Inverse Problems Exercises: 2024s s03 (non-physics)

https://www.umm.uni-heidelberg.de/miism/

Notes

- Please **DO NOT** change the name of the `.ipynb` file.
- Please **DO NOT** import extra packages to solve the tasks.
- Please put the `.ipynb` file directly into the `.zip` archive without any intermediate folder.

Please provide your personal information

- full name (Name):

YOUR ANSWER HERE

D01c: Wiener filter

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

In [ ]:
```
file_gaussian = 'file_gaussian.npz'
with np.load(file_gaussian) as data:
    f_true = data['f_true']
    h_psf = data['h_psf']
    list_gn = data['list_gn']
```

Imaging model

The imaging model can be represented by

$$g = h \otimes f_{\text{true}} = Af_{\text{true}} = \mathcal{F}^{-1}\{\mathcal{F}\{h\}\mathcal{F}\{f_{\text{true}}\}\},$$

$$g' = g + \epsilon.$$

- f_{true} is the input signal
- h is the point spread function (kernel)
- $\otimes$ is the convolution operator
- A is the Toeplitz matrix of h
- $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform operator and inverse Fourier transform operator
- ϵ is the additive Gaussian noise
- g is the filtered signal
- g' is the noisy signal

Fourier transform of the kernel

Implement the Fourier transform of the kernel $\mathcal{F}\{h\}$

- Given the kernel h
- Given the length of the transformed kernel l
- Pad zeros to both sides of the kernel
- Adjust the kernels as long as l
- Shift the origin of the kernels to the first element of the array
- Apply the Fourier transform to the shifted padded kernel (using `numpy.fft.fft()`)
- Implement the function `fft_kernel()` (using `numpy.array`)

Calculate the transformed kernel

- Apply the transform to `h_psf`
- Return the outputs of with the length of 100, 1000, 10000, respectively
- Save the outputs in the variable `list_h_fft` (as `list` of `numpy.array`)

Display the result

- Plot the absolute value of the outputs in `list_h_fft` in the same order of the parameter options in the subplots of `axs`
- Plot the outputs properly in the frequency domain
- Plot the outputs with the marker "+"
- Add proper titles to the subplots of `axs`

In [ ]:
```python
def fft_kernel(kernel, length):
    """Compute the discrete Fourier Transform of the kernel.

    :param kernel: 1d kernel of the system
    :param length: length of the transformed kernel
    :returns: Transformed kernel
    """
# YOUR CODE HERE
raise NotImplementedError()

fig, axs = plt.subplots(3, 1, figsize=(15, 10))
fig.suptitle('Fourier transform')

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```python
# This cell contains hidden tests.
```

In [ ]:
```python
# This cell contains hidden tests.
```

Mean squared error

Implement the mean squared error (MSE)

$$\mathrm{MSE}(\tilde{f}\,) = \frac{1}{n}\sum_{i=1}^{n}(f_{\mathrm{true}i} - \tilde{f}_{\,i})^2.$$

- Given the true signal f_{true}
- Given the estimate $\tilde{f}$
- Implement the function `mean_squared_error()` (using `numpy.array`)

In [ ]:
```python
def mean_squared_error(f_true, f_est):
    """ Compute the mean squared error comparing to the true signal:

    :param f_true: True signal.
    :param f_est: Estimate of the signal.
    :returns: Mean squared error.
    """
# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```python
# This cell contains hidden tests.
```

Inverse filter

Implement the inverse filter

$$\tilde{f}_{\text{inv}} = \mathcal{F}^{-1}\{\frac{1}{\mathcal{F}\{h\} + s^2} \cdot \mathcal{F}\{g'\}\}.$$

- Given the kernel h
- Given the noisy signal g'
- Given the small positive parameter s^2
- Transform the kernel by `fft_kernel()`
- Implement the function `inverse_filter()` (using `numpy.array`)

Apply the inverse filter

- Apply the inverse filter to the noisy signals in `list_gn`
- Return the outputs with s^2 of 0.1
- Save the outputs in the variable `list_f_inv` (as `list` of `numpy.array`)
- Save the mean squared error of each output comparing to `f_true` in the variable `list_mse_inv` (as `list`)

Display the result

- Plot the outputs in `list_f_inv` in the same order of the noisy signals in the subplots of `axs`
- Plot the corresponding noisy signal in each subplot (after the filter output)
- Plot the input signal `f_true` in each subplot (after the noisy signal)
- Show the legend in each subplot
- Show the case information in the titles to the subplots
- Show the mean squared error in `list_mse_inv` in the titles to the subplots

In [ ]:

```
def inverse_filter(kernel, signal, s_sqr):
    """Apply an inverse filter kernel to a signal to deblur it.
    Use a small positive parameter s_sqr to avoid division by zero.

    :param kernel: 1d kernel of the system
    :param signal: 1d signal, which should be filtered
    :param s_sqr: Small positive parameter
    :returns: Filtered signal
    """
# YOUR CODE HERE
raise NotImplementedError()

fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Inverse filter')

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:

```
# This cell contains hidden tests.
```

In [ ]:
```
# This cell contains hidden tests.
```

Question: Inverse filter

What's the influence of s^2 on the result?

YOUR ANSWER HERE

Estimated power spectrum

According to the imaging model, the noise could be estimated by

$$\tilde{\epsilon}_{\text{inv}} = g' - h \otimes \tilde{f}_{\text{inv}},$$

and the corresponding power spectra are

$$P(\tilde{f}_{\text{inv}}) = |\mathcal{F}\{\tilde{f}_{\text{inv}}\}|^2,$$

$$P(\tilde{\epsilon}_{\text{inv}}) = |\mathcal{F}\{\tilde{\epsilon}_{\text{inv}}\}|^2.$$

Display the result

- Plot the estimated power spectrum of the input signal and that of the noise in each subplot
- Show the plot with log scaling on the y-axis
- Show the legend in each subplot
- Show the case information in the titles to the subplots

In [ ]:
```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Estimated power spectrum')

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```
# This cell contains hidden tests.
```

Wiener filter 1

See

- https://en.wikipedia.org/wiki/Wiener_deconvolution

Implement the Wiener filter

$$
\begin{aligned}
\tilde{f}_{\text{Wiener}} &= \mathcal{F}^{-1}\{W \cdot \mathcal{F}\{g'\}\} \\
&= \mathcal{F}^{-1}\{\frac{\mathcal{F}\{h\}^*}{|\mathcal{F}\{h\}|^2 + P(\epsilon)/P(f)} \cdot \mathcal{F}\{g'\}\} \\
&= \mathcal{F}^{-1}\{\frac{1}{\mathcal{F}\{h\}} \cdot \frac{|\mathcal{F}\{h\}|^2}{|\mathcal{F}\{h\}|^2 + P(\epsilon)/P(f)} \cdot \mathcal{F}\{g'\}\}
\end{aligned}
$$

- Given the kernel h
- Given the noisy signal g'
- Given the power spectra $P(f)$ and $P(\epsilon)$
- Transform the kernel by `fft_kernel()`
- Implement the function `wiener_filter()` (using `numpy.array`)

Apply the Wiener filter

- Apply the Wiener filter to the noisy signals in `list_gn`
- Return the outputs with $P(f) = P(\tilde{f}_{\text{inv}})$ and $P(\epsilon) = P(\tilde{\epsilon}_{\text{inv}})$
- Save the outputs in the variable `list_f_wiener_1` (as `list` of `numpy.array`)
- Save the mean squared error of each output comparing to `f_true` in the variable `list_mse_wiener_1` (as `list`)

Display the result

- Plot the outputs in `list_f_wiener_1` in the same order of the noisy signals in the subplots of `axs`
- Plot the corresponding noisy signal in each subplot (after the filter output)
- Plot the input signal `f_true` in each subplot (after the noisy signal)
- Show the legend in each subplot
- Show the case information in the titles to the subplots
- Show the mean squared error in `list_mse_wiener_1` in the titles to the subplots

In [ ]:
```python
def wiener_filter(kernel, signal, P_f, P_e):
    """Apply a wiener filer on the signal to deblur and denoise it.

    :param kernel: 1d kernel of the system
    :param signal: 1d blurred signal with noise
    :param P_f: power spectrum of the deblurred signal
    :param P_e: power spectrum of the noise
    :returns: Deblured and denoised signal
    """
# YOUR CODE HERE
raise NotImplementedError()

fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Wiener filter 1')

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```python
# This cell contains hidden tests.
```

In [ ]:
```python
# This cell contains hidden tests.
```

True power spectrum

According to the imaging model, the true noise is

$$\epsilon_{\text{true}} = g' - h \otimes f_{\text{true}},$$

and the corresponding power spectra are

$$P(f_{\text{true}}) = |\mathcal{F}\{f_{\text{true}}\}|^2,$$

$$P(\epsilon_{\text{true}}) = |\mathcal{F}\{\epsilon_{\text{true}}\}|^2.$$

Display the result

- Plot the true power spectrum of the input signal and that of the noise in each subplot
- Show the plot with log scaling on the y-axis
- Show the legend in each subplot
- Show the case information in the titles to the subplots

In [ ]:
```python
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('True power spectrum')

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```python
# This cell contains hidden tests.
```

Wiener filter 2

Apply the Wiener filter

- Apply the Wiener filter to the noisy signals in `list_gn`
- Return the outputs with $P(f) = P(f_{\text{true}})$ and $P(\epsilon) = P(\epsilon_{\text{true}})$
- Save the outputs in the variable `list_f_wiener_2` (as `list` of `numpy.array`)
- Save the mean squared error of each output comparing to `f_true` in the variable `list_mse_wiener_2` (as `list`)

Display the result

- Plot the outputs in `list_f_wiener_2` in the same order of the noisy signals in the subplots of `axs`
- Plot the corresponding noisy signal in each subplot (after the filter output)
- Plot the input signal `f_true` in each subplot (after the noisy signal)
- Show the legend in each subplot
- Show the case information in the titles to the subplots
- Show the mean squared error in `list_mse_wiener_2` in the titles to the subplots

In [ ]:
```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Wiener filter 2')

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:
```
# This cell contains hidden tests.
```

Question: Wiener filter

Regarding the deblurring task, how sensitive is the reconstruction to estimated power spectrum?

YOUR ANSWER HERE