EX10.1

wiener_filter_template.py

```
import numpy as np

def wiener_filter(u, y, L):
    """
    Wiener Filter

    Parameters
    ----------
    u : array_like of shape (K,)
        Input Signal u (will be filtered by h)
    y : array_like of shape(K,)
        Reference Signal y (will be compared with y_hat)
    L : int
        Filter length, has to be a positive integer

    Return
    ------
    y_hat : np.ndarray of shape (K,)
        filter output estimation
    h_hat : np.ndarray of shape (filter_length,)
        estimated filter coefficients
    """

    K = len(u) # length of signal
    dummy_array = np.zeros(K)
    dummy_array2 = np.random.randn(L)

    S = np.zeros((K, L)) # memory allocation

    # create S matrix
    for k in range(L, K):

        # 1. crate basis vector p_k and save it into the S matrix (note
the starting index if k)
        S[k:,] = u[k-L:k][::-1]  # CHANGE THIS LINE (right side of equal
sign) !!!

    # 2. estimate filter coefficients h_hat
    h_hat = np.linalg.inv(S.T@S)@S.T.dot(y)
```

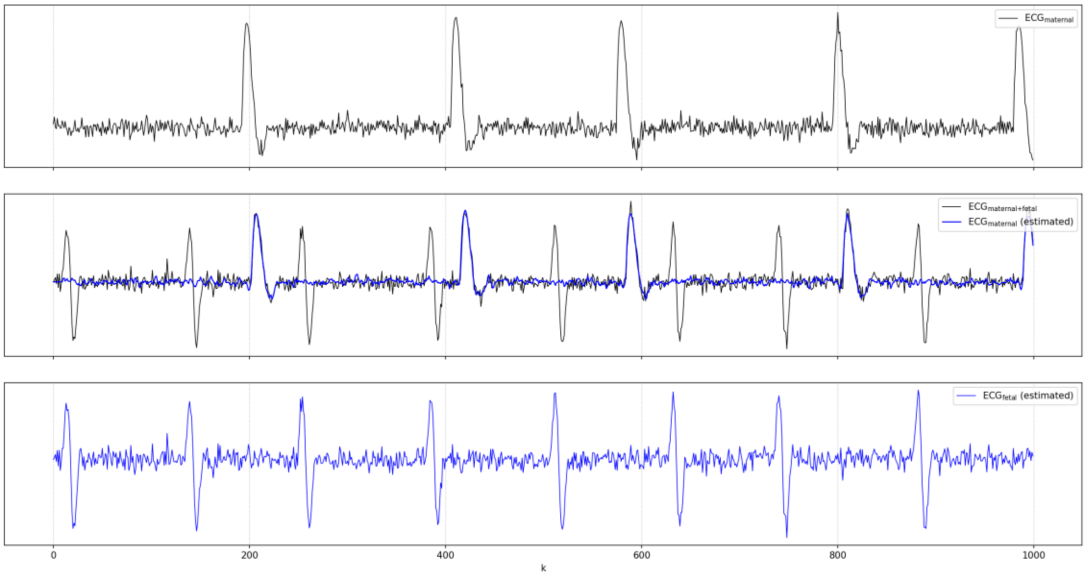
```python
    # 3. filter output estimate y_hat
    y_hat = S @ h_hat # CHANGE THIS LINE

    return y_hat, h_hat
```

ex-10.1-Wiener-Filter-fetal-ECG-reconstruction-template.py

```python
import numpy as np
import matplotlib.pyplot as plt

from wiener_filter_template import wiener_filter

# -----------------------------------------------------------------------
-----------------
# load data
data = np.genfromtxt("ecg_maternal_fetal.csv", delimiter=",",
skip_header=1)

y_maternal = data[:, 0]
y_maternal_fetal = data[:, 1]

dummy_array = np.random.randn(len(y_maternal))*1e-3

# -----------------------------------------------------------------------
-----------------
# Wiener Filter
L = 10 # filter length

# CHANGE NEXT TWO LINES !!!
u = y_maternal
y = y_maternal_fetal

y_hat, h_hat = wiener_filter(u, y, L)

# CHANGE NEXT LINE !!!
y_fetal = y-y_hat

# print coefficients
print("Estimate Coefficients: \n", np.round(h_hat, 2))

# -----------------------------------------------------------------------
-----------------
# plot
```

```
fig, axs = plt.subplots(3, sharex='all')

axs[0].plot(y_maternal, c='k', lw=0.8,
label=r'$\text{ECG}_{\text{maternal}}$')
axs[1].plot(y_maternal_fetal, c='k', lw=0.8,
label=r'$\text{ECG}_{\text{maternal+fetal}}$')
axs[1].plot(y_hat, c='b', lw=1.2,
label=r'$\text{ECG}_{\text{maternal}}$ (estimated)')
axs[2].plot(y_fetal, c='b', lw=0.8,
label=r'$\text{ECG}_{\text{fetal}}$ (estimated)')

for ax in axs:
    ax.grid(lw=0.5, c='grey', ls=':')
    ax.legend(loc=1)
    ax.set_yticks([])
axs[-1].set_xlabel('k')

plt.tight_layout()
plt.show()

# ---------------------------------------------------------------------
------------------
'''
optional
'''
y_maternal_clean=data[:,3]
y_maternal_fetal_clean=data[:,4]

u=y_maternal_clean
y=y_maternal_fetal_clean
_, h_hat = wiener_filter(u, y, L)
print(f"h coefficient: {h_hat}")
```

Result:



EX10.2

lms_filter_template.py

```
import numpy as np

def lms_filter(u, y, L, mu):
    """
    Least Mean Square (LMS) Filter

    Parameters
    ----------
    u : array_like of shape (K,)
        Input Signal u (will be filtered by h)
    y : array_like of shape(K,)
        Reference Signal y (will be compared with y_hat)
    L : int
        LMS filter length, has to be a positive integer
    mu : float
```

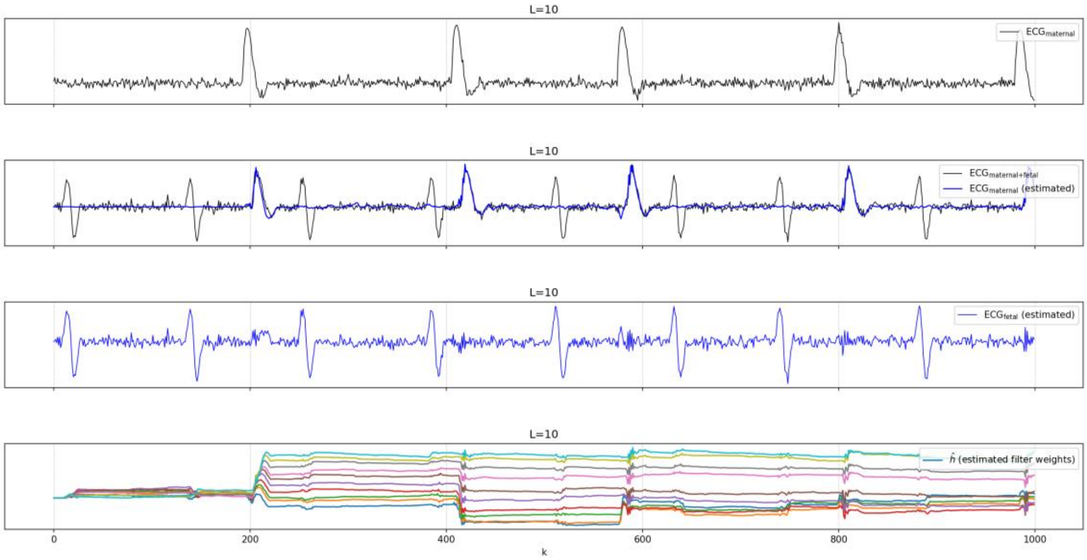
```
        LMS step size, has to be a positive number

    Return
    ------
    y_hat : np.ndarray of shape (K,)
        filter output estimation
    h_hat : np.ndarray of shape (K, filter_length)
        estimated filter coefficients
    e : np.ndarray of shape (K,)
        Error between estimate and reference signal
    """

    K = len(u) # length of signal
    y_hat = np.zeros(K) # allocate output estimate memory
    e = np.zeros(K) # allocate error memory
    h_hat = np.zeros((K+1, L)) # filter coefficients initialization

    dummy_array = np.random.randn(L)

    # iterate through the signal
    for k in range(L, K):

        # Convolution between the input signal and the filter
coefficients (note that the input snippet requires inverse indexing in
the convolution)

        # 1.  x = input signal snippet reversed
        x = u[k-L:k][::-1] # CHANGE THIS LINE (right side of equal
sign) !!!

        # 2. output_estimat[k] = inner product of filter_coefficients
and x
        y_hat[k] = np.dot(h_hat[k], x)

        # Calulate the error
        e[k] = y[k] - y_hat[k]

        # Update the filter coefficients
        h_hat[k+1] = h_hat[k]+mu*e[k]*x # CHANGE THIS LINE (right side
of equal sign) !!!


    return y_hat, h_hat[:K], e
```

ex-10.2-LMS-fetal-ECG-reconstruction-template.py

```python
import numpy as np
import matplotlib.pyplot as plt

from lms_filter_template import lms_filter


# ----------------------------------------------------------------------
------------------
# load data
data = np.genfromtxt("ecg_maternal_fetal.csv", delimiter=",",
skip_header=1)

y_maternal = data[:, 0]
y_maternal_fetal = data[:, 1]

dummy_array = np.zeros_like(y_maternal)

# ----------------------------------------------------------------------
------------------
# LMS Filter
L = 10 # filter length
mu = 0.25 # step size

# CHANGE NEXT TWO LINES !!!
u = y_maternal
y = y_maternal_fetal

y_hat, h_hat, error = lms_filter(u, y, L, mu)

# CHANGE NEXT LINE !!!
y_fetal = y_maternal_fetal-y_hat

# ----------------------------------------------------------------------
------------------
# plot
fig, axs = plt.subplots(4, sharex='all')

axs[0].plot(y_maternal, c='k', lw=0.8,
label=r'$\text{ECG}_{\text{maternal}}$')
axs[1].plot(y_maternal_fetal, c='k', lw=0.8,
label=r'$\text{ECG}_{\text{maternal+fetal}}$')
axs[1].plot(y_hat, c='b', lw=1.2,
label=r'$\text{ECG}_{\text{maternal}}$ (estimated)')
```

```
axs[2].plot(y_fetal, c='b', lw=0.8,
label=r'$\text{ECG}_{\text{fetal}}$ (estimated)')
axs[3].plot(h_hat, label=[r'$\hat{h}$ (estimated filter
weights)']+[None]*(L-1))

for ax in axs:
    ax.grid(lw=0.5, c='grey', ls=':')
    ax.legend(loc=1)
    ax.set_yticks([])
    ax.set_title("L=10")
axs[-1].set_xlabel('k')

plt.tight_layout()
plt.show()

# ---------------------------------------------------------------------
------------------
'''
(c)
if we change the value of L, we can see that the larger the L is , the
estimated ECG curve changes more drastically, and h_hat varies more
the smaller the L is, the estimated ECG curve and h_hat change more
smoothly and stable
'''
```

Result:



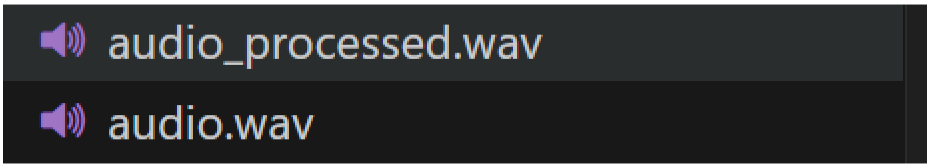
EX10.3

ex-10.3-Echo-Cancelling-template.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile

from lms_filter_template import lms_filter
from wiener_filter_template import wiener_filter

# ----------------------------------------------------------------------
------------------
# load data
fs, data = wavfile.read('audio.wav')

y_clean = data[:, 0].astype(float)
y_mixed = data[:, 1].astype(float)

# noramlize for numerical stability
y_clean /= max(abs(y_clean)) # voice of p1
y_mixed /= max(abs(y_mixed)) # voide of p2 and echo from p1


dummy_array = np.zeros_like(y_clean)

# ----------------------------------------------------------------------
------------------
# LMS and Wiener Filter

#USE_FILER = 'LMS'
USE_FILER = 'WIENER'

L = 1200 # filter length
mu = 1.2e-2 # LMS step size

# CHANGE NEXT TWO LINES !!!
u = y_clean
y = y_mixed

if USE_FILER == 'LMS':
    y_hat, h_hat, error = lms_filter(u, y, L, mu)

if USE_FILER == 'WIENER':
    y_hat, h_hat = wiener_filter(u, y, L)
```

```python
# CHANGE NEXT LINE !!!
y_out = y_mixed-y_hat


# -----------------------------------------------------------------------
------------------
# plot and wav export

# export wav: DO NOT CHANGE!
float32_data = y_out/np.ptp(y_out)
float32_data = float32_data*32767
wavfile.write('audio_processed.wav', fs, float32_data.astype(np.int16))

# plot
fig, axs = plt.subplots(3, sharex='all')

axs[0].plot(y_clean, c='k', lw=0.8, label=r'$u$ (clean)')
axs[1].plot(y_mixed, c='k', lw=0.8, label=r'$y$ (mixed up)')
axs[1].plot(y_hat, c='b', lw=1.2, label=r'$\hat{y}} (estimated)')
axs[2].plot(y_out, c='b', lw=0.8,
label=r'$y_{\text{out}}$ (estimated)')

for ax in axs:
    ax.grid(lw=0.5, c='grey', ls=':')
    ax.legend(loc=1)
    ax.set_yticks([])
axs[-1].set_xlabel('k')

plt.tight_layout()
plt.show()

#-----------------------------------------------------------------------
-------------------------------------------
'''
(b)

the Wiener Filter is offline, there is a time offset, and this filter
runs slower.
The Wiener fifilter is a common method for estimating fifilter
coeffifificients that do not change over time.
Compared with LMS filter, the Wiener filter can give back the optimum
least square solution for the entire signal y.
'''
```

Result:



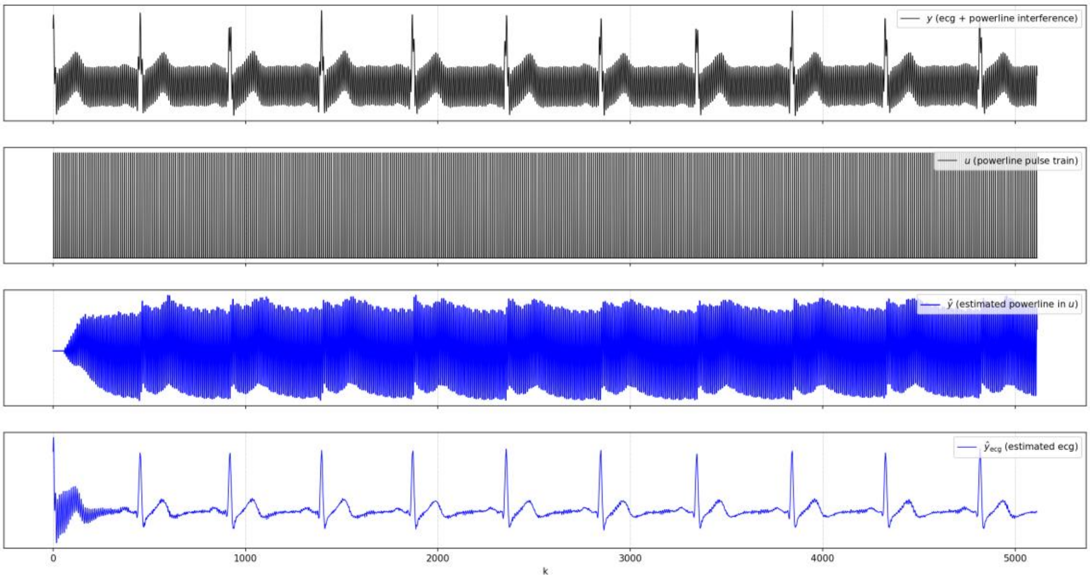
EX10.4

ex-10.4-Remove-Powerline-Interferences-template.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy.signal import find_peaks

from lms_filter_template import lms_filter

# ---------------------------------------------------------------------
------------------
# load data
ecg_data_table = np.genfromtxt('secg-poweline-50Hz-fs-512Hz.csv',
delimiter=',', dtype=float)
fs = 512
t = ecg_data_table[:, 0]  # time
y_ecg_powerline = ecg_data_table[:, 1]  # observations
y_ecg = ecg_data_table[:, 2]   # "true" signal
y_powerline = ecg_data_table[:, 3]   # powerline

dummy_array = np.zeros_like(y_ecg_powerline)

# Generate reference signal
# CHANGE & COMPLETE CODE !
ind, _ = find_peaks(y_powerline, distance=10)
```

```
y_dirac_comb_50hz = np.zeros_like(t)
y_dirac_comb_50hz[ind] = 1
# ----------------------------------------------------------------------
------------------
# LMS Filter

L = 50 # filter length
mu = 20e-3 # LMS step size

# CHANGE NEXT TWO LINES !!!
u = y_dirac_comb_50hz
y = y_ecg_powerline

y_hat, h_hat, error = lms_filter(u, y, L, mu)

# CHANGE NEXT LINE !!!
y_out = y_ecg_powerline - y_hat


# ----------------------------------------------------------------------
------------------
# plot
fig, axs = plt.subplots(4, sharex='all')
axs[0].plot(y_ecg_powerline, c='k', lw=0.8, label=r'$y$ (ecg +
powerline interference)')
axs[1].plot(y_dirac_comb_50hz, c='k', lw=0.8, label=r'$u$ (powerline
pulse train)')
axs[2].plot(y_hat, c='b', lw=1.2, label=r'$\hat{y}$ (estimated
powerline in $u$)')
axs[3].plot(y_out, c='b', lw=0.8,
label=r'$\hat{y}_\text{ecg}$ (estimated ecg)')

for ax in axs:
    ax.grid(lw=0.5, c='grey', ls=':')
    ax.legend(loc=1)
    ax.set_yticks([])
axs[-1].set_xlabel('k')

plt.tight_layout()
plt.show()
```

Result:



y (ecg + powerline interference)
u (powerline pulse train)
ŷ (estimated powerline in u)
ŷ_ecg (estimated ecg)
0
1000
2000
3000
4000
5000
k