Inverse Problems Exercises: 2024s s06 (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

D05b: Pseudo-inverse

In [ ]:

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

from scipy.linalg import solve_sylvester
```

In [ ]:

```
file_gaussian = 'file_gaussian.npz'
with np.load(file_gaussian) as data:
    f_true = data['f_true']
    A_psf = data['A_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

Downsampling

- Implement the downsampling matrix

$$D_{\text{ds}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & \dots \\ & & & \dots & & \end{bmatrix}_{n/2 \times n}$$

- Given the size n_{ds}
- Implement the function `get_downsampling_matrix()` (using `numpy.array`)

Prepare the data with downsampling

- Downsample the signal in `list_gn[0]` and save the output in the variable `gn_ds` (as `numpy.array`)
- Calculate the system matrix with downsampling with `A_psf` and save the output in the variable `A_ds` (as `numpy.array`)

In [ ]:
```
def get_downsampling_matrix(n):
    """ Create downsampling matrix.

    :param n: Size of the input signal
    :returns: 2d matrix of size (n/2, n)
    """
# YOUR CODE HERE
raise NotImplementedError()

# YOUR CODE HERE
raise NotImplementedError()
```

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

Pseudo-inverse solutions

In the structure of Tikhonov regularization, a general solution can be written with a pseudo-inverse A^{-I}:

$$\tilde{f} = A^{-I} g.$$

A^{-I} is a matrix satisfying the following expression:

$$\alpha_1 (A^T A) A^{-I} + A^{-I} (\alpha_2 A A^T + \alpha_3 \operatorname{cov}(g)) = (\alpha_1 + \alpha_2) A^T,$$

where α_1, α_2, α_3 are the specific parameters. Especially,

- when $\alpha_1 = 1$, $\alpha_2 = 0$, $\alpha_3 = \lambda$, $\operatorname{cov}(g) = I$, it is the damped least squares

$$A^{-I} = (A^T A + \lambda I)^{-1} A^T,$$

- when $\alpha_1 = 0$, $\alpha_2 = 1$, $\alpha_3 = \lambda$, $\operatorname{cov}(g) = I$, it is the damped minimum length

$$A^{-I} = A^T (A^T A + \lambda I)^{-1}.$$

Implement the pseudo-inverse calculation

- Given the system matrix A
- Given the covariance $\operatorname{cov}(g)$
- Given the parameter set $(\alpha_1, \alpha_2, \alpha_3)$
- Solving the Sylvester equation (using `scipy.linalg.solve_sylvester()`)
- Implement the function `solve_pseudo_inverse()` (using `numpy.array`)

Calculation the pseudo-inverse solutions

- Calculate the solutions for the downsampled data `gn_ds` and `A_ds`
- Use $\operatorname{cov}(g) = I$
- Use 9 different parameter sets $(\alpha_1, \alpha_2, \alpha_3)$ as follows:

$$\begin{array}{lll} (1,0,0), & (0.5,0.5,0), & (0,1,0), \\ (1,0,0.01), & (0.5,0.5,0.01), & (0,1,0.01), \\ (1,0,0.1), & (0.5,0.5,0.1), & (0,1,0.1) \end{array}$$

- Save the pseudo-inverse matrices in the variable `list_A_inv` (as `list` of `numpy.array`)
- Save the pseudo-inverse solutions in the variable `list_f_inv` (as `list` of `numpy.array`)

Display the result

- Plot the outputs in `list_f_inv` in the same order of the parameter options in the subplots of `axs`
- Plot the noisy signal `gn_ds` at the corresponding space coordinates in each subplot
- Plot the input signal `f_true` in each subplot
- Show the legend in each subplot

- Show the case information in the titles to the subplots

In [ ]:
```python
def solve_pseudo_inverse(A, cov_g, alpha):
    """
    :param A: System matrix.
    :param cov_g: Covariance of g.
    :param alpha: Array of 3 alpha values.
    :returns: Pseudo-inverse matrix.
    """
    # YOUR CODE HERE
    raise NotImplementedError()

fig, axs = plt.subplots(3, 3, figsize=(15, 15))
fig.suptitle('Pseudo-inverse solutions')

# YOUR CODE HERE
raise NotImplementedError()
```

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

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

Question: Bias

In which result do you observe the bias of the solution?

- Bias means that the amplitude of the signal is changed in the solution.

YOUR ANSWER HERE

Resolution matrices and covariance matrix

The related matrices, i.e. the data resolution matrix N, the model resolution matrix R and the covariance matrix $\mathrm{cov}(f)$, are defined as follows:

$$N = AA^{-I}$$
$$R = A^{-I}A$$
$$\mathrm{cov}(f) = A^{-I}\,\mathrm{cov}(g)(A^{-I})^T$$

- Given A, A^{-I}, $\mathrm{cov}(g)$
- Implement the function `get_data_resolution_matrix()` (using `numpy.array`)
- Implement the function `get_model_resolution_matrix()` (using `numpy.array`)
- Implement the function `get_cov_f()` (using `numpy.array`)

Calculate the matrices

- Calculate the matrices for the pseudo-inverse matrices in `list_A_inv`
- Save the data resolution matrices N in the variable `list_N` (as `list` of `numpy.array`)
- Save the model resolution matrices R in the variable `list_R` (as `list` of `numpy.array`)
- Save the covariance matrices $\mathrm{cov}(f)$ in the variable `list_cov_f` (as `list` of `numpy.array`)

Display the result

- Plot the matrices in `list_N` as images in the same order of the parameter options in the subplots of `axs_N`
- Plot the matrices in `list_R` as images in the same order of the parameter options in the subplots of `axs_R`
- Plot the matrices in `list_cov_f` as images in the same order of the parameter options in the subplots of `axs_cov_F`
- Show the colorbar of each subplot
- Show the case information in the titles to the subplots

In [ ]:

```
def get_data_resolution_matrix(A, A_I):
    """
    :param A: System matrix.
    :param A_I: Pseudo-inverse matrix.
    :returns: Data resolutin matrix.
    """
# YOUR CODE HERE
raise NotImplementedError()

def get_model_resolution_matrix(A, A_I):
    """
    :param A: System matrix.
    :param A_I: Pseudo-inverse matrix.
    :returns: Model resolutin matrix.
    """
# YOUR CODE HERE
raise NotImplementedError()

def get_cov_f(cov_g, A_I):
    """
    :param cov_g: Covariance of g.
    :param A_I: Pseudo-inverse matrix.
    :returns: Covariance of f.
    """
# YOUR CODE HERE
raise NotImplementedError()

# YOUR CODE HERE
raise NotImplementedError()

fig, axs_N = plt.subplots(3, 3, figsize=(15, 15))
fig.suptitle('data resolution matrix (N)')

# YOUR CODE HERE
raise NotImplementedError()

fig, axs_R = plt.subplots(3, 3, figsize=(15, 15))
fig.suptitle('model resolution matrix (R)')
fig.patch.set_facecolor('yellow')
fig.patch.set_alpha(0.3)

# YOUR CODE HERE
raise NotImplementedError()

fig, axs_cov_f = plt.subplots(3, 3, figsize=(15, 15))
fig.suptitle('covariance cov(f)')

# YOUR CODE HERE
raise NotImplementedError()

# * some points
#   - Check whether the colorbar is shown
#   - Check whether the titles are correct
```

In [ ]:

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

In [ ]:

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

Characteristics

The characteristics, i.e. the spread of a matrix $\mathrm{spread}()$ and the signal size $\mathrm{size}()$, are defined as follows:

$$\mathrm{spread}(B) = \|B - I\|_2^2 = \sum_{ij} (B_{ij} - \delta_{ij})^2$$
$$\mathrm{size}(f) = \sum_i \mathrm{cov}(f)_{ii}$$

- Implement the function `get_spread()` (using `numpy.array`)
- Implement the function `get_size()` (using `numpy.array`)

Calculate the characteristics

- Calculate the characteristics for the matrices in `list_N`, `list_R`, `list_cov_f`
- Save the spread $\mathrm{spread}(N)$ in the variable `list_spread_N` (as `list`)
- Save the spread $\mathrm{spread}(R)$ in the variable `list_spread_R` (as `list`)
- Save the size $\mathrm{size}(f)$ in the variable `list_size_f` (as `list`)
- Save the case index corresponding to the minimal $\mathrm{spread}(N)$ in the variable `idx_spread_N` (as `scalar`)
- Save the case index corresponding to the minimal $\mathrm{spread}(R)$ in the variable `idx_spread_R` (as `scalar`)
- Save the case index corresponding to the minimal $\mathrm{size}(f)$ in the variable `idx_size_f` (as `scalar`)

Display the result

- Plot the value pairs in `list_spread_N` and `list_spread_R` as 2D scatter points in different colors in the left subplot of `axs`
- Plot the value pairs in `list_spread_R` and `list_size_f` as 2D scatter points in different colors in the middle subplot of `axs`
- Plot the value pairs in `list_size_f` and `list_spread_N` as 2D scatter points in different colors in the right subplot of `axs`
- Show the legend in each subplot
- Show the case information and point values in the legend
- Highlight the cases corresponding to the minimal values in the legend

In [ ]:
```
def get_spread(B):
    """
    :param B: Input matrix.
    :returns: Spread of the input matrix.
    """
    # YOUR CODE HERE
    raise NotImplementedError()

def get_size(cov_f):
    """
    :param cov_f: Covariance of f.
    :returns: Size of f.
    """
    # YOUR CODE HERE
    raise NotImplementedError()

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

# YOUR CODE HERE
raise NotImplementedError()
```

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

Question: Selection

Which pseudo inverse do you prefer to minimize $\text{spread}(N) + \text{spread}(R) + \text{size}(f)$?

YOUR ANSWER HERE