In [1]:
```
import numpy as np
import sympy as sp

'''
Design an Extended Kalman Filter (EKF) for the given 2-state nonlinear system
and compute its posteriori estimate xm(1), given z(1) = 0.5.
Usual independance assumptions apply
'''

'''
Define given nonlinear system and measurement model. Note that process and
measureent noise will be set to zero for EKF but is included in function to
abide by math notation. Assume no inputs to system.
'''



def nlsys(xk, vk):  # 'q'
    return np.array([[np.sin(xk[0][0]) + np.cos(xk[1][0]) + vk[0]],
                     [np.cos(xk[0][0]) - np.sin(xk[1][0]) + vk[1]]])


def nlmeas(xk, w):  # 'h
    return xk[0][0] * xk[1][0] + w


# Define global vars
V, W = np.array([[0.3, 0], [0, 0.3]]), 0.2


# (a) Initialization
xm, Pm = np.array([[0.5], [0.5]]), np.array([[0.5, 0], [0, 0.5]])

# (b) Prior update: Jacobian matrices A and L
x1, x2, v1, v2 = sp.symbols('x1 x2 v1 v2')
xk, vk = sp.Matrix([[x1], [x2]]), sp.Matrix([[v1], [v2]])

q = sp.Matrix([[sp.sin(x1) + sp.cos(x2) + v1],
               [sp.cos(x1) - sp.sin(x2) + v2]])

# Take jacobians of nl system model and lambdify to functions of np arrays
A = sp.lambdify([x1, x2, v1, v2], q.jacobian(xk))
L = sp.lambdify([x1, x2, v1, v2], q.jacobian(vk))

# Input values
Ak = A(xm[0][0], xm[1][0], 0, 0)
Lk = L(xm[0][0], xm[1][0], 0, 0)

# Apply linear KF prediction equations
xp, Pp = nlsys(xm, [0, 0]), Ak @ Pm @ Ak.T + Lk @ V @ Lk.T

# (c) Measurement update: Jacobian matrices H and M
w = sp.symbols('w')
h = sp.Matrix([[x1*x2 + w]])

# Take jacobians of nl meas model and lambdify to functions of np arrays
H = sp.lambdify([x1, x2, w], h.jacobian(xk))
M = sp.lambdify([x1, x2, w], h.diff(w))

# Input values
Hk = H(xp[0][0], xp[1][0], 0)
Mk = M(xp[0][0], xp[1][0], 0)

# Apply linear KF measurement update equations
z1 = 0.5
Kk = Pp @ Hk.T @ np.linalg.inv(Hk @ Pp @ Hk.T + Mk * W * Mk)
xm = xp + Kk * (z1 - nlmeas(xp, 0))
Pm = (np.eye(2) - Kk @ Hk) @ Pp

# (d) Highlight final answer xm(k = 1)
print('Posterior estimate, xm(k=1), given z(k=1) is: ' + repr(xm))
```

```
Posterior estimate, xm(k=1), given z(k=1) is: array([[1.34987626],
       [0.37385011]])
```

Problem 2. Let x be a scalar random variable, with a symmetric PDF (that is, $f_x(\mu + \tilde{x}) = f_x(\mu_x - \tilde{x})$ for $\mu_x = E[x]$ and any $\tilde{x}$). Let $y = g(x)$ for an analytic, scalar-valued, function g.

(a) Show that the unscented transform correctly predicts the mean of y up to third order in the function g.

(b) Up to which order is the unscented transform variance prediction correct?

(a) For a taylor expansion of $g(x)$ about point $\tilde{x}$

$$g(x) \approx g(\tilde{x}) + \frac{\partial g(\tilde{x})}{\partial x}(x-\tilde{x}) + \frac{1}{2}\frac{\partial^2 g(\tilde{x})}{\partial x^2}(x-\tilde{x})^2 + \frac{1}{6}\frac{\partial^3 g(\tilde{x})}{\partial x^3}(x-\tilde{x})^3$$

- Now apply this to $E[y]$ below, using μ_x as our linearization point.
- $E[y] = E[g(x)] \approx E\left[g(\mu_x) + \frac{\partial g(\mu_x)}{\partial x}(x-\mu_x) + \frac{1}{2}\frac{\partial^2 g(\mu_x)}{\partial x^2}(x-\mu_x)^2 + \frac{1}{6}\frac{\partial^3 g(\mu_x)}{\partial x^3}(x-\mu_x)^3 \right]$
- Expectation is distributable...

$$= E[g(\mu_x)] + \frac{\partial g(\mu_x)}{\partial x}\left(E[x] - E[\mu_x]\right) + \frac{1}{2}\frac{\partial^2 g(\mu_x)}{\partial x^2}E[(x-\mu_x)^2] + \frac{1}{6}\frac{\partial^3 g(\mu_x)}{\partial x^3}E[(x-\mu_x)^3]$$

($E[x] - E[\mu_x] = 0$; $E[(x-\mu_x)^2] = \mathrm{Var}[x]$)

$$= g(\mu_x) + \frac{1}{2}\frac{\partial^2 g}{\partial x}\mathrm{Var}[x] + \frac{1}{6}\frac{\partial^3 g(\mu_x)}{\partial x^3}E[(x^3-\mu_x)^3]$$

- Can compare w/ unscented transform w/ $n=1$ giving 2 sigma points

$$s_{x,0} = \mu_x + \sqrt{\mathrm{Var}[x]} \qquad s_{x,1} = \mu_x - \sqrt{\mathrm{Var}[x]}$$

- $s_{y,0} = g(s_{x0}) = g(\mu_x) + \frac{\partial g}{\partial x}(\mu_x)(s_{x,0}-\mu_x) + \frac{1}{2}\frac{\partial^2 g}{\partial x^2}(\mu_x)(s_{x,0}-\mu_x)^2 + \frac{1}{6}\frac{\partial^3 g}{\partial x^2}(\mu_x)(s_{x0}-\mu_x)^3$

- since $S_{x0} - \mu_x = +\sqrt{Var[x]}$

- $S_{y,0} = g(\mu_x) + \frac{\partial g}{\partial x}(\mu_x)\sqrt{Var[x]} + \frac{1}{2}\frac{\partial^2 g}{\partial x^2}(\mu_x)Var[x]$

$+ \frac{1}{6}\frac{\partial^3 g}{\partial x^3}(\mu_x)(Var[x])^{3/2}$

- Similarly, $(S_{x1} - \mu_x) = -\sqrt{Var[x]}$

- $S_{y,1} = g(\mu_x) - \frac{\partial g}{\partial x}\sqrt{Var[x]} + \frac{1}{2}\frac{\partial^2 g}{\partial x^2}(\mu_x)Var[x]$

$- \frac{1}{6}\frac{\partial^3 g}{\partial x^3}(\mu_x)(Var[x])^{3/2}$

- Mean of unscented transform is then,

$\mu_y = \sum_{i=0}^{1} \frac{1}{2} S_{y,i} = g(\mu_x) + \frac{1}{2}\frac{\partial^2 g}{\partial x^2}(\mu_x)Var[x]$

- Comparing this to taylor series,

$E[y] = g(\mu_x) + \frac{1}{2}\frac{\partial^2 g}{\partial x}Var[x] + \frac{1}{6}\frac{\partial^3 g(\mu_x)}{\partial x}E[(x^3 - \mu_x)^3]$

- The two solutions are different, but maybe the third order term simplifies to 0? I stopped the problem here.

In [1]:
```
import numpy as np
import sympy as sp

'''
This problem will look at three different systems with deterministic dynamics,
but uncertain initial state. In each case, E[x(0)] = 1 and Var [x(0)] = 4,
and we are interested in x(1) = q(x(0)). We will consider three
different approaches to predict E[x(1)] and Var [x(1)], described below.
'''

E_x, Var_x = 1, 4

print('Part (a):')


# Consider q(x) = -x + 2|x|
def q_a(x): return -x + 2*abs(x)


# (a-i) Use techniques from the EKF (linearization), create function for dq/dx
def A_a(x):
    if x >= 0:
        return 1
    else:
        return -3


# Prediction (with x0 = E[x0], and P0 = Var[x0])
xp, Pp = q_a(E_x), A_a(E_x) * Var_x * A_a(E_x)

print('Using techniques from the EKF, E[x(1)] = ' + repr(round(xp, 4))
      + ' and Var[x(1)] = ' + repr(round(Pp, 4)))

# (a-ii) Use the unscented transform

# The state, x, is scalar so we will have sigma points 0 and (2(1)-1) = 1
sx0, sx1 = E_x + np.sqrt(Var_x), E_x - np.sqrt(Var_x)

# Transform sigma-points through nl function
sy0, sy1 = q_a(sx0), q_a(sx1)

# Compute approximation for resulting mean and variance
xp, Pp = np.mean([sy0, sy1]), np.var([sy0, sy1])

print('Using techniques from the unscented transform, E[x(1)] = '
      + repr(round(xp, 4)) + ' and Var[x(1)] = ' + repr(round(Pp, 4)))

'''
(a-iii) Numerically approximate the statistics by repeated sampling.
Generate 10**6 samples, and do this for both x(0) normally distributed,
and uniformly distributed.
'''
samp = 10**6


# Function to compute uniform distrubution bounds from sample statistics
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


# Generate samples for x0 being normally and uniformly distributed
a, b = get_bounds(E_x, Var_x)
x0_norm = np.random.normal(E_x, np.sqrt(Var_x), samp)
x0_uni = np.random.uniform(a, b, samp)

# Transform x0 in x1 with nonlinear function and compute resulting statistics
x1_norm, x1_uni = q_a(x0_norm), q_a(x0_uni)
xp_norm, Pp_norm = np.mean(x1_norm), np.var(x1_norm)
xp_uni, Pp_uni = np.mean(x1_uni), np.var(x1_uni)

print('Using repeated sampling with x0 normally distributed, E[x(1)] = '
      + repr(round(xp_norm, 4)) + ' and Var[x(1)] = '
      + repr(round(Pp_norm, 4)))

print('Using repeated sampling with x0 uniformly distributed, E[x(1)] = '
      + repr(round(xp_uni, 4)) + ' and Var[x(1)] = ' + repr(round(Pp_uni, 4)))
```

```
Part (a):
Using techniques from the EKF, E[x(1)] = 1 and Var[x(1)] = 4
Using techniques from the unscented transform, E[x(1)] = 3.0 and Var[x(1)] = 0.0
Using repeated sampling with x0 normally distributed, E[x(1)] = 2.5817 and Var[x(1)] = 5.0269
Using repeated sampling with x0 uniformly distributed, E[x(1)] = 2.7524 and Var[x(1)] = 3.1847
```

In [2]:
```
print('Part (b):')


# Consider q(x) = (x - 1)**3
def q_b(x): return (x - 1)**3


# (b-i) Use techniques from the EKF (linearization), create function for dq/dx
def A_b(x): return 3*(x-1)**2


# Prediction (with x0 = E[x0], and P0 = Var[x0])
xp, Pp = q_b(E_x), A_b(E_x) * Var_x * A_b(E_x)

print('Using techniques from the EKF, E[x(1)] = ' + repr(round(xp, 4))
      + ' and Var[x(1)] = ' + repr(round(Pp, 4)))

# (b-ii) Use the unscented transform

# The state, x, is still scalar so we can use same sx0 and sx1 points as above

# Transform sigma-points through nl function
sy0, sy1 = q_b(sx0), q_b(sx1)

# Compute approximation for resulting mean and variance
xp, Pp = np.mean([sy0, sy1]), np.var([sy0, sy1])

print('Using techniques from the unscented transform, E[x(1)] = '
      + repr(round(xp, 4)) + ' and Var[x(1)] = ' + repr(round(Pp, 4)))

'''
(b-iii) Numerically approximate the statistics by repeated sampling.
Generate 10**6 samples, and do this for both x(0) normally distributed,
and uniformly distributed.
'''

# Use above generated samples for x0 being normally and uniformly distributed

# Transform x0 in x1 with nonlinear function and compute resulting statistics
x1_norm, x1_uni = q_b(x0_norm), q_b(x0_uni)
xp_norm, Pp_norm = np.mean(x1_norm), np.var(x1_norm)
xp_uni, Pp_uni = np.mean(x1_uni), np.var(x1_uni)

print('Using repeated sampling with x0 normally distributed, E[x(1)] = '
      + repr(round(xp_norm, 4)) + ' and Var[x(1)] = '
      + repr(round(Pp_norm, 4)))

print('Using repeated sampling with x0 uniformly distributed, E[x(1)] = '
      + repr(round(xp_uni, 4)) + ' and Var[x(1)] = ' + repr(round(Pp_uni, 4)))
```

```
Part (b):
Using techniques from the EKF, E[x(1)] = 0 and Var[x(1)] = 0
Using techniques from the unscented transform, E[x(1)] = 0.0 and Var[x(1)] = 64.0
Using repeated sampling with x0 normally distributed, E[x(1)] = 0.0428 and Var[x(1)] = 960.4209
Using repeated sampling with x0 uniformly distributed, E[x(1)] = -0.0079 and Var[x(1)] = 247.0387
```

In [3]:
```
print('Part (c):')


# Consider q(x) = 3x
def q_c(x): return 3*x


# (c-i) Use techniques from the EKF (linearization), create function for dq/dx
def A_c(): return 3


# Prediction (with x0 = E[x0], and P0 = Var[x0])
xp, Pp = q_c(E_x), A_c() * Var_x * A_c()

print('Using techniques from the EKF, E[x(1)] = ' + repr(round(xp, 4))
      + ' and Var[x(1)] = ' + repr(round(Pp, 4)))

# (c-ii) Use the unscented transform

# The state, x, is still scalar so we can use same sx0 and sx1 points as above

# Transform sigma-points through nl function
sy0, sy1 = q_c(sx0), q_c(sx1)

# Compute approximation for resulting mean and variance
xp, Pp = np.mean([sy0, sy1]), np.var([sy0, sy1])

print('Using techniques from the unscented transform, E[x(1)] = '
      + repr(round(xp, 4)) + ' and Var[x(1)] = ' + repr(round(Pp, 4)))

'''
(c-iii) Numerically approximate the statistics by repeated sampling.
Generate 10**6 samples, and do this for both x(0) normally distributed,
and uniformly distributed.
'''

# Use above generated samples for x0 being normally and uniformly distributed

# Transform x0 in x1 with nonlinear function and compute resulting statistics
x1_norm, x1_uni = q_c(x0_norm), q_c(x0_uni)
xp_norm, Pp_norm = np.mean(x1_norm), np.var(x1_norm)
xp_uni, Pp_uni = np.mean(x1_uni), np.var(x1_uni)

print('Using repeated sampling with x0 normally distributed, E[x(1)] = '
      + repr(round(xp_norm, 4)) + ' and Var[x(1)] = '
      + repr(round(Pp_norm, 4)))

print('Using repeated sampling with x0 uniformly distributed, E[x(1)] = '
      + repr(round(xp_uni, 4)) + ' and Var[x(1)] = ' + repr(round(Pp_uni, 4)))
```

```
Part (c):
Using techniques from the EKF, E[x(1)] = 3 and Var[x(1)] = 36
Using techniques from the unscented transform, E[x(1)] = 3.0 and Var[x(1)] = 36.0
Using repeated sampling with x0 normally distributed, E[x(1)] = 3.0058 and Var[x(1)] = 36.0035
Using repeated sampling with x0 uniformly distributed, E[x(1)] = 2.9972 and Var[x(1)] = 35.991
```

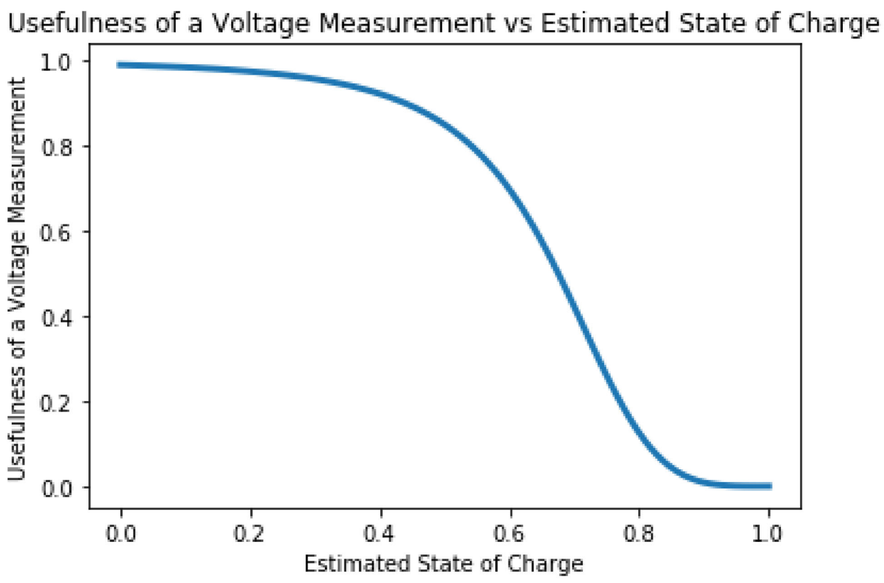
In [ ]:

In [1]:
```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt

'''
A battery's state of charge at time step k is
given by q(k), with q(k) = 1 corresponding to fully
charged and q(k) = 0 depleted. In each time step
(e.g. each hour) k, the battery powers a process
which consumes energy j(k) = j0 + v(k), where j0
is the average amount of energy consumed, and v(k)
is a random deviation, so that
q(k) = q(k-1) - j(k-1).
We have perfect knowledge of the battery's initial
charge, q(0) = 1, and we know that v(k) is white
and normally distributed as given in problem.

we now add a voltage sensor which gives us a noisy
reading of the battery voltage after each time cycle.
The measurement is z(k) = h(q(k)) + w(k) with w(k) normally distributed,
and h(q) is a nonlinear function mapping from current state of charge
to voltage, h(q) = 4 + (q - 1)**3
'''

# (a) Design an extended Kalman filter (EKF) to estimate the state of charge.


'''
Define given system and nonlinear measurement model. Note that process noise
will be set to zero for EKF but is included in function to abide by math
notation. Assume no inputs to system.
'''


def q_sys(q, vk): return q - (j0 + vk)


# Meas noise is included in 'h' defintion here to abide by EKF notation
def h_nlmeas(q, wk): return 4 + (q - 1)**3 + wk


# Define global vars
V, W, j0 = 0.05**2, 0.1**2, 0.1

# EKF initialization, q(0) = 1 with no uncertainty
xm, Pm = 1, 0


# Prior update: Jacobian matrices A and L:

'''
Scalar system model is already linear in q and v, therefore A and L jacobians
are constant for all k and equal to 1.
'''
Ak = 1
Lk = 1

# Define linear KF prediction function


def time_update(xm, Pm):
    xp = q_sys(xm, 0)
    Pp = Ak * Pm * Ak + Lk * V * Lk
    return xp, Pp


# Measurement update: Jacobian matrices H and M
q, w = sp.symbols('q w')

# Meas noise is included in 'h' defintion here to abide by EKF notation
h = 4 + (q - 1)**3 + w

# Take jacobians of nl meas model and lambdify to functions of np arrays
H = sp.lambdify([q, w], h.diff(q))
M = sp.lambdify([q, w], h.diff(w))

# Define linear KF measurement update function


def meas_update(xp, Pp, z):

    # Plug in values for H, L
    Hk = H(xp, 0)
    Mk = M(xp, 0)

    # KF equations
    Kk = Pp * Hk * (Hk * Pp * Hk + Mk * W * Mk)**-1
    xm = xp + Kk * (z - h_nlmeas(xp, 0))
```

```
    Pm = (np.eye(1) - Kk * Hk) * Pp

    return xm, Pm
```

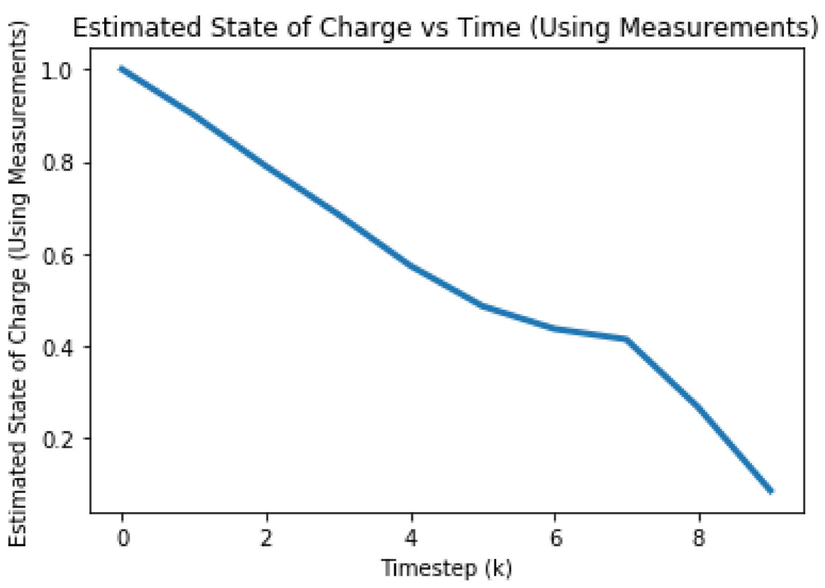
In [2]:
```
'''
(c) Using the variance metric derived in (b), (see handwritten notes), make
a plot of the usefulness of a voltage measurement as a function of the
estimated state of charge, for q in [0 1] (with usefulness as defined in
the subproblem b). Set Pp(k) = 0.1, and W = 0.1. Where is the measurement
most informative? Where is it least informative?
'''
q, Pp_c, W_c = np.linspace(0, 1, 100), 0.1, 0.1


# note that 'H' in this problem is same as lambda function 'H' defined above
usefulness = H(q, 0)**2 * Pp_c / (W + H(q, 0)**2 * Pp_c)

plt.figure(0)
plt.plot(q, usefulness, linewidth=3)
plt.xlabel('Estimated State of Charge')
plt.ylabel('Usefulness of a Voltage Measurement')
plt.title(r'Usefulness of a Voltage Measurement vs Estimated State of Charge')

print('The measurements are most informative for low state of charge values,'
      ' and least informative for high state of charge values. This makes'
      ' sense intuitively because the battery starts at a state of charge'
      ' of q = 1 with no uncertainty. After each succsessive timestep, another'
      ' process uncertainty value, v(k), is imposed on the system, thus'
      ' increasing the cumulative process unceratinty of our battery state.'
      ' Measurements used at this area of high process uncertainty (low q)'
      ' are most useful because the EKF is an optimal estimator, and will'
      ' weight measurement values higher when the process model'
      ' is less reliable.')
```

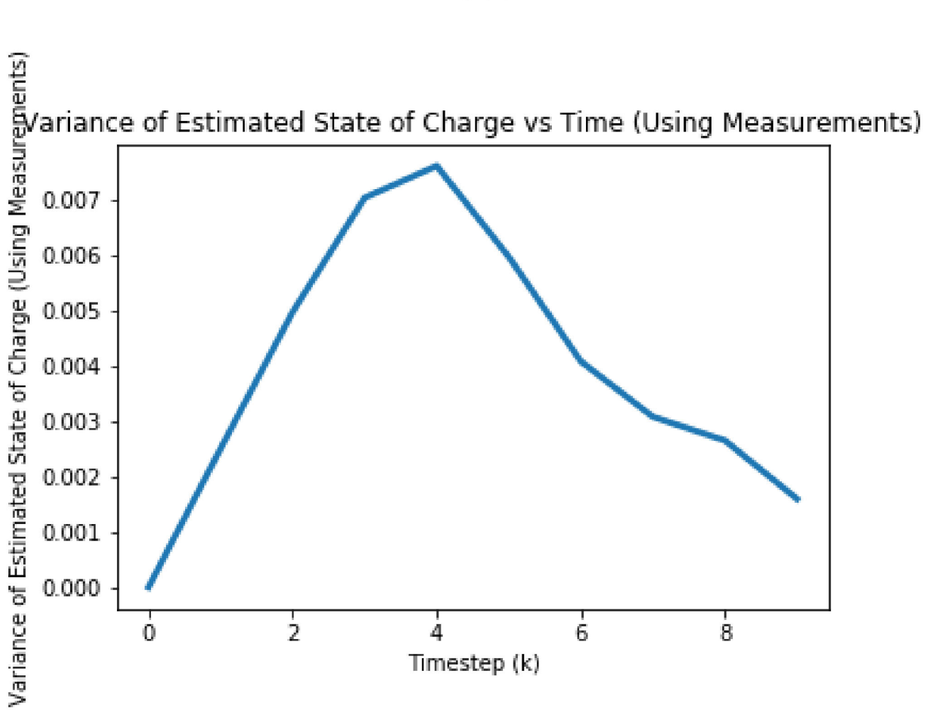
```
The measurements are most informative for low state of charge values, and least informative for high state of charge values. This ma
kes sense intuitively because the battery starts at a state of charge of q = 1 with no uncertainty. After each succsessive timestep,
another process uncertainty value, v(k), is imposed on the system, thus increasing the cumulative process unceratinty of our battery
state. Measurements used at this area of high process uncertainty (low q) are most useful because the EKF is an optimal estimator, a
nd will weight measurement values higher when the process model is less reliable.
```



In [3]:
```
'''
For the remainder of the problem, use the following sequence of measurements
starting at time k = 1 (dummy meas added at k = 0 to abide by math notiation):
'''
z = np.array([0, 4.04, 3.81, 3.95, 3.90, 3.88, 3.88, 3.90, 3.55, 3.18])

'''
(d) Run your extended Kalman filter with this data, and generate two plots:
the estimated state of charge, and the variance of this estimate, across k.
'''

Tf = 9

# Initialze plotting arrays
q_est, var_est = np.zeros(10), np.zeros(10)
q_est[0], var_est[0] = xm, Pm

# Note that EKF is already initialized for k = 0 in (a)
for k in range(1, Tf + 1):

    # Run EKF
    xp, Pp = time_update(xm, Pm)
    xm, Pm = meas_update(xp, Pp, z[k])

    # Store values for plotting
    q_est[k], var_est[k] = xm, Pm

#  Plotting
plt.figure(1)
plt.plot(range(0, Tf + 1), q_est, linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Estimated State of Charge (Using Measurements)')
plt.title(r'Estimated State of Charge vs Time (Using Measurements)')

plt.figure(2)
plt.plot(range(0, Tf + 1), var_est, linewidth=3)
plt.xlabel('Timestep (k)')
plt.ylabel('Variance of Estimated State of Charge (Using Measurements)')
plt.title(r'Variance of Estimated State of Charge vs Time (Using Measurements)')
```

Out[3]:
```
Text(0.5, 1.0, 'Variance of Estimated State of Charge vs Time (Using Measurements)')
```





In [4]:
```
'''
(e) After 9 steps, what would the mean and variance be if you did not have the
voltage measurements. How does this compare to your EKF output?
'''
# Initialze plotting arrays
q_est_nodata, var_est_nodata = np.zeros(10), np.zeros(10)
q_est_nodata[0], var_est_nodata[0] = 1, 0

# Intialize xp, Pp for EKF at k = 0
xp, Pp = 1, 0

for k in range(1, Tf + 1):

    # Run EKF
    xp, Pp = time_update(xp, Pp)

    # Store values for plotting
    q_est_nodata[k], var_est_nodata[k] = xp, Pp

print(
    'After 9 steps, the mean and variance without using voltage measurements'
    ' would be: ' + repr(round(q_est_nodata[Tf], 4)) + ' and: '
    + repr(round(var_est_nodata[Tf], 4)) + ' respectively.')

print(
    'After 9 steps, the mean and variance for our EKF estimate'
    ' would be: ' + repr(round(q_est[Tf], 4)) + ' and: '
    + repr(round(var_est[Tf], 4)) + ' respectively.')
```

```
print('By comparing our EKF output to a system estimation that doesn\'t'
      ' use measurements, we can see that by timestep k = 9, the EKF is able'
      ' to keep the estimate variance significantly lower than that of an'
      ' estimate without measurements. The EKF also gave us an estimate that'
      ' was slightly lower than our no data model.'
      )
```

```
After 9 steps, the mean and variance without using voltage measurements would be: 0.1 and: 0.0225 respectively.
After 9 steps, the mean and variance for our EKF estimate would be: 0.0867 and: 0.0016 respectively.
By comparing our EKF output to a system estimation that doesn't use measurements, we can see that by timestep k = 9, the EKF is able
to keep the estimate variance significantly lower than that of an estimate without measurements. The EKF also gave us an estimate th
at was slightly lower than our no data model.
```

In [ ]:

Problem 4(b). Show that the reduction in variance due to a measurement (ie a measure of its usefulness) can be described as below:

$$\frac{P_p(k) - P_m(k)}{P_p(k)} = \frac{H(k)^2 P_p(k)}{W + H(k)^2 P_p(k)}$$

- $\frac{P_p(k) - P_m(k)}{P_p(k)} = \frac{P_p(k) - (I - K(k)H(k))P_p(k)}{P_p(k)}$

$= I - (I - K(k)H(k)) = K(k)H(k)$

$= P_p(k) H^T(k) (H(k) P_p(k) H^T(k) + M(k) W(k) M^T(k))^{-1} H(k)$

- We have a scalar system, and $M(k) = \partial h / \partial w = 1$

$$\Rightarrow \frac{H(k)^2 P_p(k)}{H(k)^2 P_p(k) + (1) W(k) (1)} = \boxed{\frac{H(k)^2 P_p(k)}{W + H(k)^2 P_p(k)}}$$