COMP0114 Inverse Problems in Imaging. Coursework 1 Hints

Introduction

This document is designed as support for Coursework 1, containing programming hints for Matlab and Python.

HINTS

1. Solving Underdetermined Problems

a.) **Matlab:** You can use the concept of an *anonymous function* or *function handle* and define an anonymous function `Phi(x,p)` that computes Φ.
Python: You can use the concept of **lambda** functions or a function definition **def** to compute Φ.

b.) **Matlab:** You can use the function `fmincon` to solve the constrained optimisation problem here, study its documentation to figure out the correct orders and data types of input and output arguments.
Python: The function `scipy.optimize.minimize` can be used to solve the above constrained optimization problem.

c.) **Matlab:** To plot the function and points, you need to study the MATLAB functions of graphics. Familiarise yourself with Matlab's 2D graphics (MATLAB > Graphics > 2-D and 3-D Plots > Line Plots) to plot your results as shown in Figure 1: one may use *hold on* to plot lines and points on the same figure and modify the configurations of plot functions for customised shapes and colours.

To export the line plot, consult the documentation (MATLAB > Graphics > Printing and Saving > Save Figure for Document or Presentation) and use the functions `print` or `saveas` to save the graphic in a high quality image format (for instance as `.pdf` or `.png`; avoid using compressed formats such as `.jpeg`). Clip the exported image if necessary (see "Tools").

Python: Familiarize yourself with the plotting library `matplotlib` to produce similar high quality figures.

d.) **Matlab:** You can directly compute the pseudo inverse by `pinv`. There is another possibility to compute a robust solution to a system `Bx=b` with the *backslash operator* $B\backslash b$. Do these methods yield the same solution?
Python: You can use the function `numpy.linalg.pinv`. Similarly to Matlab, the library `numpy` also provided a class `linalg` which performs the *backslash operation*.

2. Singular Value Decomposition

a.) **Matlab:** You can also use the function `linspace`.
Python: The library `numpy` contains a function `linspace` for this purpose.

b.) **Matlab:** To visualise a vector g, simply plot it by: `plot(g)`.
Python: you can simply use `matplotlib.pyplot.plot(g)` to visualizse a vector g.

d.) **Matlab:** One way of visualising A is to use MATLAB's `imagesc` function to plot A and and use `saveas` to export the plot as a `.png`. Often, it is advantageous to export images directly into a raster graphic format. This can be done using Matlab's `imwrite` function, but requires a rescaling of A and the explicit definition of a *colormap*:

```
>> Aimg = ceil(A/max(A(:))*256);
>> colorMap = parula(256);
>> imwrite(Aimg,colorMap,'Aimage2.png')
```

Consult the documentation to comprehend the above commands.

Python: We require the library `opencv` (For installation use for instance `pip install opencv-python`) to obtain similar results as in Matlab:

```
import cv2
import numpy as np
Atmp = np.array(np.ceil(A/np.max(A)*256), dtype = np.uint8)
Aimg = cv2.applyColorMap(Atmp, cv2.COLORMAP_JET)
cv2.imwrite("Aimage3.png",Aimg)
```

e.) **Matlab:** Use the function `svd` to compute the SVD of A.

You can check if two matrices A and B are equal (or close) by computing the norm of their difference: `norm(A-B)`.

Python: Use `numpy.linalg.svd` to compute the SVD of A.

You can use `numpy.linalg.norm` to compute the norm.

f.) **Matlab:** One can construct the pseudoinverse $W^\dagger$ of W as a *sparse matrix* by using the `spdiags` function (familiarise yourself with the concept of sparse matrices), in order to save space for large dimensional cases. One can use the function `spy` to check whether $W^\dagger$ has the desired form and verify that $A^\dagger = VW^\dagger U^T$ by using Matlab's `pinv` function to compute $A^\dagger$.

To plot diag(W) on a logarithmic scale on the y-axis, you can use the function `semilogy` instead of plot, i.e. `semilogy(diag(W))`.

Python: Construct the pseudoinverse $W^\dagger$ of W as a *sparse matrix* by using `scipy.sparse.spdiags` function. Once can use function `matplotlib.pyplot.spy` to check the form of $W^\dagger$ and verify that $A^\dagger = VW^\dagger U^T$ by function `numpy.linalg.pinv`.

You can use `matplotlib.pyplot.semilogy(numpy.diag((W))` to make a plot with logarithmic scaling on the y-axis.

3. Convolutions and Fourier transform

e.) As the output of the Fourier transform is complex, you have to look at real, imaginary or absolute values separately for plotting purposes. It can be also useful to look at the output with and without using shifting as explained in the following:
Matlab: For computational reasons FFT performs a shifting, to keep the correct order of your signal `x` perform the Fourier transform by `Fx=fftshift(fft(fftshift(x)))` and similarly for the inverse Fourier transform `fftshift(ifft(fftshift(Fx)))`. For pointwise multiplication in Fourier space use the operation: `.*`
Python: We require `numpy.fft` module to perform FFT as in Matlab:

```
import numpy as np
Fx = np.fft.fftshift(np.fft.fft(np.fft.fftshift(x)))
x  = np.real(np.fft.fftshift(np.fft.ifft(np.fft.fftshift(Fx))))
```