Name: Abijith J. Kamath
Student Id: 17788

E1 244: Detection and Estimation

February-May 2021

Solution – Homework 2

Analysis and Algorithms for Faulty Sensor Interpolation

Part A: Derivation and Modelling

Consider the auto-regressive signal of order 1 modelled using the parameter $\alpha \in \mathbb{R}$ as: $x(n) = \alpha x(n-1) + w(n)$ where $w(n)$ is additive white Gaussian noise with variance σ_w^2. The measurements of the signal x is incomplete with one sample missing at index n_0. The $N-1$ length measurement vector can be given as $\mathbf{x} = [x(0)\ x(1)\ \cdots\ x(n_0-1)\ x(n_0+1)\ \cdots\ x(N-1)]^\mathsf{T}$. The interpolation problem is to estimate the sample $x(n_0)$ given complete or partial information $\mathbf{x}$.

Consider the auto-regressive sequence of order 1, modelled as

$$x(n) = \alpha x(n-1) + w(n), \tag{1}$$

where $w(n) \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_w^2)$. The autocorrelation sequence $r_k = \mathbb{E}[x(n)x(n-k)]$ can be computed as:

$$\begin{aligned} r_k &= \mathbb{E}[x(n)x(n-k)] \\ &= \mathbb{E}\left[(\alpha x(n-1) + w(n))x(n-k)\right] \\ &= \alpha\mathbb{E}\left[x(n-1)x(n-k)\right] + \mathbb{E}\left[w(n)x(n-k)\right] \\ &\overset{(a)}{=} \alpha r_{k-1}, \end{aligned} \tag{2}$$

where (a) follows from assuming the noise and the signal are independent. The autocorrelation sequence has a recursive form, with

$$\begin{aligned} r_0 &= \mathbb{E}\left[(\alpha x(n-1) + w(n))(\alpha x(n-1) + w(n))\right], \\ &= \mathbb{E}\left[\alpha^2 x(n-1)x(n-1) + 2\alpha x(n-1)w(n) + w(n)w(n)\right], \\ &= \alpha^2 r_0 + \sigma_w^2, \\ \implies r_0 &= \frac{\sigma_w^2}{1-\alpha^2}. \end{aligned} \tag{3}$$

Wiener Interpolator

The full Wiener interpolator is a linear estimator that minimises the Bayesian mean-squared error (BMSE) using all the sample points in $\mathbf{x}$. The Wiener interpolator is of the form $\hat{x}_{WF}(n_0) = \sum_{i=0, i\neq n_0}^{N-1} a_i x(i) = \mathbf{a}_{WF}^\mathsf{T}\mathbf{x}$, where the weights of the linear interpolator are in the vector $\mathbf{a} = [a_0\ a_1\ \cdots\ a_{n_0-1}\ a_{n_0+1}\ \cdots\ a_{N-1}]^\mathsf{T}$. The BMSE of some estimator $\hat{x}(n_0)$ as a function of the weight vector $\mathbf{a}$:

$$\begin{aligned} bmse(\hat{x}(n_0)) &= \mathbb{E}\left[(x(n_0) - \hat{x}(n_0))^2\right], \\ &= \mathbb{E}\left[x(n_0)x(n_0) - 2x(n_0)\mathbf{a}^\mathsf{T}\mathbf{x} + \mathbf{a}^\mathsf{T}\mathbf{x}\mathbf{x}^\mathsf{T}\mathbf{a}\right]. \end{aligned} \tag{4}$$

The Wiener interpolator is the minimiser of BMSE with respect to the weights $\mathbf{a}$. The Wiener interpolator has weights that satisfy the equation:

$$\frac{\partial}{\partial \mathbf{a}} bmse(\hat{x}_{WF}(n_0)) = \mathbb{E}\left[-2x(n_0)\mathbf{x} + 2\mathbf{x}\mathbf{x}^{\mathsf{T}}\mathbf{a}_{WF}\right] = \mathbf{0}, \tag{5}$$

i.e., the weights satisfy the linear system of equations $\mathbb{E}[\mathbf{x}\mathbf{x}^{\mathsf{T}}]\mathbf{a}_{WF} = \mathbb{E}[x(n_0)\mathbf{x}]$:

$$\underbrace{\begin{bmatrix} r_0 & r_1 & \cdots & r_{n_0-1} & r_{n_0+1} & \cdots & r_{N-1} \\ r_1 & r_0 & \cdots & \cdots & \cdots & \cdots & r_{N-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{n_0-1} & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{n_0+1} & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{N-1} & r_{N-2} & \cdots & \cdots & \cdots & \cdots & r_0 \end{bmatrix}}_{\mathbf{R}_{WF}} \mathbf{a}_{WF} = \underbrace{\begin{bmatrix} r_{n_0} \\ r_{n_0-1} \\ \vdots \\ r_1 \\ r_1 \\ \vdots \\ r_{N-n_0-1} \end{bmatrix}}_{\mathbf{r}_{WF}}. \tag{6}$$

Therefore, the Wiener interpolator has weights $\mathbf{a}_{WF} = \mathbf{R}_{WF}^{-1}\mathbf{r}_{WF}$, and hence $\hat{x}_{WF}(n_0) = \mathbf{a}_{WF}{}^{\mathsf{T}}\mathbf{x}$. Using this in (7),

$$\begin{aligned} bmse(\hat{x}_{WF}(n_0)) &= \mathbb{E}\left[x(n_0)x(n_0) - 2x(n_0)\mathbf{x}^{\mathsf{T}}\mathbf{R}_{WF}^{-1}\mathbf{r}_{WF} + \mathbf{r}_{WF}^{\mathsf{T}}\mathbf{R}_{WF}^{-T}\mathbf{x}\mathbf{x}^{\mathsf{T}}\mathbf{R}_{WF}^{-1}\mathbf{r}_{WF}\right], \\ &= r_0 - 2\mathbf{r}_{WF}^{\mathsf{T}}\mathbf{R}_{WF}^{-1}\mathbf{r}_{WF} + \mathbf{r}_{WF}^{\mathsf{T}}\mathbf{R}_{WF}^{-1}\mathbf{r}_{WF}, \\ &= r_0 - \mathbf{r}_{WF}^{\mathsf{T}}\mathbf{R}_{WF}^{-1}\mathbf{r}_{WF}. \end{aligned} \tag{7}$$

Two-Point-Average Interpolator

The two-point average (TPA) interpolator is a linear estimator that minimises the BMSE using the adjacent samples to the missing sample. The TPA interpolator has the form $\hat{x}_{TPA}(n_0) = a_1 x(n_0 - 1) + a_2 x(n_0 + 1)$, where $\mathbf{a}_{TPA} = [a_1\ a_2]^{\mathsf{T}}$ are the parameters of the estimator. An estimator with this structure is similar to the Wiener interpolator where the coefficients other than that of $x(n_0 - 1)$ and $x(n_0 + 1)$ are set to zero. The corresponding solution is obtained from (10) by taking the 2×2 block:

$$\underbrace{\begin{bmatrix} r_0 & r_2 \\ r_2 & r_0 \end{bmatrix}}_{\mathbf{R}_{TPA}} \mathbf{a}_{TPA} = \underbrace{\begin{bmatrix} r_1 \\ r_1 \end{bmatrix}}_{\mathbf{r}_{TPA}}. \tag{8}$$

Therefore, the two-point average interpolator has weights $\mathbf{a}_{TPA} = \mathbf{R}_{TPA}^{-1}\mathbf{r}_{TPA}$, and hence $\hat{x}_{TPA}(n_0) = \mathbf{a}_{TPA}^{\mathsf{T}}[x(n_0 - 1)\ x(n_0 + 1)]^{\mathsf{T}}$. The solution here can be obtained in closed form with $a_1 = a_2 = \dfrac{\alpha}{1 + \alpha^2}$. Similar to the calculation in (7), the BMSE for TPA interpolator is:

$$bmse(\hat{x}_{TPA}(n_0)) = r_0 - \mathbf{r}_{TPA}^{\mathsf{T}}\mathbf{R}_{TPA}^{-1}\mathbf{r}_{TPA}. \tag{9}$$

Causal Wiener Interpolator

The causal Wiener (CWF) interpolator is a linear estimator that minimises the BMSE using only the previous samples to the missing samples. The CWF interpolator has the form $\hat{x}_{CWF}(n_0) = \sum_{i=0}^{n_0-1} a_i x(i)$, where $\mathbf{a}_{CWF} = [a_0\ a_1\ \cdots\ a_{n_0-1}]^{\mathsf{T}}$ are the parameters of the estimator. This is similar to the structure in the Wiener filter with the coefficients of the positive delays set to zero. The corresponding solution is obtained from (10) by taking the top right $n_0 \times n_0$ block:

$$\underbrace{\begin{bmatrix} r_0 & r_1 & \cdots & r_{n_0-1} \\ r_1 & r_0 & \cdots & r_{n_0-2} \\ \vdots & \vdots & \vdots & \vdots \\ r_{n_0-1} & \cdots & \cdots & r_0 \end{bmatrix}}_{\mathbf{R}_{CWF}} \mathbf{a}_{CWF} = \underbrace{\begin{bmatrix} r_{n_0} \\ r_{n_0-1} \\ \vdots \\ r_1 \end{bmatrix}}_{\mathbf{r}_{WF}}. \tag{10}$$

Therefore, the causal Wiener interpolator has weights $\mathbf{a}_{CWF} = \mathbf{R}_{CWF}^{-1}\mathbf{r}_{CWF}$, and hence $\hat{x}_{CWF}(n_0) = \mathbf{a}_{CWF}^{\mathsf{T}}[x(0) \ \cdots \ x(n_0-1)]^{\mathsf{T}}$. Similar to the calculation in (7), the BMSE for TPA interpolator is:

$$bmse(\hat{x}_{CWF}(n_0)) = r_0 - \mathbf{r}_{CWF}^{\mathsf{T}}\mathbf{R}_{CWF}^{-1}\mathbf{r}_{CWF}. \tag{11}$$

Kalman Filter

Suppose the measurements of the signal be noisy, i.e., $y(n) = x(n) + v(n)$, where $v(n) \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_v^2)$. The observation matrix is identity. The state evolution is given by the auto-regressive process with order 1 as in (1), and the state-transition matrix is the scalar α. With some initialisation for the signal $\hat{x}(0|0)$ and variance of the error $P(0|0)$, the Kalman filter predictions are given as:

$$\begin{aligned} \hat{x}(n|n-1) &= \alpha\hat{x}(n-1|n-1), \\ P(n|n-1) &= \alpha^2 P(n-1|n-1) + \sigma_w^2. \end{aligned} \tag{12}$$

The Kalman gain is defined as $K(n) = P(n|n-1)\left(\sigma_v^2 + P(n|n-1)\right)^{-1}$. The update on the signal and the variance of the error are given by:

$$\begin{aligned} \hat{x}(n|n) &= \hat{x}(n|n-1) + K(n)\left(y(n) + \hat{x}(n|n-1)\right), \\ P(n|n) &= (1 - K(n))P(n|n-1). \end{aligned} \tag{13}$$

Part B: Implementation

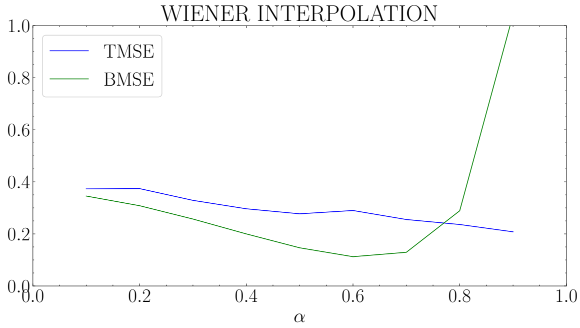
Figure 1 shows Monte-Carlo simulation of interpolation using Wiener interpolation methods. The auto-regressive signal of order 1 with varying α from 0.1 to 0.9 are taken with noise variance fixed at $\sigma_w^2 = 0.36$. The same signal is subject to interpolation at index 40 using three methods based on the Wiener filter idea. The errors upon averaging over 1000 realisations are shown in Figure 1.

Wiener Interpolator

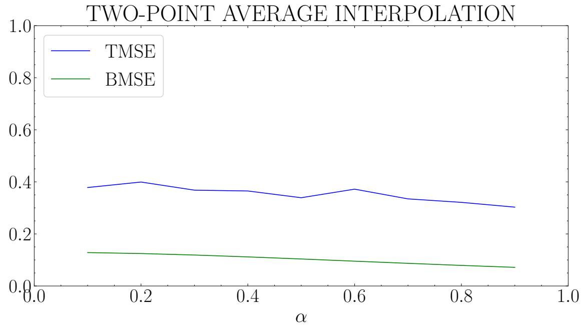
Figure 1(a) shows the Monte-Carlo simulation using complete Wiener filter. The BMSE decreases with α increasing up to 0.6, and then increases. The theoretical mean-squared error (TMSE) shows a decreasing trend with increasing α. It can be seen that the BMSE can be lower than the TMSE, which is indicative of Bayesian methods that introduce bias to reduce the error. As α increases, the signal is approximately a constant buried in noise, where the estimation becomes difficult.

Two-Point-Average Interpolator

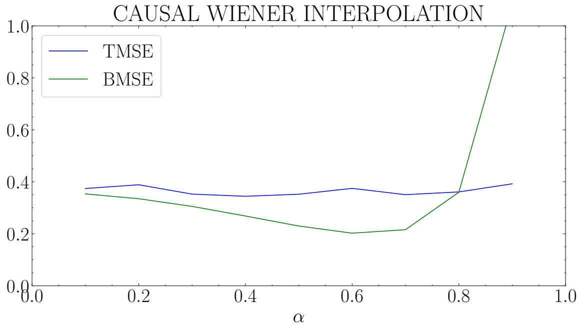
Figure 1(b) shows the Monte-Carlo simulation using two-point average interpolation. The BMSE and TMSE decrease with α and the BMSE is always lower than the TMSE. The two-point average interpolation only considers the relevant sample points and therefore is numerically more stable.



(a)



(b)



(c)

Figure 1: Monte-Carlo analysis of interpolation using Wiener interpolation methods.

Causal Wiener Interpolator

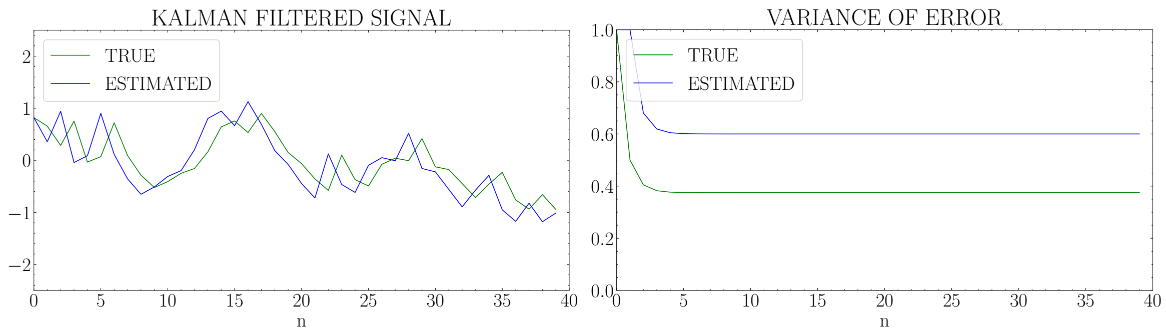
Figure 1(c) shows the Monte-Carlo simulation using causal Wiener interpolation. The BMSE decreases with α increasing up to 0.6, and then increases. The theoretical mean-squared error (TMSE) shows a decreasing trend with increasing α. The errors are higher than the errors using the complete Wiener interpolation as shown in Figure 1(a) as the complete Wiener interpolation uses more samples than the causal Wiener interpolation.

Kalman Filter

Figure 2 shows the results of interpolation using Kalman filter. The auto-regressive signal of order 1 with $\alpha = 0.8$ is taken with noise variance $\sigma_w^2 = 0.36$. The signal is measured directly with Gaussian measurement noise of variance $\sigma_w^2 = 1$. Figure 2(a) shows the true signal and the estimated signal. It can be seen that the estimated signal follows the true signal with an approximately constant delay. This can be seen in 2(b) with the errors saturating at constant values. The errors are high initially as the random initialisations maybe inaccurate, however, with future updates the errors settle.

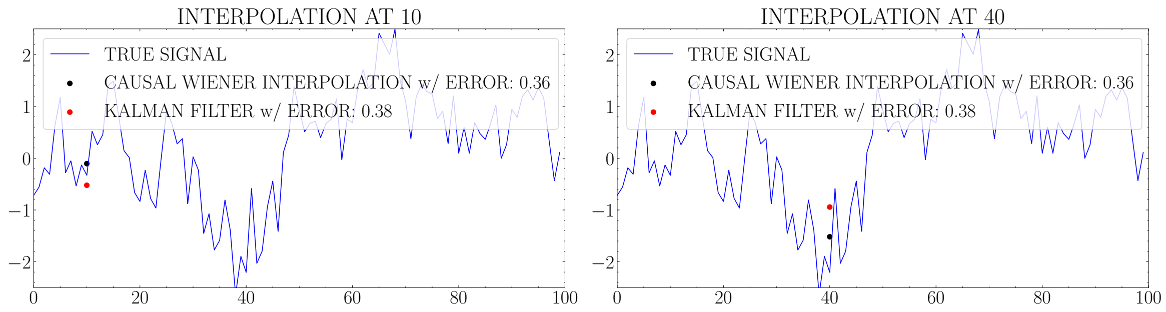
Comparison: Causal Wiener Interpolation vs. Kalman Filter

Figure 3 shows the results of interpolation using causal Wiener interpolation and Kalman filter. The auto-regressive signal of order 1 with $\alpha = 0.8$ is taken with noise variance $\sigma_w^2 = 0.36$. The interpolated point at locations 10 and 40 are shown in Figure 3(a) and 3(b) respectively, along with their errors. The causal Wiener interpolation performs marginally better than the Kalman filter in both cases.



(a) (b)

Figure 2: Interpolation using Kalman filter.



(a) (b)

Figure 3: Comparison of interpolation using causal Wiener interpolation and Kalman filter.

Scripts

The Python3 scripts to generate all figures can be downloaded from the GitHub repository `https://github.com/kamath-abhijith/Wiener_Filter`. Use `requirements.txt` to install all dependencies. Also, see the following code snippets for reference.

Implementation of Interpolation using Wiener filter based methods

The relevant functions are in `utils.py`.

```
'''

FAULTY SENSOR INPERPOLATION USING WIENER FILTER

AUTHOR: ABIJITH J KAMATH
abijithj@iisc.ac.in

'''

# %% LOAD LIBRARIES

import os
import numpy as np

from matplotlib import style
from matplotlib import rcParams
from matplotlib import pyplot as plt

import utils

# %% PLOT SETTINGS

plt.style.use(['science','ieee'])

plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["cm"],
    "mathtext.fontset": "cm",
    "font.size": 24})

# %% INITIALISATION

NUM_REALISATIONS = 1000

ALPHAS = np.arange(0.1,1,0.1)
noise_var = 0.36
idx = 40

TMSE_1 = np.zeros(len(ALPHAS))
TMSE_2 = np.zeros(len(ALPHAS))
TMSE_3 = np.zeros(len(ALPHAS))

BMSE_1 = np.zeros(len(ALPHAS))
BMSE_2 = np.zeros(len(ALPHAS))
BMSE_3 = np.zeros(len(ALPHAS))

# %% MONTE-CARLO ANALYSIS

for alpha_itr, alpha in enumerate(ALPHAS):
    TMSE_ALPHA_1 = np.zeros(NUM_REALISATIONS)
    TMSE_ALPHA_2 = np.zeros(NUM_REALISATIONS)
    TMSE_ALPHA_3 = np.zeros(NUM_REALISATIONS)

    BMSE_ALPHA_1 = np.zeros(NUM_REALISATIONS)
    BMSE_ALPHA_2 = np.zeros(NUM_REALISATIONS)
    BMSE_ALPHA_3 = np.zeros(NUM_REALISATIONS)

    for itr in range(NUM_REALISATIONS):
```

```
        signal = utils.gen_ar1(alpha, noise_var)
        measurements = np.delete(signal, idx)

        interp1, BMSE_ALPHA_1[itr] = utils.wiener_interpolator1(measurements, \
            idx, alpha)
        interp2, BMSE_ALPHA_2[itr] = utils.wiener_interpolator2(measurements, \
            idx, alpha)
        interp3, BMSE_ALPHA_3[itr] = utils.wiener_interpolator3(measurements, \
            idx, alpha)

        TMSE_ALPHA_1[itr] = (signal[idx] - interp1) ** 2
        TMSE_ALPHA_2[itr] = (signal[idx] - interp2) ** 2
        TMSE_ALPHA_3[itr] = (signal[idx] - interp3) ** 2

    TMSE_1[alpha_itr] = np.mean(TMSE_ALPHA_1)
    TMSE_2[alpha_itr] = np.mean(TMSE_ALPHA_2)
    TMSE_3[alpha_itr] = np.mean(TMSE_ALPHA_3)

    BMSE_1[alpha_itr] = np.mean(BMSE_ALPHA_1)
    BMSE_2[alpha_itr] = np.mean(BMSE_ALPHA_2)
    BMSE_3[alpha_itr] = np.mean(BMSE_ALPHA_3)

# %% PLOT SCORES

os.makedirs('./results', exist_ok=True)
path = './results/'

# Wiener interpolator
plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_signal(ALPHAS, TMSE_1, ax=ax, plot_colour='blue', legend_label=r'TMSE',
    xlimits=[0,1], ylimits=[0,1])
utils.plot_signal(ALPHAS, BMSE_1, ax=ax, plot_colour='green', legend_label=r'BMSE',
    xlimits=[0,1], ylimits=[0,1], xaxis_label=r'$\alpha$',
    title_text=r'WIENER INTERPOLATION', save=path+'wiener1')

# Two-point average interpolator
plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_signal(ALPHAS, TMSE_2, ax=ax, plot_colour='blue', legend_label=r'TMSE',
    xlimits=[0,1], ylimits=[0,1])
utils.plot_signal(ALPHAS, BMSE_2, ax=ax, plot_colour='green', legend_label=r'BMSE',
    xlimits=[0,1], ylimits=[0,1], xaxis_label=r'$\alpha$',
    title_text=r'TWO-POINT AVERAGE INTERPOLATION', save=path+'wiener2')

# Causal Wiener interpolator
plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_signal(ALPHAS, TMSE_3, ax=ax, plot_colour='blue', legend_label=r'TMSE',
    xlimits=[0,1], ylimits=[0,1])
utils.plot_signal(ALPHAS, BMSE_3, ax=ax, plot_colour='green', legend_label=r'BMSE',
    xlimits=[0,1], ylimits=[0,1], xaxis_label=r'$\alpha$',
    title_text=r'CAUSAL WIENER INTERPOLATION', save=path+'wiener3')

# %%
```

Implementation of Kalman Filter and Comparisons

The relevant functions are in `utils.py`.

```
'''

FAULTY SENSOR INPERPOLATION USING KALMAN FILTER

AUTHOR: ABIJITH J KAMATH
abijithj@iisc.ac.in

'''

# %% LOAD LIBRARIES

import os
import numpy as np

from matplotlib import style
from matplotlib import rcParams
from matplotlib import pyplot as plt

import utils

# %% PLOT SETTINGS

plt.style.use(['science','ieee'])

plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["cm"],
    "mathtext.fontset": "cm",
    "font.size": 24})

# %% INITIALISATION

alpha = 0.8
process_noise_var = 0.36
meas_noise_var = 1

signal = utils.gen_ar1(alpha, process_noise_var)
measurements = signal + np.random.randn(len(signal))

idx = 40

# %% KALMAN FILTERING

update = np.zeros(idx)
update_var = np.zeros(idx)
prediction = np.zeros(idx)
prediction_var = np.zeros(idx)

update[0] = measurements[0]
prediction[0] = measurements[0]
update_var[0] = 1
prediction_var[0] = 1

for i in range(1, idx):
    update[i], update_var[i], prediction[i], prediction_var[i] = \
        utils.kf(measurements[i], meas_noise_var, process_noise_var, update[i-1],
    update_var[i-1], alpha)

```

```python
# %% COMPARISON

interp3, bmse3 = utils.wiener_interpolator3(signal, idx, alpha)

# %% PLOTS

os.makedirs('./results', exist_ok=True)
path = './results/'

# Signal plots
plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_signal(np.arange(0,idx), prediction, ax=ax, plot_colour='green',
    legend_label=r'TRUE')
utils.plot_signal(np.arange(0,idx), update, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED', xaxis_label=r'n', xlimits=[0,40],
    title_text=r'KALMAN FILTERED SIGNAL', save=path+'kalman1')

# Error plots
plt.figure(figsize=(12,6))
ax = plt.gca()

utils.plot_signal(np.arange(0,idx), update_var, ax=ax, plot_colour='green',
    legend_label=r'TRUE')
utils.plot_signal(np.arange(0,idx), prediction_var, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED', xaxis_label=r'n', xlimits=[0,40], ylimits=[0,1],
    title_text=r'VARIANCE OF ERROR', save=path+'kalman2')

# Comparisons
plt.figure(figsize=(12,6))
ax = plt.gca()

ax.scatter(idx, interp3, label=r'CAUSAL WIENER INTERPOLATION w/ ERROR: %.2f' %(bmse3))
ax.scatter(idx, prediction[idx-1], label=r'KALMAN FILTER w/ ERROR: %.2f' %(update_var[idx-1]))
ax.legend(loc='upper left', frameon=True, framealpha=0.8, facecolor='white')
utils.plot_signal(np.arange(0,100), signal, ax=ax, plot_colour='blue',
    legend_label=r'TRUE SIGNAL', title_text=r'INTERPOLATION AT %d' %(idx), save=path+'comparison1')

# %% COMPARISON

idx = 10
for i in range(1, idx):
    update[i], update_var[i], prediction[i], prediction_var[i] = \
        utils.kf(measurements[i], meas_noise_var, process_noise_var, update[i-1], update_var[i-1], alpha)

interp3, bmse3 = utils.wiener_interpolator3(signal, idx, alpha)

# %% PLOTS

plt.figure(figsize=(12,6))
ax = plt.gca()

ax.scatter(idx, interp3, label=r'CAUSAL WIENER INTERPOLATION w/ ERROR: %.2f' %(bmse3))
ax.scatter(idx, prediction[idx-1], label=r'KALMAN FILTER w/ ERROR: %.2f' %(update_var[idx-1]))
ax.legend(loc='upper left', frameon=True, framealpha=0.8, facecolor='white')
utils.plot_signal(np.arange(0,100), signal, ax=ax, plot_colour='blue',
```

```
    legend_label=r'TRUE SIGNAL', title_text=r'INTERPOLATION AT %d' %(idx), save=path+'
    comparison2')
```

utils.py

This script contains all the relevant functions and helpers.

```
'''

TOOLS FOR FAULTY SENSOR INTERPOLATION
USING WIENER AND KALMAN FILTER

AUTHOR: ABIJITH J KAMATH
abijithj@iisc.ac.in, kamath-abhijith.github.io

'''

# %% LOAD LIBRARIES

import numpy as np

from matplotlib import pyplot as plt

# %% SIGNALS

def gen_ar1(alpha, noise_var, num_points=100):
    '''
    Generate auto-regressive 1 sequence with weight alpha

    :param alpha: weight
    :param noise_var: variance of AWGN
    :optional points: length of the sequence
    :optional init: inital value

    :return: ar1 sequence

    '''

    x = np.zeros(num_points)
    x[0] = np.sqrt(noise_var/(1-alpha**2))*np.random.randn()
    for itr in range(1,num_points):
        x[itr] = alpha*x[itr-1] + np.sqrt(noise_var)*np.random.randn()

    return x

# %% OPERATORS

def acorr_matrix(num_points, idx, alpha):
    '''
    Returns the full autocorrelation matrix for
    full Wiener filter
    '''

    acorr_mtx = np.zeros((num_points-1, num_points-1))

    for i in range(num_points - 1):
        for j in range(num_points - 1):
            if np.abs(i - j) < idx:
                acorr_mtx[i, j] = alpha ** np.abs(i - j)
            else:
```

```
                acorr_mtx[i, j] = alpha ** np.abs(i - j + 1)

    return acorr_mtx

def acorr_sequence(num_points, idx, alpha):
    '''
    Returns the full autocorrelation sequence for
    full Wiener filter
    '''

    acorr_seq = np.zeros(num_points-1)

    for i in range(num_points - 1):
        if i < idx:
            acorr_seq[i] = alpha ** (idx - i)
        else:
            acorr_seq[i] = alpha ** (i - idx + 1)

    return acorr_seq


# %% INTERPOLATORS

def wiener_interpolator1(signal, idx, alpha, noise_var=0.36):
    '''
    Wiener interpolator for one point in AR1 signal

    :param signal: input signal
    :param idx: index to perform interpolation
    :param alpha: weight of AR1 signal
    :param noise_var: variance of awgn on signal

    :return interp: interpolated point
    :return bmse: bayesian mse

    '''

    num_points = len(signal) + 1

    acorr_mtx = acorr_matrix(num_points, idx, alpha)
    acorr_seq = acorr_sequence(num_points, idx, alpha)

    weights = np.linalg.pinv(acorr_mtx) @ acorr_seq
    interp = np.dot(weights, signal)

    acorr_seq0 = noise_var / (1-alpha**2)
    bmse = acorr_seq0 - (acorr_seq.T) @ np.linalg.pinv(acorr_mtx) @ acorr_seq

    return interp, bmse

def wiener_interpolator2(signal, idx, alpha, noise_var=0.36):
    '''
    Two-point average interpolator for one point in AR1 signal

    :param signal: input signal
    :param idx: index to interpolate
    :param alpha: weight of the AR1 signal
    :param noise_var: variance of noise on signal

    :return interp: interpolation
    :return bmse: bayesian mse
```

```

    '''

    interp = alpha / (1 + alpha**2) * (signal[idx-1] + signal[idx+1])

    r0 = noise_var ** 2 / (1 - alpha ** 2)
    acorr_seq = np.array([alpha*r0, alpha*r0])
    acorr_mtx = np.array([[r0, (alpha**2)*r0], [(alpha**2)*r0, r0]])

    bmse = r0 - (acorr_seq.T) @ np.linalg.pinv(acorr_mtx) @ acorr_seq

    return interp, bmse

def wiener_interpolator3(signal, idx, alpha, noise_var=0.36):
    '''
    Causal Wiener interpolator for one point in AR1 signal

    :param signal: input signal
    :param idx: index to interpolate
    :param alpha: weight of the AR1 signal
    :param noise_var: variance of noise on signal

    :return interp: interpolation
    :return bmse: bayesian mse

    '''

    num_points = len(signal) + 1

    acorr_mtx = acorr_matrix(num_points, idx, alpha)
    acorr_mtx = acorr_mtx[:idx, :idx]

    acorr_seq = acorr_sequence(num_points, idx, alpha)
    acorr_seq = acorr_seq[:idx]

    weights = np.linalg.pinv(acorr_mtx) @ acorr_seq
    interp = np.dot(weights, signal[:idx])

    acorr_seq0 = noise_var / (1-alpha**2)
    bmse = acorr_seq0 - (acorr_seq.T) @ np.linalg.pinv(acorr_mtx) @ acorr_seq

    return interp, bmse

def kf(meas, meas_noise_var, process_noise_var, prediction,
    prediction_var, alpha=0.8):
    '''
    Kalman filter update for noisy measurements of AR1 signal

    :param meas: latest measurement
    :param meas_noise_var: variance of measurements
    :param process_noise_var: variance of process
    :param prediction: signal prediction
    :param prediction_var: variance of prediction

    :return update: signal update
    :return update_var: variance of the update
    :return up_pred: updated prediction
    :return up_pred_var: updated variance of prediction

    '''

```

```python
    up_pred = alpha * prediction
    up_pred_var = alpha**2 * prediction_var + process_noise_var

    kalman_gain = up_pred_var / (meas_noise_var + up_pred_var)

    update = up_pred + kalman_gain * (meas - up_pred)
    update_var = (1 - kalman_gain) * up_pred_var

    return update, update_var, up_pred, up_pred_var

# %% PLOTTING

def plot_signal(x, y, ax=None, plot_colour='blue', xaxis_label=None,
    yaxis_label=None, title_text=None, legend_label=None, legend_show=True,
    legend_loc='upper left', line_style='-', line_width=None,
    show=False, xlimits=[0,100], ylimits=[-2.5,2.5], save=None):
    '''
    Plots signal with abscissa in x and ordinates in y

    '''
    if ax is None:
        fig = plt.figure(figsize=(12,6))
        ax = plt.gca()

    plt.plot(x, y, linestyle=line_style, linewidth=line_width, color=plot_colour,
        label=legend_label)
    if legend_label and legend_show:
        plt.legend(loc=legend_loc, frameon=True, framealpha=0.8, facecolor='white')
    plt.xlabel(xaxis_label)
    plt.ylabel(yaxis_label)

    plt.xlim(xlimits)
    plt.ylim(ylimits)
    plt.title(title_text)

    if save:
        plt.savefig(save + '.pdf', format='pdf')

    if show:
        plt.show()

    return
```