time (Part c)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="lower left")

linestyle = ['-.', '--', ':', '-']
plt.figure(8)
for k in range(0, 4):
    plt.plot(range(0, T_f), P_est[:, k], linestyle=linestyle[k], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Uncertainty of Estimated Volume of Water')
plt.title(r'Uncertainty of Estimated Volume of Water '
          'vs Time (Part c)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="upper right")

linestyle = ['-', '--', '-', '-']
plt.figure(9)
for k in range(4, 8):
    plt.plot(range(0, T_f), P_est[:, k], linestyle=linestyle[k-4], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Uncertainty of Estimated Consumption of Water')
plt.title(r'Uncertainty of Estimated Consumption of Water '
          'vs Time (Part c)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="lower left")

print(
    'The estimate uncertainty for tank 1 is lower than the solution we had'
    ' for the simpler network because the system dynamics bewteen tanks are'
    ' coupled. Therefore the additional measurements of added states reduces '
    'the estimated uncertainty of tank 1.')
```

The estimate uncertainty for tank 1 is lower than the solution we had for the simpler network because the system dynamics bewteen tanks are coupled. Therefore the additional measurements of added states reduces the estimated uncertainty of tank 1.





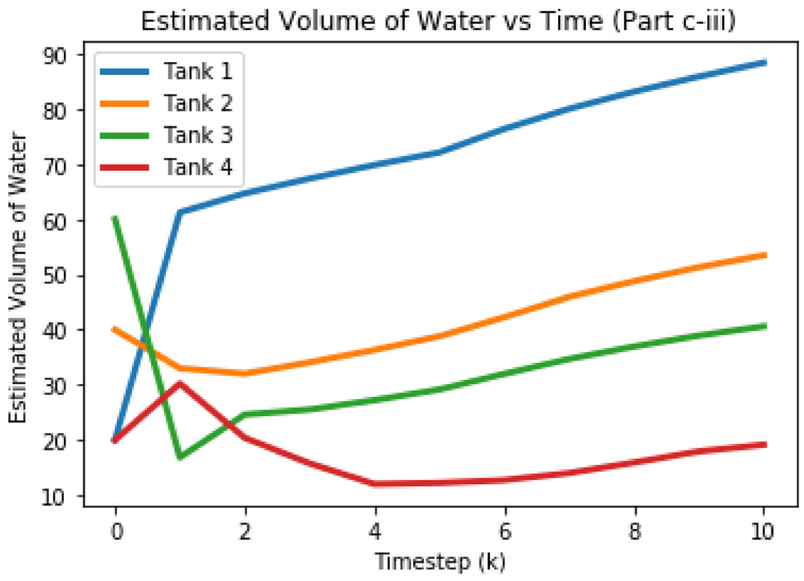


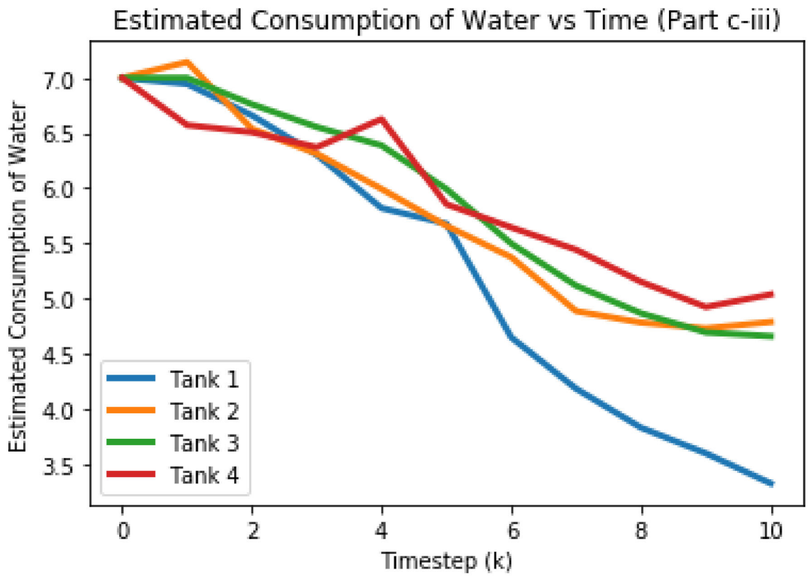


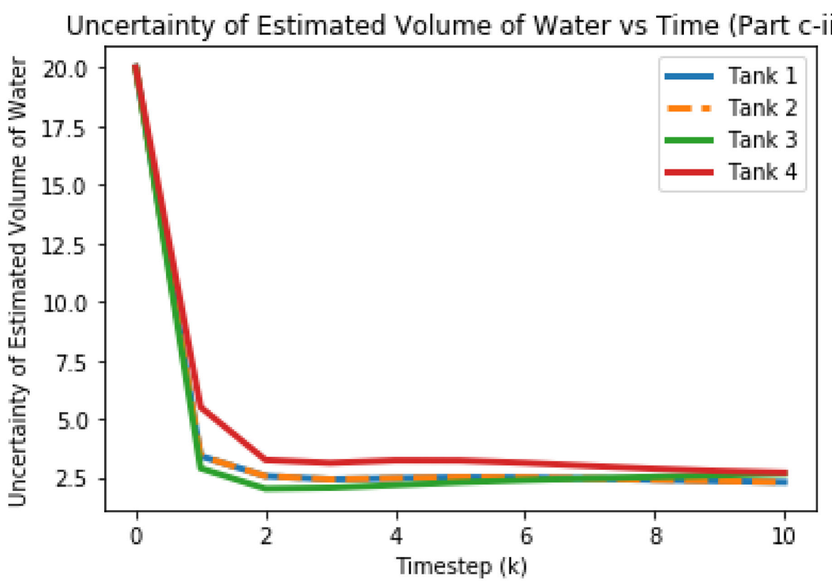
In [21]:
```
'''
(c-iii) We will now repeat the previous problem, except that now the sensor
of tank 3 has failed, and thus no longer provides any measurements. Modify
your Kalman filter from before to adjust for the lost sensor, otherwise
using previous data.
'''

# N, d, alpha, and A are same as before, H is redefined below
H = np.concatenate((H[0:2], H[3:4]), axis=0)

# Process noise is the same as before, measurement noise redefined below
W = 25 * np.eye(3)  # z(k) is of size 3

# Process and measurement noise are zero mean, white, and independant

# Initialize estimate and covariance of state (at k = 0)
xm = np.array([[20, 40, 60, 20, 7, 7, 7, 7]]).T
Pm = np.diag([20, 20, 20, 20, 1, 1, 1, 1])

# preallocate parameters
x_est = np.zeros((T_f, N))
x_est[0] = xm.ravel()
P_est = np.zeros((T_f, N))
P_est[0] = Pm.diagonal()

# Given measurements (tank 3 sensor removed)
z = np.array([[62.6, 70.3, 73.5, 77.2, 73.2, 94.2, 87.4, 89.7, 90.4, 94.2],
              [29.4, 44.8, 37.3, 40.1, 44.1, 43.8, 53.8, 49.9, 51.9, 52.1],
              [40.9, 21.9, 18.0, 08.8, 23.9, 17.2, 18.6, 22.0, 22.9, 17.2]])

for k in range(1, T_f):  # Note that estimate for k=0 is initialized above

    # Kalman filter estimation: time update
    xp, Pp = time_update_c(xm, Pm)

    # Kalman filter estimation: measurement update
    # (ensuring z(k) is 3x1 after slicing)
    xm, Pm = meas_update_c(xp, Pp, z[0:3, k-1].reshape(3, 1))

    # Store results for plotting
    x_est[k] = xm.ravel()
    P_est[k] = Pm.diagonal()
```

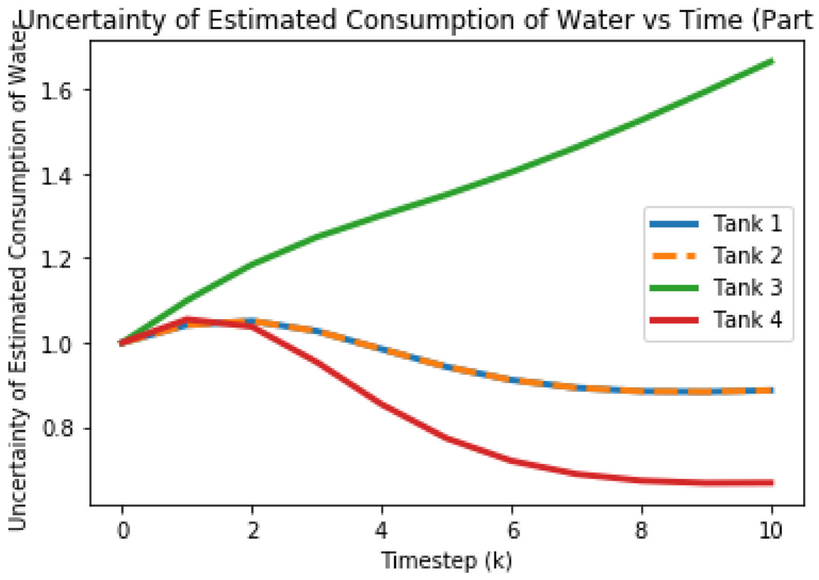
In [22]:
```
#  Plotting
plt.figure(10)
plt.plot(range(0, T_f), x_est[:, 0:4], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated Volume of Water')
plt.title(r'Estimated Volume of Water vs Time (Part c-iii)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="upper left")

plt.figure(11)
plt.plot(range(0, T_f), x_est[:, 4:9], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated Consumption of Water')
plt.title(r'Estimated Consumption of Water vs Time (Part c-iii)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="lower left")

linestyle = ['-', '--', '-', '-']
plt.figure(12)
for k in range(0, 4):
    plt.plot(range(0, T_f), P_est[:, k], linestyle=linestyle[k], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Uncertainty of Estimated Volume of Water')
plt.title(r'Uncertainty of Estimated Volume of Water '
          'vs Time (Part c-iii)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="upper right")

linestyle = ['-', '--', '-', '-']
plt.figure(13)
for k in range(4, 8):
    plt.plot(range(0, T_f), P_est[:, k], linestyle=linestyle[k-4], linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Uncertainty of Estimated Consumption of Water')
plt.title(r'Uncertainty of Estimated Consumption of Water '
          'vs Time (Part c-iii)')
plt.legend(labels=['Tank 1', 'Tank 2', 'Tank 3', 'Tank 4'], loc="right")

print('Losing this sensor has slightly affected our ability to estimate'
      ' the tank levels. As seen by comparing the graphs bewteen (c) and'
      ' (c-iii), Losing the tank 3 sensor moderately increased the uncertainty'
      ' for the estimated consumption and volume of tank 3 only. Uncertainties'
      ' for the remaining 6 states remained relatively unchanged. Estimation'
      ' graphs were affected by removing the sensor, but their overall shapes'
      ' remained consistent. ')
```

Losing this sensor has slightly affected our ability to estimate the tank levels. As seen by comparing the graphs bewteen (c) and (c-iii), Losing the tank 3 sensor moderately increased the uncertainty for the estimated consumption and volume of tank 3 only. Uncertainties for the remaining 6 states remained relatively unchanged. Estimation graphs were affected by removing the sensor, but their overall shapes remained consistent.









In [ ]: