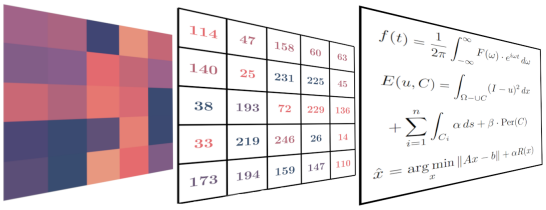
UNIVERSITY OF CAMBRIDGE



MPhil in Data Intensive Science

Submission: 11:59pm on Friday the 21st of June

Coursework Assignment - Image Analysis

Dr. A. Biguri, Dr. P. Cañizares, Dr. S. Mukherjee, Dr. M. Roberts

The coursework consists of 3 parts: (1) Fundamentals of Image Analysis, (2) Inverse problems and multi-resolution analysis, and (3) Variational methods for inverse problems in imaging. The final report should explain how you have arrived to the conclusions of your answers and the choices for each step. The report should include your analysis, presenting the illustrations generated, a discussion and what you have learned doing this coursework for each part. The marks for each part are indicated in square brackets next to the heading. The coursework has a total of [100] marks available.

The coursework will be submitted via a GitLab repository which will be created for you. You should place all your code and your report in this repository. The report should be in PDF format in a folder called "report". You will be provided access to the repository until the deadline above. After this you will lose access which will constitute submission of your work.

You are expected to submit code and associated material that demonstrates good software development practices as covered in the Research Computing module.

Your report should not exceed 3000 words (including tables, figure captions and appendices but excluding references); please indicate its word count on its front cover.

You are reminded to comply with the requirements given in the Course Handbook regarding the use of, and declaration of, autogeneration tools for both the code and report.

Module 1: Fundamentals of Image Analysis

[33]

Exercise 1.1 Given the following 3 images, segment the corresponding part (see image caption) using any image processing technique that is not using supervised machine learning (i.e. no training, no CNNs, etc). Using `skimage` do the following:
Question 1.1:

(a) Produce a successful segmentation

(b) Use three different segmentation algorithms

(c) Argue why the chosen pre-processing and segmentation is appropriate for each particular case

(d) Write your own code for one segmentation algorithm

There is no score/ground truth that your results will be compared against, qualitative evaluation only. You are only allow to use the information given by the task text e.g. for b, the location of the flowers (middle) has not been given to you in the text, so you can't use that information to segment.
Images (find original files in Moodle):

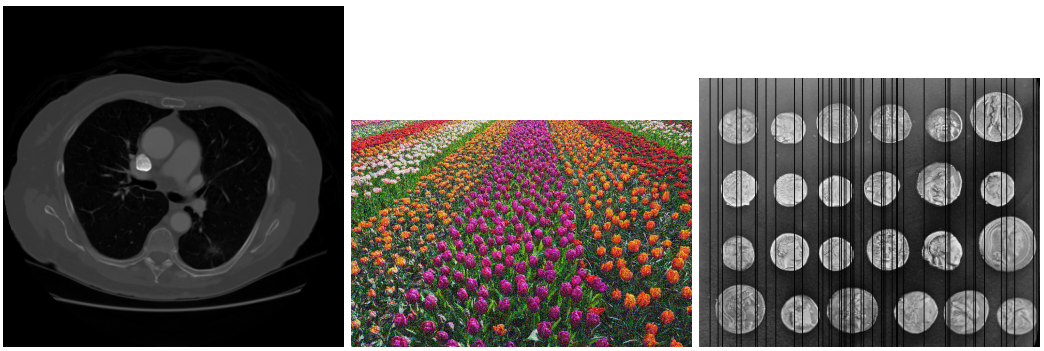


Figure 1: (a) CT image of the chest (from the LIDC-IDRI dataset). Segment the lung area, including any nodule/tissue inside of it. (b) Noisy image of tulips, select the purple ones (the petals only, not the stem). (c) Damaged picture of coins. Provide segmentation of the first coin in the first row, second coin in second row, third coin in third row and fourth coin in fourth row.

Module 2: Inverse Problems and multiresolution analysis

[33]

Exercise 2.1: During the lectures, we saw that ill-posed inverse problems can be solved using optimisation (minimisation) algorithms of the form:

$$\min_u \|Au - f\|_{\ell_n}$$

with different types of distances (norms), typically ℓ_1 ($n = 1$) and ℓ_2 ($n = 2$). Both norms have quite different properties when it comes to fit data. One important feature of the ℓ_2 norm is that the associated minimisation problem is warranted to produce a unique solution, while the ℓ_1 does not. In particular, we have seen that least squares is a suitable algorithm to find a solution for ℓ_2 minimisation problems. However, the most important feature of ℓ_1 is the sparsity. Hence, instead of finding a smoothed fit, like ℓ_2, an ℓ_1 minimisation problem looks for sparse solutions.
This exercise will explore the difference between ℓ_1 and ℓ_2 minimisation in a simple 1D case, fitting a straight line.

$$\min_{a,b} \|a\boldsymbol{x} + b - y\|_{\ell_n}$$

for $n =, 1, 2$. To that end, get the data for the straight line with random noise from the **y_line.txt** file, and the data with oulier from the **y_outlier_line.txt** file. Once you have the data:
Question 2.1:

(a) Use ℓ_1 minimisation and ℓ_2 minimisation to find the best coefficients.

(b) Which method is more sensitive to outliers? Which method finds the best solution in the noisy case (not outliers), and why?

Exercise 2.2: One of the key aspects of compressed sensing theory is random undersampling. To understand why, in this basic exercise, we will compare uniform (equidistant) undersampling and random undersampling to reconstruct a noisy signal by solving:

$$\arg\min \left(\frac{1}{2} \|\mathcal{F}_\Omega \hat{x} - y\|_2^2 + \lambda |\hat{x}|_1 \right)$$

where $\hat{x}$ is our predicted signal, $\mathcal{F}_\Omega \hat{x}$ is our undersampled Fourier transform of the prediction, and y is the measured Fourier transform. Notice that $\hat{u}$ and y are related through a FT and there is no closed-form solution. We can solve this problem iteratively using (i) *soft-thresholding* and (ii) constraining data consistency to fill missed k-space points:

$$\hat{X}_{i+j}[j] = \begin{cases} \hat{X}_i[j] & \text{if} y[j] = 0 \\ y[j] & \text{otherwise} \end{cases}$$

(TURN OVER

You can find a sample algorithm for iterative soft-thresholding in the helper function.
Question 2.2: To do so:

(a) Generate a length-100 vector, with 10 non-zero coefficients and permute them randomly (Hint use: np.random.permutation function).

(b) Add random Gaussian noise with standard deviation $\sigma = 0.05$.

(c) Generate uniformly (equidistant) sampled data: undersample the data by taking 32 random samples. Compute the zero-filled inverse Fourier transform and multiply by four. This will be your 4-fold aliased signal.

(d) Generate random sampled data: undersample the data by taking 32 random samples. Compute the zero-filled inverse Fourier transform and multiply by four again. You have generated undersampled and incoherent data with 4-fold aliasing. This will be your noisy signal.

(e) Obtain the original signal by applying iterative soft-thresholding ($n_{iter} = 100$) to the (1) randomly sample data and (2) the equally spaced data.

(f) Plot the reconstructed signal using random and equidistant undersampling. What is the difference, and why?

Exercise 2.3 Compressed reconstruction. In general, images are not sparse. However, natural and medical images can have a sparse representation in a suitable transform domain, such as the wavelet domain. Here, we will use the PyWavelets package to perform wavelet transforms. You can install the package with the command >> pip install PyWavelet.
Question 2.3:

(a) Open the "river_side.jpeg" image provided in Moodle, compute the Daubechies wavelet transform, display it, and reconstruct it again. Compare it to the original image qualitatively and by computing the difference image.

(b) Wavelet coefficients represent both space and spatial frequency information. Each band of wavelet coefficients represents a scale (frequency band) of the image. The location of the wavelet coefficient within the band represents its location in space. Threshold the wavelet coefficients retaining only the largest 15%.

(c) Visualize the results when you retain the largest 20%, 10%, 5% and 2.5% of the coefficients. Reconstruct the corresponding image and display it.

Module 3: Solving inverse problems

[34]

Exercise 3.1: Recall that for gradient descent on an L-smooth function $f(x)$, the following error bound holds:

$$f(x_K) - f(x^*) \leq \frac{L \|x_0 - x^*\|_2^2}{2K}, \tag{1}$$

Question 3.1: Consider the task of minimizing the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined as $f(x) = \frac{[x]_1^2}{2} + [x]_2^2$, where $x = \begin{bmatrix} [x]_1 \\ [x]_2 \end{bmatrix}$, using gradient descent with initialization $x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

1. Using (1), calculate how many iterations K will be sufficient to achieve an accuracy of $\epsilon = 0.01$.

2. Write a script to calculate how many iterations are actually needed to achieve a solution with accuracy $\epsilon = 0.01$. Is it different from what your theoretical calculations produced in Part 1? If so, can you explain why?

Hint: Can you argue that the given function is L-smooth with $L = 2$?

Exercise 3.2: This exercise is meant to give you a hands-on experience to compare the performance of a classical model-driven regularization with a data-driven reconstruction approach (using a single image, namely the Shepp-Logan phantom). We will give you a standalone `python` script (with the necessary boilerplate code) that you can execute on `Google Colab`. You will need to fill in some important parts in the script (indicated as "YOUR CODE HERE") to complete your coursework.

1. **Step 1**: In this step, we will implement the ADMM algorithm (c.f. Part 3) using the `ODL` library for solving the total-variation (TV) regularized reconstruction problem for sparse-view computed tomography. Recall that the TV-regularized reconstruction problem reads

$$\min_x \ \|y - Ax\|_2^2 + \lambda \|\nabla x\|_1 , \tag{2}$$

where A is the CT forward operator, y is the noisy sinogram, and λ is a regularization penalty parameter. This part has been coded completely.

2. **Step 2**: In this part, you will need to compare the performance of TV-regularized reconstruction with that of a data-driven approach (namely, learned gradient descent, which parameterizes the reconstruction operator by unrolling the gradient descent algorithm). In particular, the reconstruction is given by x_M, where x_{k+1} is computed from x_k using

$$x_{k+1} = x_k + \tau_k \, h_{\theta_k}\left(x_k, \quad \underbrace{A^\top (A x_k - y)}_{\text{gradient of the fidelity term}} \right), \ \text{for } k = 0, 1, \cdots, M-1. \tag{3}$$

(TURN OVER

Here, h_{θ_k}s are shallow (three-layer) convolutional neural networks (CNNs) with learnable parameters θ_k and τ_ks are learnable step-sizes in each iteration. We choose M, the number of unfolded iterations, to be $M = 5$. We will initialize the unrolled iterations with the filtered back-projection (FBP) reconstruction.

Use the example script at

`https://colab.research.google.com/drive/1JNVhV1wmx1F-tw50t42lPyLM1MR4qtbY?usp=sharing` to complete this part of the coursework. The parts to be filled in by you are marked as "YOUR CODE HERE" in three different places in Cell-5. You can copy this and create your own notebook or Python script (recommended) to complete this coursework.

Question 3.2:

(a) Complete the code for performing the forward pass in the `prox_net` module that implements the architecture of the CNN h_{θ_k}.

(b) Complete the code for forward pass in the `LGD_net` module (which implements the iteratively unrolled network described by (3)).

(c) Complete the steps inside the training loop for updating network parameters.