Inverse Problems Exercises: 2024s s05 (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

D02b: Tikhonov approach

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']
    A_psf = data['A_psf']
    list_gn = data['list_gn']
```

Imaging model

The imaging model can be represented by

$$g = h \otimes f_{\text{true}} = A f_{\text{true}} = \mathcal{F}^{-1}\{\mathcal{F}\{h\}\mathcal{F}\{f_{\text{true}}\}\},$$
$$g' = g + \epsilon.$$

- 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

Mean squared error

Implement the mean squared error (MSE)

$$\text{MSE}(f) = \frac{1}{n}\sum_{i=1}^{n}(f_i - f_{\text{true}i})^2$$

- Given the input signal f
- Given the true signal f_{true}
- Implement the function `mean_squared_error()` (using `numpy.array`)

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

    :param f: Input signal.
    :param f_true: True signal.
    :returns: Mean squared error.
    """
# YOUR CODE HERE
raise NotImplementedError()
```

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

Difference matrix

Implement the difference matrix D_{diff}

$$D_{\text{diff}} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & -1 \\ -1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 & 0 \\ & & & \dots & & & \\ 0 & 0 & 0 & 0 & \dots & -1 & 1 \end{bmatrix}$$

- Given the size n_{diff}
- Note the -1 on the top right of the matrix
- Implement the function `get_diff_matrix()` (using `numpy.array`)

Calculate the difference matrix

- Return the outputs of with the size of $10, 50, 100$, respectively
- Save the outputs in the variable `list_D_diff` (as `list` of `numpy.array`)

Display the result

- Plot the matrices in `list_D_diff` as grayscale images in the same order of the parameter options in the subplots of `axs`
- Show the colorbar of each subplot
- Add proper titles to the subplots of `axs`

In [ ]:

```
def get_diff_matrix(n):
    """ Compute a matrix to calculate the difference along a vector of the size
    between two neighboring elements.

    :param n: Size of the target vector.
    :returns: Matrix with shape (n, n), which calculates the difference.
    """
# YOUR CODE HERE
raise NotImplementedError()

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

# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:

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

In [ ]:

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

Tikhonov regularization

Implement the objective function with Tikhonov regularization

$$L(f) = \|Af - g'\|_2^2 + \lambda \|D'f\|_2^2$$

- Given the input signal f
- Given the system matrix A
- Given the measurement g'
- Given the regularization matrix D'
- Given the regularization parameter λ
- Implement the function `objective_tikhonov()` (using `numpy.array`)

Implement the closed form solution of the regularized objective function

$$\tilde{f} = (A^T A + \lambda D'^T D')^{-1} A^T g' = A_\lambda^{PI} g'$$

- Given the system matrix A
- Given the measurement g'
- Given the regularization matrix D'
- Given the regularization parameter λ
- Implement the function `solution_tikhonov()` (using `numpy.array`)

In [ ]:

```
def objective_tikhonov(f, A, g, D, lb):
    """ Compute the objective function with Tikhonov regularization.

    :param f: Current estimate of the signal.
    :param A: 2D matrix of the linear problem.
    :param g: Observed signal.
    :param D: 2D matrix in the regularization term.
    :param lb: Regularization parameter.
    :returns: Objective function value.
    """
    # YOUR CODE HERE
    raise NotImplementedError()

def solution_tikhonov(A, g, D, lb):
    """ Compute the estimate of the true signal with Tikhonov regularization.

    Use a regularization term to suppress noise.

    :param A: 2d matrix A of the linear problem.
    :param g: Observed signal.
    :param D: 2D matrix in the regularization term.
    :param lb: Regularization parameter.
    :returns: Estimate of the true signal.
    """
# YOUR CODE HERE
raise NotImplementedError()
```

In [ ]:

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

In [ ]:

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

Best fit solution

The best fit solution is the solution without regularization, i.e. $D' = 0$

- Calculate the closed form solution for the noisy signals in `list_gn`
- Save the outputs in the variable `list_f_est` (as `list` of `numpy.array`)

Display the result

- Plot the outputs in `list_f_est` in the same order of the parameter options in the subplots of `axs`
- Plot the corresponding noisy signal 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
- Show the mean squared error of each output comparing to `f_true` in the titles to the subplots
- Show the objective function value of each output in the titles to the subplots

In [ ]:
```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Best fit solution')

# YOUR CODE HERE
raise NotImplementedError()
```

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

Question: Noise

Why does this solution amplify the noise?

YOUR ANSWER HERE

Minimal length solution

The minimal length solution is the solution with $D' = I$

- Calculate the closed form solution for the noisy signals in `list_gn`
- Return the outputs with λ of 0.1, 0.01, 0.001, respectively
- Save the outputs in the variable `list_f_est` (as `list` of `numpy.array`)

Display the result

- Plot the outputs in `list_f_est` in the same order of the parameter options in the subplots of `axs`
- Show the cases of the same noisy signal in the same subplot column
- Show the cases with the same λ in the same subplot row
- Plot the corresponding noisy signal 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
- Show the mean squared error of each output comparing to `f_true` in the titles to the subplots
- Show the objective function value of each output in the titles to the subplots

In [ ]:
```
fig, axs = plt.subplots(3, 3, figsize=(15, 15))
fig.suptitle('Minimal length solution')

# YOUR CODE HERE
raise NotImplementedError()
```

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

Gradient magnitude solution

The gradient magnitude solution is the solution with $D' = D_{\text{diff}}$

- Calculate the closed form solution for the noisy signals in `list_gn`
- Return the outputs with λ of 0.1, 0.01, 0.001, respectively
- Save the outputs in the variable `list_f_est` (as `list` of `numpy.array`)

Display the result

- Plot the outputs in `list_f_est` in the same order of the parameter options in the subplots of `axs`
- Show the cases of the same noisy signal in the same subplot column
- Show the cases with the same λ in the same subplot row
- Plot the corresponding noisy signal 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
- Show the mean squared error of each output comparing to `f_true` in the titles to the subplots
- Show the objective function value of each output in the titles to the subplots

In [ ]:
```
fig, axs = plt.subplots(3, 3, figsize=(15, 15))
fig.suptitle('Gradient magnitude solution')

# YOUR CODE HERE
raise NotImplementedError()
```

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

Question: Regularization

1. How can the different reconstruction techniques be characterized when comparing the results, especially for corners or edges?
2. What is the consequence when changing λ, especially for very small and very large values, as observed on the solution?

YOUR ANSWER HERE