E1 244: Detection and Estimation Theory **(Due: 22/05/20)**

Assignment #2

Instructor: Sundeep Chepuri *Name:* Vineeth S, *SR No.:* 16543

We are given $x(n)$ is a zero-mean real-valued wide-sense stationary (WSS) random process with autocorrelation sequence $r(k)$. More specifically, the signal $x(n)$ is Autoregressive process of order 1 that is generated by the difference equation,

$$x(n) = \alpha x(n-1) + w(n)$$

where $w(n)$ is white noise with variance σ_w^2. We have access to the data set $\mathbf{x} = [x(0), x(1), ..., x(n-1)]^T$ or its noisy version. We want to estimate or interpolate $x(n_0)$ where n_0 is the index of the sensor which is faulty.

PART A: Problem 1 LMMSE (Wiener) interpolator

(Solution) Consider the class of LMMSE interpolators given,

$$\begin{aligned}\hat{x}(n_0) &= \sum_{\substack{i=0\\ i\neq n_0}}^{N-1} a_i x(i)\\ &= \underline{a}^T \underline{x}\end{aligned}$$

where,

$$\underline{a} = \begin{bmatrix} a_0 \\ a_1 \\ .. \\ a_{n-1} \\ a_{n+1} \\ .. \\ a(N-1) \end{bmatrix} \qquad \underline{x} = \begin{bmatrix} x_0 \\ x_1 \\ .. \\ x_{n-1} \\ x_{n+1} \\ .. \\ x(N-1) \end{bmatrix}$$

We have the Bayesian Mean Squared Error (BMSE) as,

$$\begin{aligned}BMSE &= E[(x(n_0) - \hat{x}(n_0))^2]\\ &= E[(x(n_0) - \underline{a}^T \underline{x})^2]\\ &= E[x(n_0)x(n_0)^T - x(n_0)\underline{x}^T \underline{a} - \underline{a}^T \underline{x} x(n_0) + \underline{a}^T \underline{x}\underline{x}^T \underline{a}]\end{aligned}$$

Differentiating BMSE and equating to 0 to find the minimum we get the Wiener-Hopf equations,

$$\frac{\partial BMSE}{\partial \underline{a}} = E[-2x(n_0)\underline{x} + 2\underline{x}\underline{x}^T \underline{a}] = 0$$

$$E[\underline{x}\underline{x}^T]\underline{a} = E[x(n_0)\underline{x}]$$
$$\mathcal{R}\underline{a} = \underline{r}$$
$$\underline{a} = \mathcal{R}^{-1}\underline{r}$$

where,

$$\mathcal{R} = \begin{bmatrix} r(0) & r(1) & .. & r(n_0-1) & r(n_0+1) & .. & r(N-1) \\ r(1) & r(0) & .. & .. & .. & .. & r(N-2) \\ .. & .. & .. & .. & .. & .. & .. \\ r(n_0-1) & .. & .. & .. & .. & .. & .. \\ r(n_0+1) & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. \\ r(N-1) & .. & .. & .. & .. & .. & r(0) \end{bmatrix} \qquad \underline{r} = \begin{bmatrix} r(n_0) \\ r(n_0-1) \\ .. \\ r(1) \\ r(1) \\ .. \\ r(N-n_0-1) \end{bmatrix}$$

We have,

$$\begin{aligned} r(0) &= E[x(n)x(n)] \\ &= E[(\alpha x(n-1) + w[n])(\alpha x(n-1) + w[n])] \\ &= E[\alpha^2 x[n-1]x[n-1] + w[n]w[n]] \\ &= \alpha^2 r(0) + \sigma_w^2 \\ r(0) &= \frac{\sigma_w^2}{1-\alpha^2} \\ r(k) &= E[x(n)x(n-k)] \\ &= E[(\alpha x(n-1) + w[n])x(n-k)] \\ &= \alpha r(k-1) \end{aligned}$$

$$\mathcal{R} = \frac{\sigma_w^2}{1-\alpha^2}\begin{bmatrix} 1 & \alpha & .. & \alpha^{n_0-1} & \alpha^{n_0+1} & .. & \alpha^{N-1} \\ \alpha & 1 & .. & .. & .. & .. & \alpha^{N-2} \\ .. & .. & .. & .. & .. & .. & .. \\ \alpha^{n_0-1} & .. & .. & .. & .. & .. & .. \\ \alpha^{n_0+1} & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. \\ \alpha^{N-1} & .. & .. & .. & .. & .. & 1 \end{bmatrix} \qquad \underline{r} = \frac{\sigma_w^2}{1-\alpha^2}\begin{bmatrix} \alpha^{n_0} \\ \alpha^{n_0-1} \\ .. \\ \alpha \\ \alpha \\ .. \\ \alpha^{N-n_0-1} \end{bmatrix}$$

Next let's look at the BMSE value. Substituting for $\underline{a}$, We have,

$$\begin{aligned} BMSE &= E[x(n_0)x(n_0) - 2\underline{a}^T\underline{x}x(n_0) + \underline{a}^T\underline{x}\underline{x}^T\underline{a}] \\ &= r(0) - 2E[\underline{r}^T\mathcal{R}^{-T}\underline{x}x(n_0)] + E[\underline{r}^T\mathcal{R}^{-T}\underline{x}\underline{x}^T\mathcal{R}^{-1}\underline{r}] \\ &= r(0) - 2\underline{r}^T\mathcal{R}^{-T}\underline{r} + \underline{r}^T\mathcal{R}^{-T}\underline{r} \\ &= r(0) - \underline{r}^T\mathcal{R}^{-T}\underline{r} \end{aligned}$$

PART A: Problem 2 LMMSE with $x(n_0 - 1)$ and $x(n_0 + 1)$

(Solution) In this problem, we wish to interpolate $x(n_0)$ using only $x(n_0 - 1)$ and $x(n_0 + 1)$ as,

$$x(n_0) = a_1 x(n_0 - 1) + a_2 x(n_0 + 1)$$

This can be easily obtained by restricting the coefficients other than that of $x(n_0 - 1)$ and $x(n_0 + 1)$ to 0 in the Wiener-Hopf equations obtained in Problem 1.Consider the first and last Wiener-Hopf equations obtained from the matrix,

$$a_0 r(0) + a_1 r(1) + .. + a_{n_0-1} r(n_0 - 1) + a_{n_0+1} r(n_0 + 1) + .. + a_{N-1} r(N - 1) = r(n_0)$$
$$a_0 r(N - 1) + a_1 r(N - 2) + .. + a_{n_0-1} r(N - (n_0 - 1) - 1) + a_{n_0+1} r(N - (n_0 + 1) - 1) + .. + a_{N-1} r(0) = r(N - n_0 - 1)$$

Restricting other coefficients to 0 we have,

$$a_{n_0-1} r(n_0 - 1) + a_{n_0+1} r(n_0 + 1) = r(n_0)$$
$$a_{n_0-1} r(N - (n_0 - 1) - 1) + a_{n_0+1} r(N - (n_0 + 1) - 1) = r(N - n_0 - 1)$$

Substituting for autocorrelations,

$$a_{n_0-1}\alpha^{n_0-1} + a_{n_0+1}\alpha^{n_0+1} = \alpha^{n_0}$$
$$\implies a_{n_0-1} + \alpha^2 a_{n_0+1} = \alpha$$
$$a_{n_0-1}\alpha^{N-(n_0-1)-1} + a_{n_0+1}\alpha^{N-(n_0+1)-1} = \alpha^{N-n_0-1}$$
$$\implies \alpha^2 a_{n_0-1} + a_{n_0+1} = \alpha$$

On solving we have,

$$a_{n_0-1} = a_{n_0+1} = \frac{\alpha}{1 + \alpha^2}$$

From Problem 1 we have,

$$BMSE = r(0) - \underline{r}^T \mathcal{R}^{-T} \underline{r}$$

However, in this case we not using all observations. Hence following the same procedure as in Problem 1 we get,

$$BMSE = r(0) - \underline{r}^T \mathcal{R}^{-T} \underline{r}$$

where,

$$\mathcal{R} = \begin{bmatrix} r(0) & r(2) \\ r(2) & r(0) \end{bmatrix} = \frac{\sigma_w^2}{1 - \alpha^2} \begin{bmatrix} 1 & \alpha^2 \\ \alpha^2 & 1 \end{bmatrix} \qquad \underline{r} = \begin{bmatrix} r(1) \\ r(1) \end{bmatrix} = \frac{\sigma_w^2}{1 - \alpha^2} \begin{bmatrix} \alpha \\ \alpha \end{bmatrix}$$

PART A: Problem 3 Kalman filter

(Solution) Suppose we make the measurements of the process as,

$$y(n) = x(n) + v(n) \qquad\qquad n = 0, 1, ..., n_0 - 1$$

where $v(n)$ is white noise having zero mean and variance σ_v^2 . SO, we have,

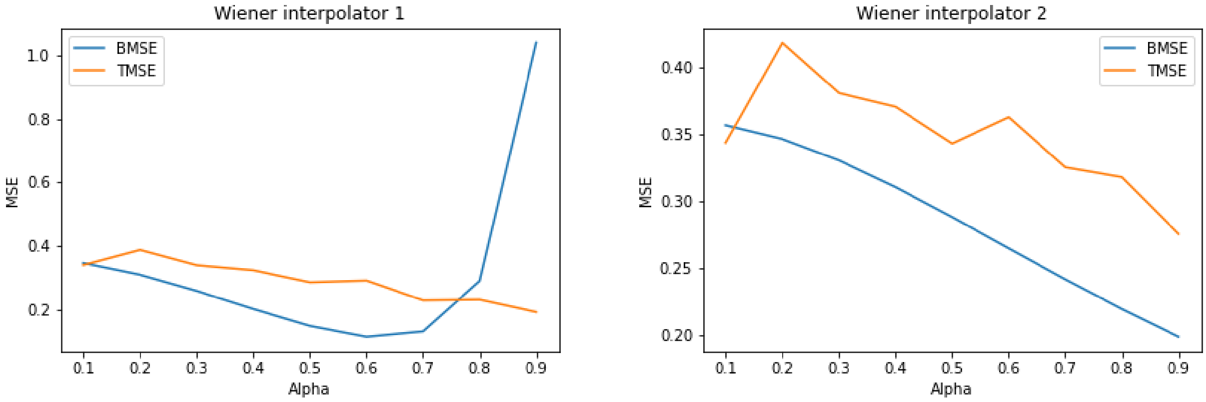
$$x(n) = \alpha x(n-1) + w(n)$$
$$y(n) = x(n) + v(n)$$

which implies that $A = \alpha$ and $C = 1$. Using these values in Kalman filter equations we have,

$$\hat{x}(n|n-1) = \alpha x(n-1|n-1)$$
$$P(n|n-1|) = \alpha^2 P(n-1|n-1) + \sigma_w^2$$
$$K(n) = P(n|n-1)(\sigma_v^2 + P(n|n-1))^{-1}$$
$$x(n|n) = \hat{x}(n|n-1) + K(n)(y(n) - \hat{x}(n|n-1))$$
$$P(n|n) = (1 - K(n))P(n|n-1)$$

where the symbols have their usual meaning.

PART B: Wiener interpolators



(a) MSE vs α for Wiener interpolator 1

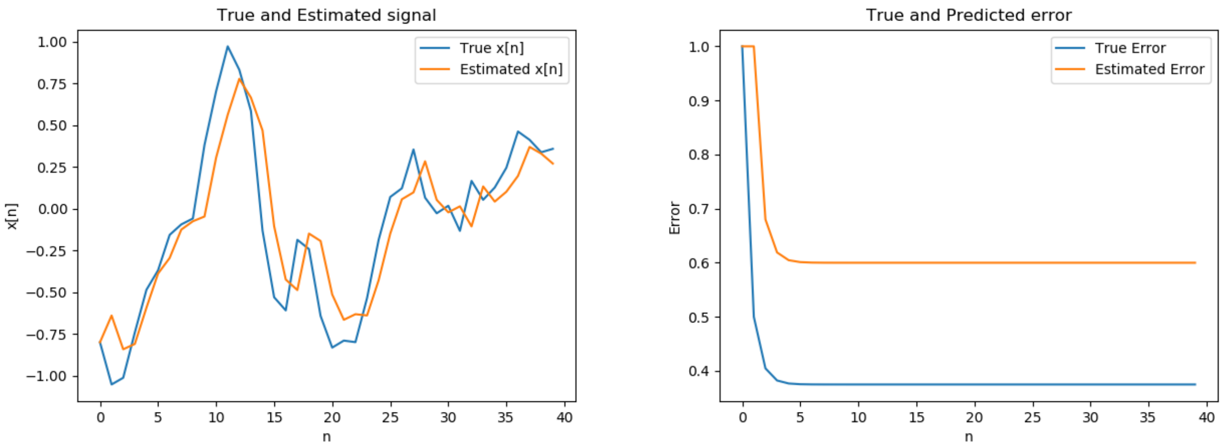
(b) MSE vs α for Wiener interpolator 2

Figure 1: Mean Squared Error vs α for Wiener interpolator

From the plot Figure(1a), we can see that the Theoretical Mean Squared Error (TMSE) decreases with the increase in α. At the same time, Bayesian Mean Squared Error (BMSE) decreases initially with α and after a certain value (0.6) it starts increasing. The MSE values might be high for low α due to numerical precision instabilities. We can also observe that the BMSE is gnerally low than TMSE for a reasonable range of α. This interpolator seems to work better compared to its other version for low values of α.

From the plot Figure(1b), we can see that both the Theoretical Mean Squared Error (TMSE) and Bayesian Mean Squared Error (BMSE) decreases with the increase in α. The MSE values might be high for low α due to numerical precision instabilities. We can also observe that the BMSE is gnerally low than TMSE for the whole range of α. We can see that this interpolator seems to work better compared to its other version for high values of α.

PART B: Kalman filter



(a) True and Estimated signal

(b) True and Predicted error

Figure 2: Kalman filter

From the plot Figure(2a) we can observe the true signal and estimated signal varying with n. Due to the random initializations, the estimated signal and true signal vary for very low values of n. But with growing n, the estimated signal tracks with true signal well. From plot Figure(2b), we can observe that the true and estimated error saturate after some value for n, but the estimated error is higher compared to error.

PART B: Problem 3 Comparison of causal Wiener filter and Kalman filter

For a particular generated observation sequence $x(n)$:

The prediction error for Kalman filter was 0.3750. The prediction error for causal Wiener filter was 0.3600. Prediction error for Kalman filter is slightly higher than that of Wiener filter.

PART C: Appendices (Python codes)

The entire program is written in six python files. The codes are provided below and are also available at https://github.com/vineeths96/Interpolation-of-faulty-sensor GitHub repository. (Repository access is private as of now. Access can me made available, if necessary).

wiener_interpolator.py

```python
import numpy as np
from parameters import *


def wiener_interpolator1(x, n_0, alpha):
    """
    Wiener interpolator using all observations
    :param x: Observation vector except at n_0
    :param n_0: The time instant of interpolation
    :param alpha: Value of alpha
    :return: The predicted value at n_0 and BMSE
    """

    R = np.zeros([N - 1, N - 1])
    r = np.zeros(N - 1)
    a = np.zeros(N - 1)

    for i in range(N - 1):
        for j in range(N - 1):
            if np.abs(i - j) < n_0:
                R[i, j] = alpha ** np.abs(i - j)
            else:
                R[i, j] = alpha ** np.abs(i - j + 1)

    for i in range(N - 1):
        if i < n_0:
            r[i] = alpha ** (n_0 - i)
        else:
            r[i] = alpha ** (i - n_0 + 1)

    R_inv = np.linalg.inv(R)
    a = R_inv @ r
    x_n_0 = np.dot(a, x)

    r_0 = sigma_w ** 2 / (1 - alpha ** 2)
    BMSE = r_0 - np.transpose(r) @ R_inv @ r

    return x_n_0, BMSE


def wiener_interpolator2(x, n_0, alpha):
    """
    Wiener interpolator using observations x[n_0 - 1] and x[n_0 + 1]
    :param x: Observation vector except at n_0
    :param n_0: The time instant of interpolation
    :param alpha: Value of alpha
    :return: The predicted value at n_0 and BMSE
    """
```

```python

    x_n_0 = alpha / (1 + alpha ** 2) * (x[n_0 - 1] + x[n_0 + 1])

    r_0 = sigma_w ** 2 / (1 - alpha ** 2)
    r_1 = alpha * r_0
    r_2 = alpha * r_1

    r = np.array([r_1, r_1])
    R = np.array([[r_0, r_2], [r_2, r_0]])
    R_inv = np.linalg.inv(R)

    BMSE = r_0 - np.transpose(r) @ R_inv @ r

    return x_n_0, BMSE
```

kalman_filter.py

```python
from parameters import *


def kalman_filter(y_n, sigma_v, sigma_w, x_n1_n1, P_n1_n1):
    """
    Kalman filter implementation
    :param y_n: Observation at n
    :param sigma_v: Std deviation for v(n)
    :param sigma_w: Std deviation for w(n)
    :param x_n1_n1: True signal at n-1
    :param P_n1_n1: True error at n-1
    :return: True and estimated signal, true and predicted error
    """

    x_n_n1 = alpha * x_n1_n1
    P_n_n1 = (alpha ** 2) * P_n1_n1 + (sigma_w ** 2)
    K = P_n_n1 * ((sigma_v ** 2 + P_n_n1) ** -1)

    x_n_n = x_n_n1 + K * (y_n - x_n_n1)
    P_n_n = (1 - K) * P_n_n1

    return x_n_n, x_n_n1, P_n_n, P_n_n1
```

wiener_plots.py

```python
import numpy as np
import matplotlib.pyplot as plt
from wiener_interpolator import wiener_interpolator1, wiener_interpolator2
from parameters import *


def generate_obs_seq():
    """
    Generates a realization of the WSS observation
    """

    x = np.zeros(N)

```

```python
    variance_x = sigma_w ** 2 / (1 - alpha ** 2)
    sigma_x = np.sqrt(variance_x)
    x_0 = sigma_x * np.random.randn(1)

    x[0] = x_0
    for n in range(1, N):
        w = sigma_w * np.random.randn(1)
        x[n] = alpha * x[n - 1] + w

    return x


def main():
    """
    For the two Wiener filter interpolators, calculate the theoretical MSE and
    Bayesian MSE averaged over NUM_REALIZATIONS for different values of alpha
    """

    alpha_list = np.arange(0.1, 1, 0.1)

    # Averaged MSEs
    TMSE_1 = np.zeros(len(alpha_list))
    TMSE_2 = np.zeros(len(alpha_list))
    BMSE_1 = np.zeros(len(alpha_list))
    BMSE_2 = np.zeros(len(alpha_list))

    # For each value of alpha find the averaged MSE
    for ind, alpha in enumerate(alpha_list):
        TMSE_1_ALPHA = np.zeros(NUM_REALIZATIONS)
        TMSE_2_ALPHA = np.zeros(NUM_REALIZATIONS)
        BMSE_1_ALPHA = np.zeros(NUM_REALIZATIONS)
        BMSE_2_ALPHA = np.zeros(NUM_REALIZATIONS)

        # Finding the MSE for each realization
        for i in range(NUM_REALIZATIONS):
            x = generate_obs_seq()
            x_n0 = x[n0]
            x = np.delete(x, n0)

            x_n0_pred_1, BMSE_1_ALPHA[i] = wiener_interpolator1(x, n0, alpha)
            x_n0_pred_2, BMSE_2_ALPHA[i] = wiener_interpolator2(x, n0, alpha)

            TMSE_1_ALPHA[i] = (x_n0 - x_n0_pred_1) ** 2
            TMSE_2_ALPHA[i] = (x_n0 - x_n0_pred_2) ** 2

        # Record the average MSE
        BMSE_1[ind] = np.sum(BMSE_1_ALPHA) / NUM_REALIZATIONS
        BMSE_2[ind] = np.sum(BMSE_2_ALPHA) / NUM_REALIZATIONS
        TMSE_1[ind] = np.sum(TMSE_1_ALPHA) / NUM_REALIZATIONS
        TMSE_2[ind] = np.sum(TMSE_2_ALPHA) / NUM_REALIZATIONS

    # Plot and save the required plots
    plt.figure()
    plt.plot(alpha_list, BMSE_1, label="BMSE")
    plt.plot(alpha_list, TMSE_1, label="TMSE")
    plt.xlabel("Alpha")
    plt.ylabel("MSE")
    plt.title("Wiener interpolator 1")
    plt.legend()
    plt.savefig('./results/Wiener_1.png')

```

```
    plt.figure()
    plt.plot(alpha_list, BMSE_2, label="BMSE")
    plt.plot(alpha_list, TMSE_2, label="TMSE")
    plt.xlabel("Alpha")
    plt.ylabel("MSE")
    plt.title("Wiener interpolator 2")
    plt.legend()
    plt.savefig('./results/Wiener_2.png')


if __name__ == '__main__':
    main()
```

kalman_plots.py

```
import numpy as np
import matplotlib.pyplot as plt
from kalman_filter import kalman_filter
from parameters import *


def generate_obs_seq():
    """
    Generates a realization of the WSS observation
    """

    x = np.zeros(N)

    variance_x = sigma_w ** 2 / (1 - alpha ** 2)
    sigma_x = np.sqrt(variance_x)
    x_0 = sigma_x * np.random.randn(1)

    x[0] = x_0
    for n in range(1, N):
        w = sigma_w * np.random.randn(1)
        x[n] = alpha * x[n - 1] + w

    return x


def main():
    """
    For the Kalman filter, find and plot the true and estimated signal
    and the true and predicted error
    """

    y = generate_obs_seq()

    x_true = np.zeros(n0)
    x_pred = np.zeros(n0)
    P_true = np.zeros(n0)
    P_pred = np.zeros(n0)

    x_true[0] = y[0]
    x_pred[0] = y[0]
    P_true[0] = 1
    P_pred[0] = 1

    for i in range(1, n0):
```

```python
        x_true[i], x_pred[i], P_true[i], P_pred[i] = kalman_filter(y[i], sigma_v, sigma_w, x_true[i - 1], P_true[i - 1])

    # Plot and save the required plots
    plt.figure()
    plt.plot(x_true, label="True x[n]")
    plt.plot(x_pred, label="Estimated x[n]")
    plt.xlabel("n")
    plt.ylabel("x[n]")
    plt.title("True and Estimated signal")
    plt.legend()
    plt.savefig('./results/Kalman_1.png')

    plt.figure()
    plt.plot(P_true, label="True Error")
    plt.plot(P_pred, label="Estimated Error")
    plt.xlabel("n")
    plt.ylabel("Error")
    plt.title("True and Predicted error")
    plt.legend()
    plt.savefig('./results/Kalman_2.png')


if __name__ == '__main__':
    main()
```

comparison.py

```python
import numpy as np
from parameters import *


def generate_obs_seq():
    """
    Generates a realization of the WSS observation
    """

    x = np.zeros(N)

    # Initialize x[0]
    variance_x = sigma_w ** 2 / (1 - alpha ** 2)
    sigma_x = np.sqrt(variance_x)
    x_0 = sigma_x * np.random.randn(1)

    # Calculate the remaining observations using equation given
    x[0] = x_0
    for n in range(1, N):
        w = sigma_w * np.random.randn(1)
        x[n] = alpha * x[n - 1] + w

    return x


def kalman_filter(y_n, sigma_v, sigma_w, x_n1_n1, P_n1_n1):
    """
    Kalman filter implementation
    :param y_n: Observation at n
    :param sigma_v: Std deviation for v(n)
    :param sigma_w: Std deviation for w(n)
    :param x_n1_n1: True signal at n-1
```

```
    :param P_n1_n1: True error at n-1
    :return: True and estimated signal, true and predicted error
    """

    x_n_n1 = alpha * x_n1_n1
    P_n_n1 = (alpha ** 2) * P_n1_n1 + (sigma_w ** 2)
    K = P_n_n1 * ((sigma_v ** 2 + P_n_n1) ** -1)

    x_n_n = x_n_n1 + K * (y_n - x_n_n1)
    P_n_n = (1 - K) * P_n_n1

    return x_n_n, x_n_n1, P_n_n, P_n_n1


def wiener_causal_interpolator(x, n_0, alpha):
    """
    Causal Wiener filter implementation
    :param x: Observation vector for past time instants (< n_0)
    :param n_0: The time instant of prediction
    :param alpha: Value of alpha
    :return: The predicted value at n_0 and BMSE
    """

    N = x.shape[0]
    R = np.zeros([N, N])
    r = np.zeros(N)
    a = np.zeros(N)

    for i in range(N):
        for j in range(N):
            if np.abs(i - j) < n_0:
                R[i, j] = alpha ** np.abs(i - j)
            else:
                R[i, j] = alpha ** np.abs(i - j + 1)

    for i in range(N):
        if i < n_0:
            r[i] = alpha ** (n_0 - i)
        else:
            r[i] = alpha ** (i - n_0 + 1)

    R_inv = np.linalg.inv(R)
    a = R_inv @ r
    x_n_0 = np.dot(a, x)

    r_0 = sigma_w ** 2 / (1 - alpha ** 2)
    BMSE = r_0 - np.transpose(r) @ R_inv @ r

    return x_n_0, BMSE


def main():
    """
    Compares and produces the result file
    """

    y = generate_obs_seq()

    x_true = np.zeros(n0)
    x_pred = np.zeros(n0)
    P_true = np.zeros(n0)
```

```
    P_pred = np.zeros(n0)

    x_true[0] = y[0]
    x_pred[0] = y[0]
    P_true[0] = 1
    P_pred[0] = 1

    for i in range(1, n0):
        x_true[i], x_pred[i], P_true[i], P_pred[i] = kalman_filter(y[i], sigma_v, sigma_w, x_true[i - 1], P_true[i - 1])

    kalman_error = P_true[-1]
    _, wiener_error = wiener_causal_interpolator(y[:n0], n0, alpha)

    with open('./results/comparison.txt', "w") as file:
        file.write("The prediction error for Kalman filter is {:0.4f}.\n".format(kalman_error))
        file.write("The prediction error for causal Wiener filter is {:0.4f}.\n".format(wiener_error))


if __name__ == '__main__':
    main()
```

parameters.py

```
N = 100

n0 = 40
sigma_v = 1
sigma_w = 0.6
alpha = 0.8

NUM_REALIZATIONS = 1000
```