Assignment III

Q1. The posterior loss expectation to adopt action $\hat{y}$, given x is

$$\rho(\hat{y}|x) = E_{p(\hat{y}|x)} = E_{p(y|x)}[L(y,\hat{y})]$$

$$= p_0 \cdot L(\hat{y},0) + p_1 \cdot L(\hat{y},1)$$

$$= p_0 \cdot L(\hat{y},0) + (1-p_0) \cdot L(\hat{y},1)$$

$$= L(\hat{y},1) + p_0 (L(\hat{y},0) - L(\hat{y},1))$$

Then,

$$\rho(0|x) = \lambda_{01} - p_0 \lambda_{01}$$

$$\rho(1|x) = p_0 - \lambda_{10}$$

$\rho(0|x)$ and $\rho(1|x)$ are both linear function of p_0, the unique optimal threshold would be $\rho(0|x) = \rho(1|x)$

as example $p_0 = \dfrac{\lambda_{01}}{\lambda_{01}+\lambda_{10}}$ $\quad p_1 = \dfrac{\lambda_{10}}{\lambda_{01}+\lambda_{10}}$

to derive the loss matrix where the threshold is 0.1

$$\lambda_{10} = 1 \quad \lambda_{01} = 9$$

$$p_0 = \frac{\lambda_{01}}{\lambda_{01}+\lambda_{10}} = \frac{9}{9+1} = 0.9$$

$$p_1 = \frac{\lambda_{10}}{\lambda_{10}+\lambda_{01}} = \frac{1}{9+1} = 0.1$$

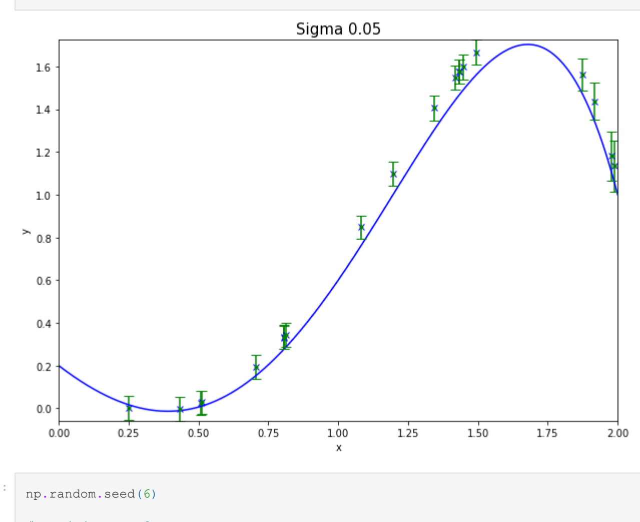
Question 3

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

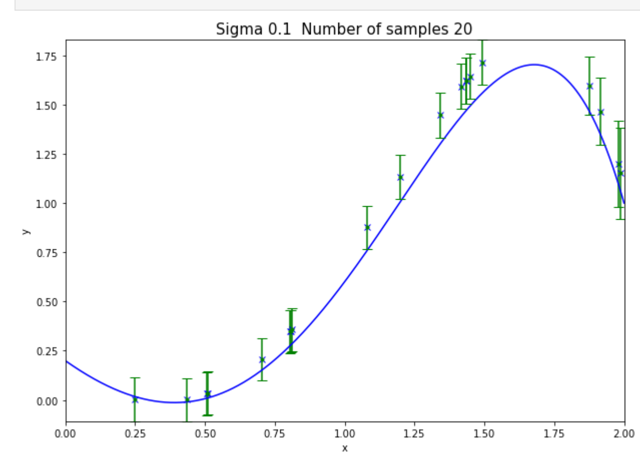
```
In [2]:
def frange(a, b, j):
    while a < b:
        yield a
        a += j
```

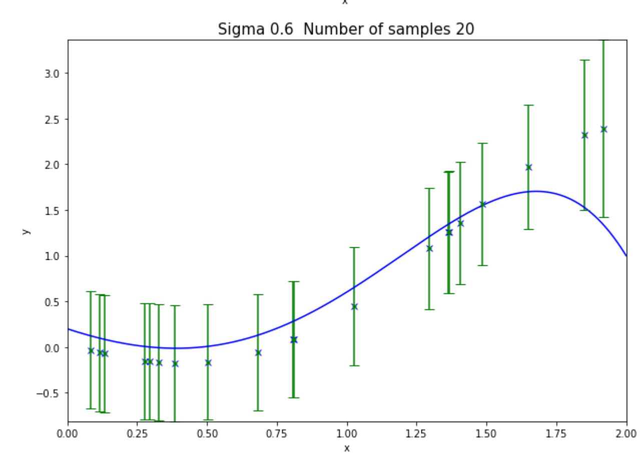
i

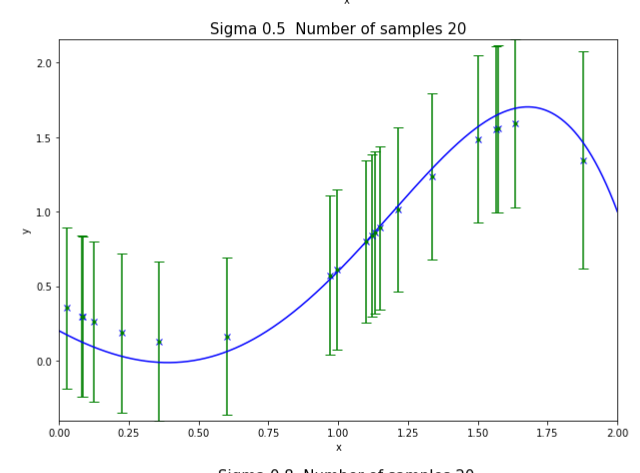
```
In [3]:
np.random.seed(6)

# training samples
N = 20

# signal curve
x = np.array(list(frange(0, 2, 0.0001)))
x = np.reshape(x, newshape=(x.shape[0], 1))
y = 0.2 * np.ones(shape=(x.shape[0],1)) - x + 0.9 * x**2 + 0.7 * x**3 - 0.2 * x**5

# sample interval [a b]
a = 0
b = 2

# samples
x1 = np.array(list(frange(a, b, b/N)))
x1 = np.reshape(x1, newshape=(x1.shape[0], 1))

# noise generation
sigma_n = 0.05
n = math.sqrt(sigma_n) * np.random.randn(N,1)

# theta
theta_tr = np.array([[0.2], [-1], [0.9], [0.7], [-0.2]])

# random theta
theta_ds = np.array([[-0.004], [-10.40], [0.480], [0.095], [-0.085]])
l = theta_tr.shape[0]

# measurement matrix
P_hi = np.ones(shape=(N, 1))
P_hi = np.concatenate((P_hi, np.array(x1)), axis=1)
P_hi = np.concatenate((P_hi, np.array(x1**2)), axis=1)
P_hi = np.concatenate((P_hi, np.array(x1**3)), axis=1)
P_hi = np.concatenate((P_hi, np.array(x1**5)), axis=1)

# noisy observations using the linear model
y1 = np.dot(P_hi, theta_tr) + n

# set the parameters of Gaussian prior
sigma_theta = 0.1
mu_prior = theta_tr

# covariance matrix of Gaussian posterior
sig_post = np.linalg.inv((sigma_theta**-1) * np.eye(l) + (sigma_n**-1) * np.dot(P_hi.conj().transpose(), P_hi))

# posterior mean
mu_post = mu_prior + (sigma_n**-1) * np.dot(np.dot(sig_post, P_hi.conj().transpose()), (y1 - np.dot(P_hi, mu_pr

# linear prediction
Np = 20

# prediction samples
x2 = (b-a) * np.random.rand(Np, 1)

# prediction measurement matrix
P_hip = np.ones(shape=(Np, 1))
P_hip = np.concatenate((P_hip, np.array(x2)), axis=1)
P_hip = np.concatenate((P_hip, np.array(x2 ** 2)), axis=1)
P_hip = np.concatenate((P_hip, np.array(x2 ** 3)), axis=1)
P_hip = np.concatenate((P_hip, np.array(x2 ** 5)), axis=1)

# predicted mean and variance
mu_pred = np.dot(P_hip, mu_post).flatten()
sig_pred_y = np.diag(sigma_n + sigma_n * sigma_theta * np.dot(np.dot(P_hip,np.linalg.inv(sigma_n * np.eye(l) +
                     P_hip.conj().transpose()))
sig_pred_y = np.reshape(sig_pred_y, newshape=(sig_pred_y.shape[0],1)).flatten()

# plot the predictions along the condifence intervals
plt.figure(figsize = (10, 7))


plt.autoscale(enable=True, axis='x', tight=True)
plt.autoscale(enable=True, axis='y', tight=True)

plt.title('Sigma ' + str(sigma_n), fontsize=15)
plt.plot(x, y, 'b')
plt.plot(x2, mu_pred, 'bx')
plt.errorbar(x2, mu_pred, sig_pred_y, fmt='g.', capsize=5)

plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

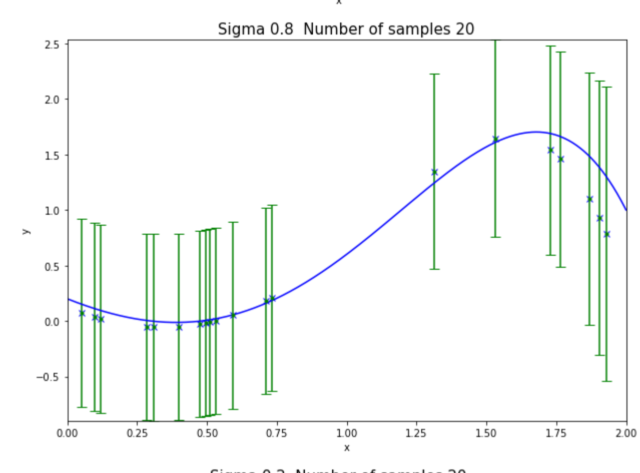


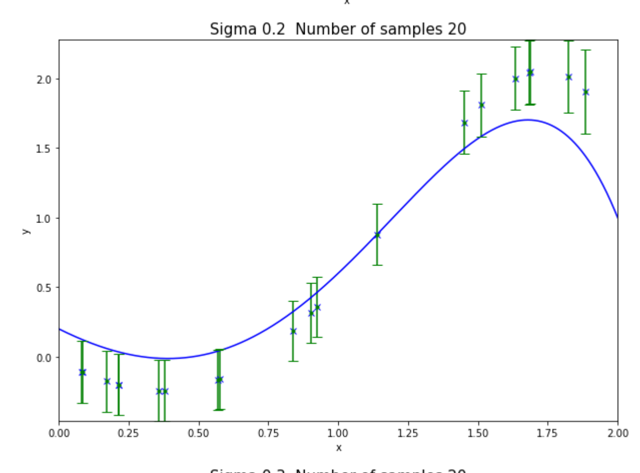
```
In [4]:
np.random.seed(6)

# training samples
N = np.array([20,500])

for j in range(2):

    # signal curve
    x = np.array(list(frange(0, 2, 0.0001)))
    x = np.reshape(x, newshape=(x.shape[0], 1))
    y = 0.2 * np.ones(shape=(x.shape[0],1)) - x + 0.9 * x**2 + 0.7 * x**3 - 0.2 * x**5

    # sample interval [a b]
    a = 0
    b = 2

    # samples
    x1 = np.array(list(frange(a, b, b/N[j])))
    x1 = np.reshape(x1, newshape=(x1.shape[0], 1))

    # noise generation with different sigma values
    sigma_narray = np.array([0.1,0.6,0.5,0.8,0.2,0.3])

    for i in range(6):

        n = math.sqrt(sigma_narray[i]) * np.random.randn(N[j],1)

        # theta
        th = np.array([[0.2], [-1], [0.9], [0.7], [-0.2]])

        # random theta
        theta_ds = np.array([[-0.004], [-10.40], [0.480], [0.095], [-0.085]])
        l = th.shape[0]

        # compute the measurement matrix
        P_hi = np.ones(shape=(N[j], 1))
        P_hi = np.concatenate((P_hi, np.array(x1)), axis=1)
        P_hi = np.concatenate((P_hi, np.array(x1**2)), axis=1)
        P_hi = np.concatenate((P_hi, np.array(x1**3)), axis=1)
        P_hi = np.concatenate((P_hi, np.array(x1**5)), axis=1)

        # noisy observations with the linear model
        y1 = np.dot(P_hi, th) + n

        # set parameters of Gaussian prior
        sigma_theta = 0.1


        mu_prior = th

        # covariance matrix of Gaussian posterior
        sig_post = np.linalg.inv((sigma_theta**-1) * np.eye(l) + (sigma_narray[i]**-1) * np.dot(P_hi.conj().tra

        # posterior mean
        mu_post = mu_prior + (sigma_narray[i]**-1) * np.dot(np.dot(sig_post, P_hi.conj().transpose()), (y1 - np.

        # linear prediction
        Np = 20

        # prediction samples
        x2 = (b-a) * np.random.rand(Np, 1)

        # prediction measurement matrix
        P_hip = np.ones(shape=(Np, 1))
        P_hip = np.concatenate((P_hip, np.array(x2)), axis=1)
        P_hip = np.concatenate((P_hip, np.array(x2 ** 2)), axis=1)
        P_hip = np.concatenate((P_hip, np.array(x2 ** 3)), axis=1)
        P_hip = np.concatenate((P_hip, np.array(x2 ** 5)), axis=1)

        # predicted mean and variance
        mu_pred = np.dot(P_hip, mu_post).flatten()
        sig_pred_y = np.diag(sigma_narray[i] + sigma_narray[i] * sigma_theta * np.dot(np.dot(P_hip,np.linalg.in
                             P_hip.conj().transpose()))
        sig_pred_y = np.reshape(sig_pred_y, newshape=(sig_pred_y.shape[0],1)).flatten()

        # plot the predictions along the condifence intervals
        plt.figure(figsize = (10, 7))

        plt.autoscale(enable=True, axis='x', tight=True)
        plt.autoscale(enable=True, axis='y', tight=True)

        plt.title('Sigma ' + str(sigma_narray[i]) + ' Number of samples ' + str(N[j]), fontsize=15)
        plt.plot(x, y, 'b')
        plt.plot(x2, mu_pred, 'bx')
        plt.errorbar(x2, mu_pred, sig_pred_y, fmt='g.', capsize=5)

        plt.xlabel('x')
        plt.ylabel('y')
        plt.show()
```

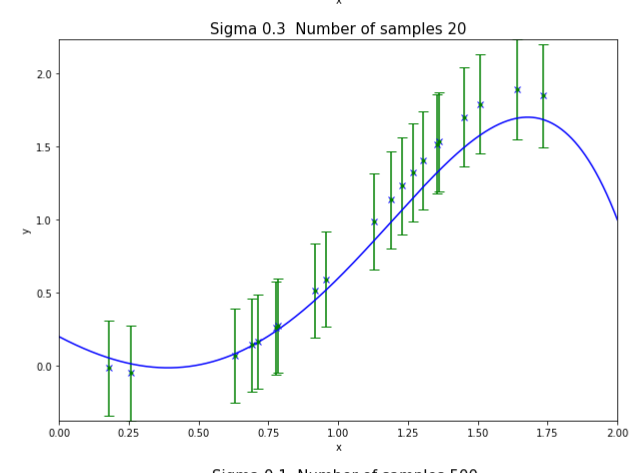


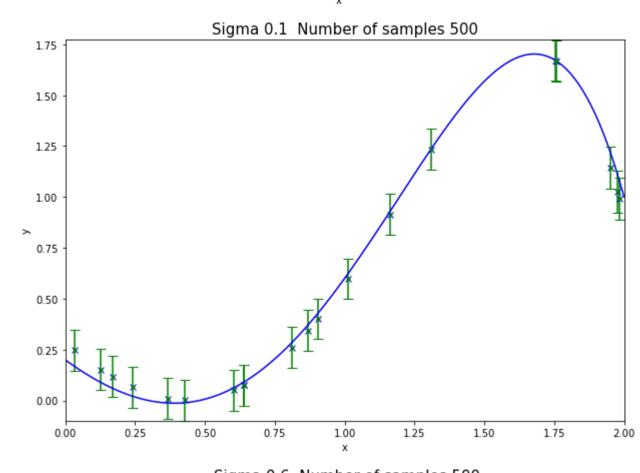


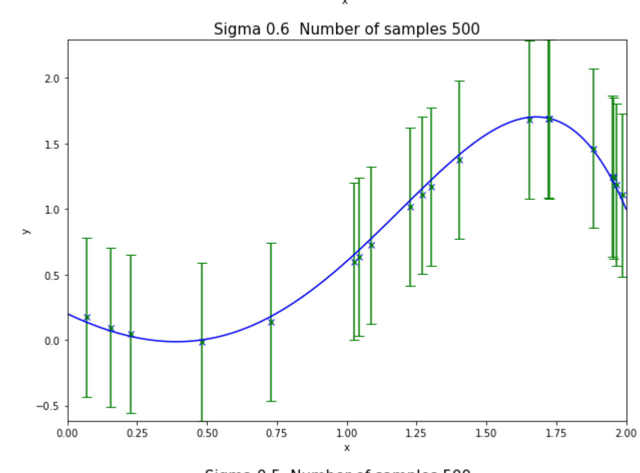


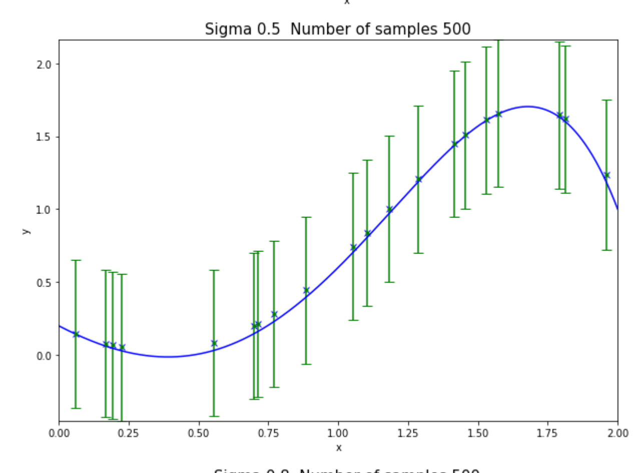


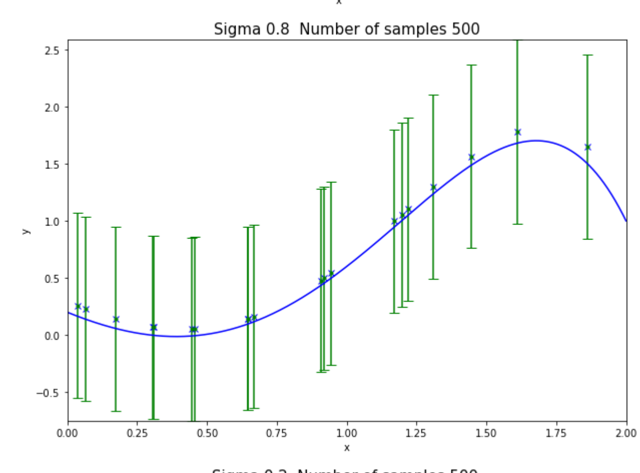


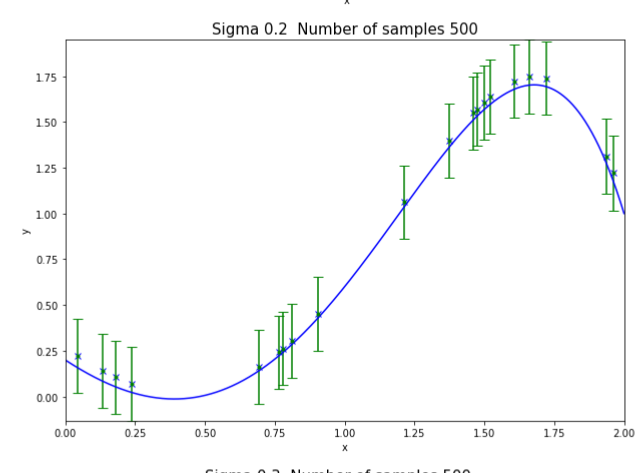


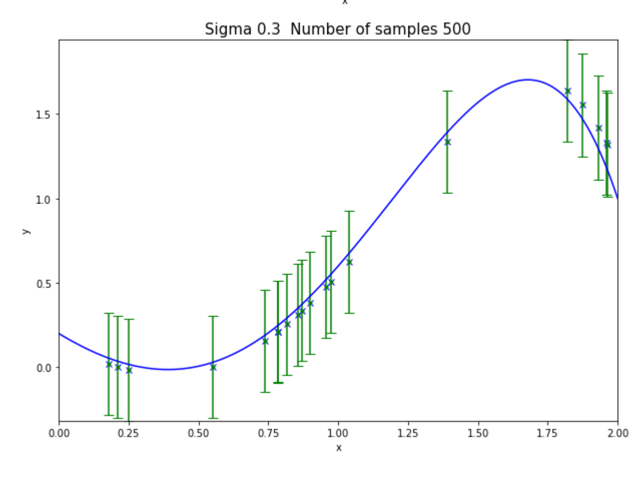






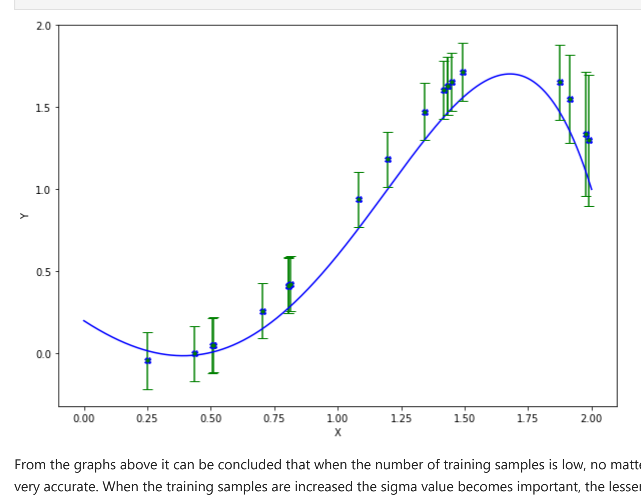






ii

```
In [5]:
np.random.seed(6)

# training samples
N = 20

# signal curve
X = np.array(list(frange(0, 2, 0.0001)))
X = np.reshape(X, newshape=(X.shape[0], 1))
Y = 0.2 * np.ones(shape=(X.shape[0],1)) - X + 0.9 * X**2 + 0.7 * X**3 - 0.2 * X**5

# sample interval [a b]
a = 0
b = 2

# samples
X1 = np.array(list(frange(a, b, b/N)))
X1 = np.reshape(X1, newshape=(X1.shape[0], 1))

# noise generation
sigma_n = 0.15
n = math.sqrt(sigma_n) * np.random.randn(N,1)

# theta
theta_tr = np.array([[0.2], [-1], [0.9], [0.7], [-0.2]])

# random theta
theta_ds = np.array([[-0.005], [-10.60], [0.470], [0.097], [-0.083]])
l = theta_tr.shape[0]

# measurement matriX
P_hi = np.ones(shape=(N, 1))
P_hi = np.concatenate((P_hi, np.array(X1)), axis=1)
P_hi = np.concatenate((P_hi, np.array(X1**2)), axis=1)
P_hi = np.concatenate((P_hi, np.array(X1**3)), axis=1)
P_hi = np.concatenate((P_hi, np.array(X1**5)), axis=1)

# noisy observations with the linear model
Y1 = np.dot(P_hi, theta_tr) + n

# set the parameters of Gaussian prior
sigma_theta = 2
mu_prior = theta_tr

# covariance matrix of Gaussian posterior
sig_post = np.linalg.inv((sigma_theta**-1) * np.eye(l) + (sigma_n**-1) * np.dot(P_hi.conj().transpose(), P_hi))

# posterior mean
mu_post = mu_prior + (sigma_n**-1) * np.dot(np.dot(sig_post, P_hi.conj().transpose()), (Y1 - np.dot(P_hi, mu_pr

# linear prediction
Np = 20

# prediction samples
X2 = (b-a) * np.random.rand(Np, 1)

# prediction measurement matriX
P_hip = np.ones(shape=(Np, 1))
P_hip = np.concatenate((P_hip, np.array(X2)), axis=1)
P_hip = np.concatenate((P_hip, np.array(X2 ** 2)), axis=1)
P_hip = np.concatenate((P_hip, np.array(X2 ** 3)), axis=1)
P_hip = np.concatenate((P_hip, np.array(X2 ** 5)), axis=1)

# predicted mean and variance
mu_pred = np.dot(P_hip, mu_post).flatten()
sig_pred_y = np.diag(sigma_n + sigma_n * sigma_theta * np.dot(np.dot(P_hip,np.linalg.inv(sigma_n * np.eye(l) +
                     P_hip.conj().transpose()))
sig_pred_y = np.reshape(sig_pred_y, newshape=(sig_pred_y.shape[0],1)).flatten()

# plot the predictions along the condifence intervals
plt.figure(figsize = (10, 7))


plt.autoscale(enable=True, axis='X', tight=True)
plt.autoscale(enable=True, axis='Y', tight=True)

plt.plot(X, Y, 'b')
plt.plot(X2, mu_pred, 'bX')
plt.errorbar(X2, mu_pred, sig_pred_y, fmt='g.', capsize=5)

plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



From the graphs above it can be concluded that when the number of training samples is low, no matter the value of sigma it will not be very accurate. When the training samples are increased the sigma value becomes important. The lower the value the points will fit more to the line. Another conclusion is that the bigger sigma is, the longer the error bars would be independently of the number of training samples.

Question 3.4

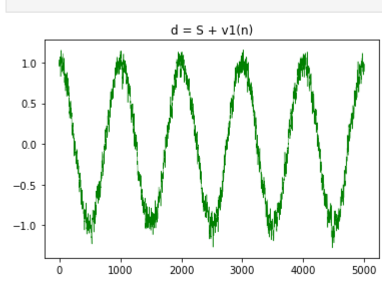
```
In [1]:
import numpy as np
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

.1 & .2

```
In [2]:
np.random.seed(10)
N = 5000 #data samples
w0 = 2 * 10**(-3) * np.pi
S = np.cos(w0 * np.arange(N)).reshape(-1,1) #signal
sigma = np.sqrt(0.0025)
noise = sigma * np.random.randn(N,1)


a1 = 0.8
v1 = np.zeros((N,1)) #initialize AR procedures
for i in range(1,N):
    v1[i] = a1 * v1[i-1] + noise[i]
```

.3

```
In [3]:
x_axis = np.arange(0,N,1)
d = S + v1 #contaminated signal
plt.figure(1)
plt.title("d = S + v1(n)")
plt.plot(x_axis,d, 'g',lw=0.5)
plt.show()
```



.4

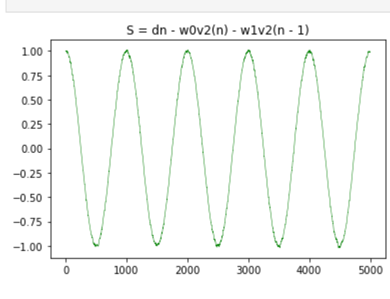
```
In [4]:
a2 = 0.75
v2 = np.zeros((N,1))
for i in range(1,N):
    v2[i] = a2 * v2[i-1] + noise[i]
```

.5

```
In [5]:
def window(a,width):
    list_A = []
    a_ext = np.hstack((a[-width:],a))
    for index in range(width+1,len(a_ext)+1):
        b = a_ext[index-width:index]
        list_A.append(b[::-1])
    return np.asarray(list_A).squeeze()
```

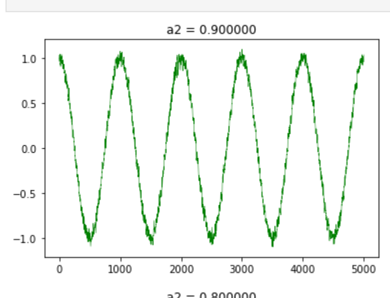
```
In [6]:
X = window(v2.ravel(), width=2)
w1 = np.linalg.inv(np.dot(X.T,X)).dot(X.T).dot(d)
d_est = X.dot(w1)
```

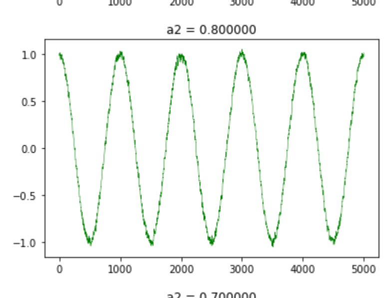
```
In [7]:
x_axis = np.arange(0,N,1)
plt.figure(1)
plt.plot(x_axis,S+v1-d_est, 'g',lw=0.5)
plt.title('S = dn - w0v2(n) - w1v2(n - 1)')
plt.show()
```

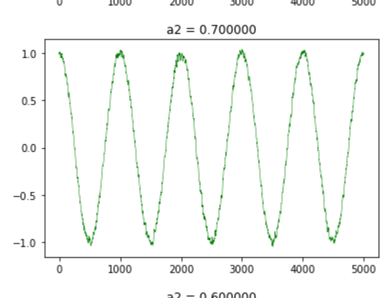


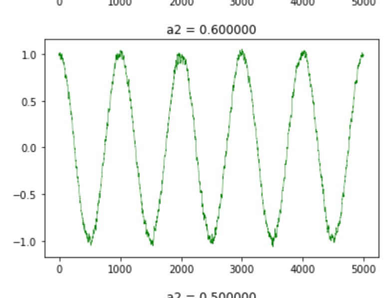
.6

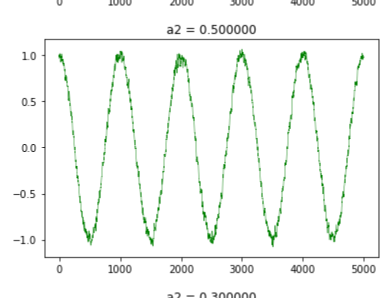
```
In [8]:
sigma = np.sqrt(0.0025)
noise = sigma * np.random.randn(N,1)
a1 = 0.8
a2 = np.array([0.9,0.8,0.7,0.6,0.5,0.3])
for j in range(len(a2)):
    v1 = np.zeros((N,1)) #initialize AR procedures
    v2 = np.zeros((N,1))
    for i in range(1,N):
        v1[i] = a1 * v1[i-1] + noise[i]
        v2[i] = a2[j] * v2[i-1] + noise[i]
    d = S + v1
    X = window(v2.ravel(), width=2)
    w1 = np.linalg.inv(np.dot(X.T,X)).dot(X.T).dot(d)
    d_est = X.dot(w1)
    x_axis = np.arange(0,N,1)
    plt.figure(j)
    plt.plot(x_axis,S+v1-d_est, 'g',lw=0.5)
    plt.title("a2 = %f" %a2[j])
```

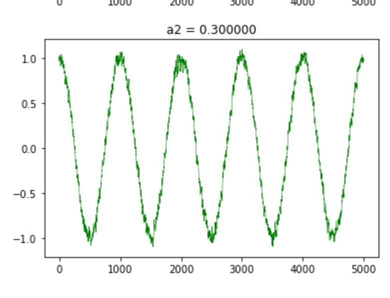






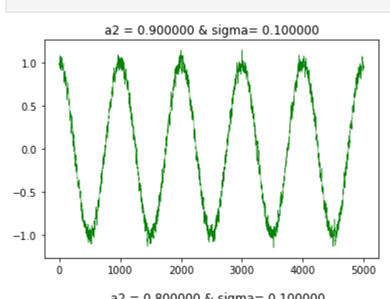


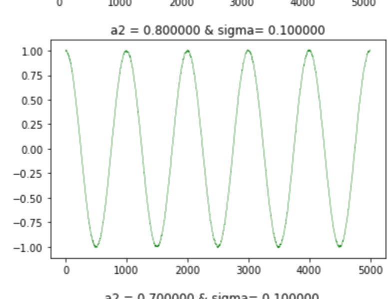


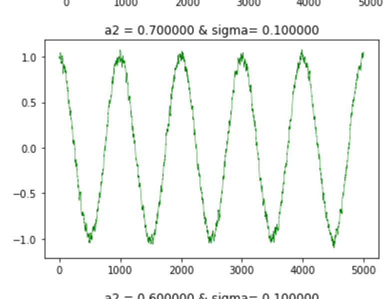


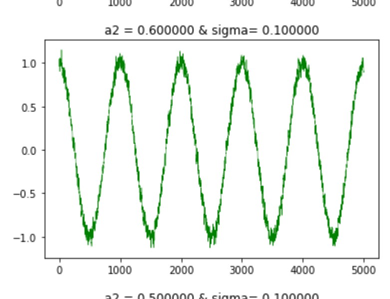
.7

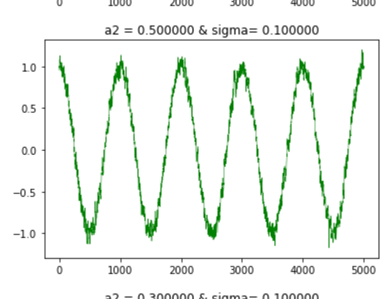
```
In [9]:
sigma = np.sqrt(np.array([0.01,0.05,0.1,0.2,0.5]))
a1 = 0.8
a2 = np.array([0.9,0.8,0.7,0.6,0.5,0.3])
for i in range(len(sigma)):
    for j in range(len(a2)):
        noise = sigma[i] * np.random.randn(N,1)
        #initialize AR procedures
        v1 = np.zeros((N,1))
        v2 = np.zeros((N,1))
        for k in range(1,N):
            v1[k] = a1 * v1[k-1] + noise[k]
            v2[k] = a2[j] * v2[k-1] + noise[k]
        d = S + v1
        X = window(v2.ravel(), width=2)
        w1 = np.linalg.inv(np.dot(X.T,X)).dot(X.T).dot(d)
        d_est = X.dot(w1)
        x_axis = np.arange(0,N,1)
        plt.figure()
        plt.plot(x_axis,S+v1-d_est, 'g',lw=0.5)
        plt.title("a2 = %f" %a2[j]+ " & sigma= %f" %sigma[i])
```

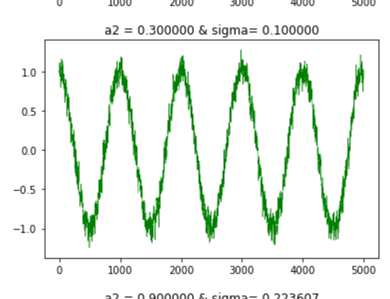


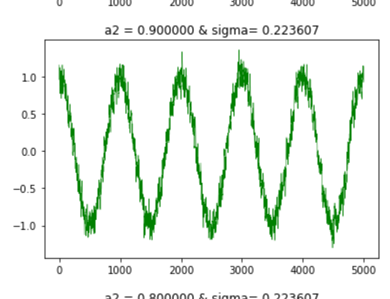


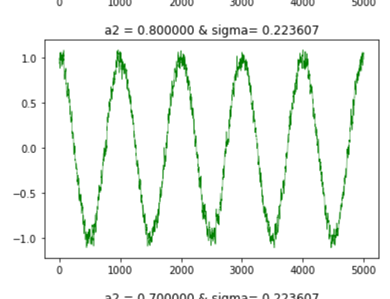


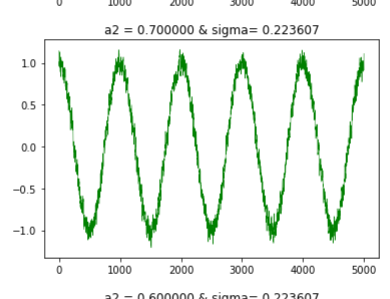


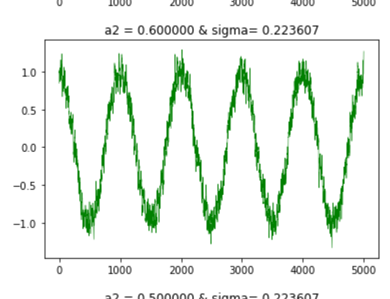


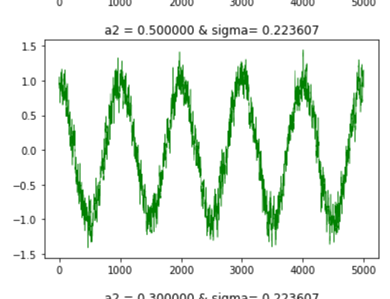


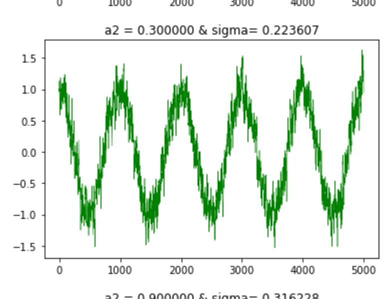


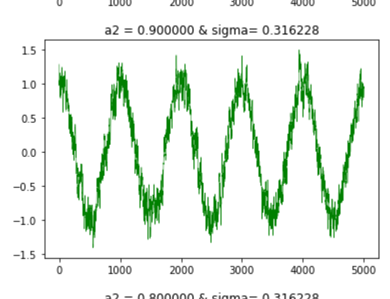


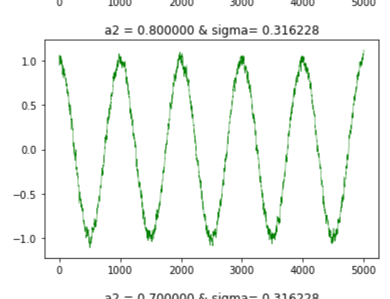


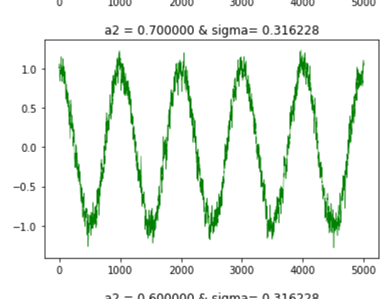


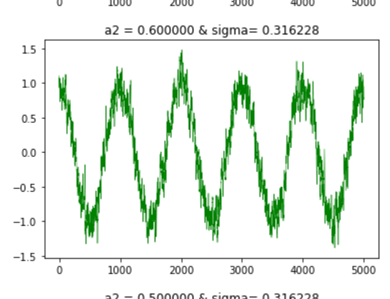


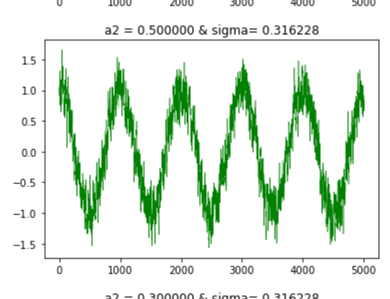


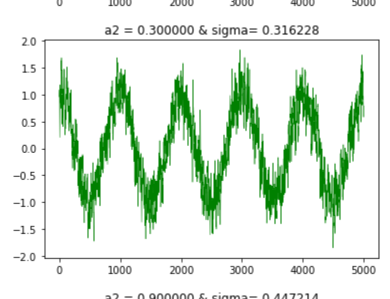


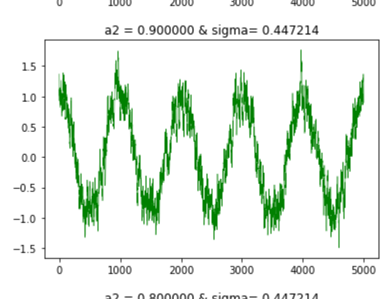


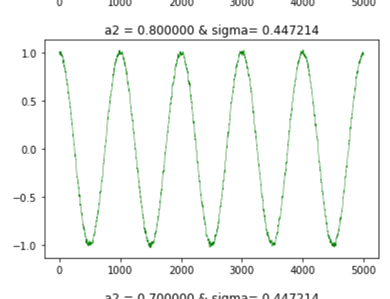


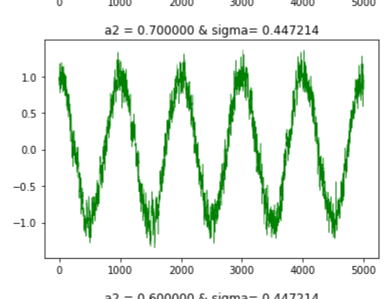


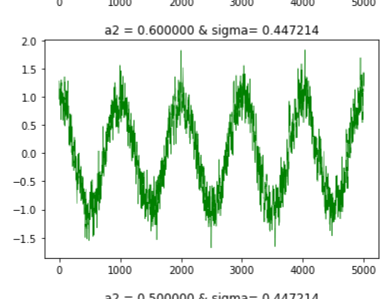


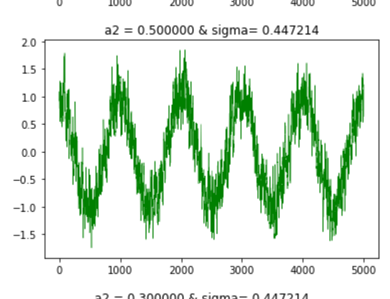


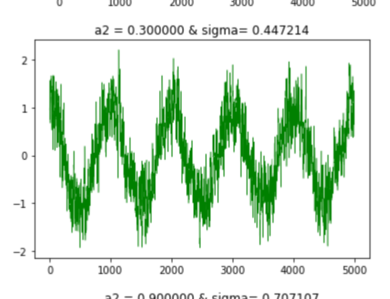


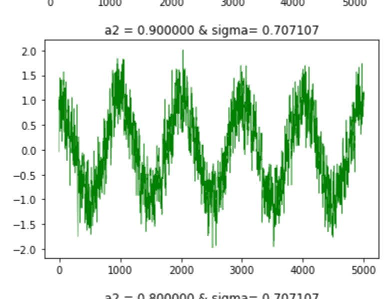


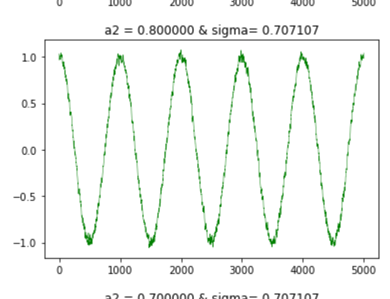


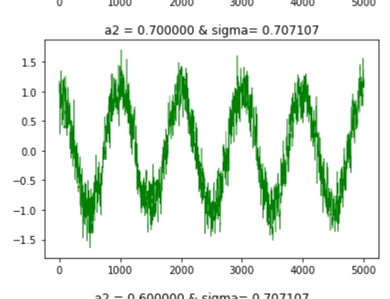


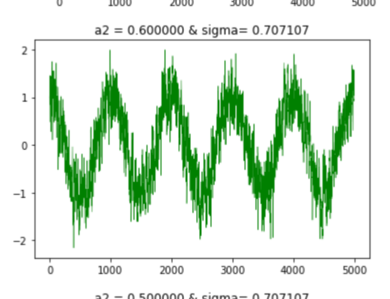


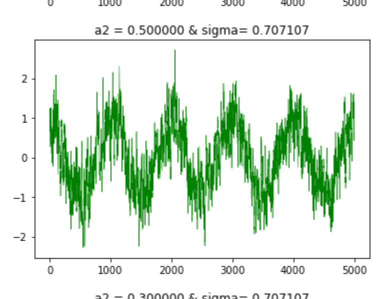


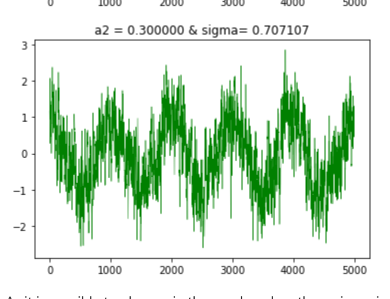












As it is possible to observe in the graphs, when the variance increases, the noise in the signal increases as well making it hard for the algorithm to solve it. In the other hand when a2 is equal to a1 the noise is canceled completely and when it decreases the noise propagates more.