Python-Mini-Project
Sampling Signals with Finite Rate of Innovation with an Application to Image Super-Resolution

Pier Luigi Dragotti, Jesse Berent, Loïc Baboulaz, Marek Hilton
Communications and Signal Processing Group
Department of Electrical and Electronic Engineering
Imperial College London

The coursework consists of a Python Miniproject. It counts 25% of the final mark. Students should produce a written final report along with code which is to be submitted separately. We expect the code to be neatly written and well documented.

1 Submission details and Guidelines

The deliverable for this course consists of two parts:

- The final written report must be submitted through blackboard by the deadline indicated on the module page
- The code submission, consisting of a single python script called submission.py, must be submitted to a separate submission page on blackboard. A blank submission.py already exists in the project folder.
- The project files including skeleton code and data can be found on blackboard.

1.1 Report submission

The report should give full answers to each question including plots and derivations, however, details of code implementation should be confined to the code submission and not the report.

1.2 Code submission

Code submissions for this coursework must be written in Python. Python has been chosen over MATLAB since it has more widespread use outside of academic circles. Furthermore, it is free to use without a license.

Code should be clear and well documented. In cases where code produces the wrong result and is also convoluted or hard to understand, no marks will be given.

1.2.1 Allowed Python modules

Python scripts must utilise only the following modules:

1. **PyWavelets** for generating wavelets
2. **numpy** for general matrix and vector manipulation
3. **matplotlib** for plotting

The following libraries will also be used by the image registration skeleton code:

- **Pillow**
- **scipy**
- **scikit-image**

1.2.2 Submission format

Each question that requires a code submission will specify a prototype of a function (i.e. the name, inputs, and outputs) that will be required to answer the question. For each input and output the type will also be given. For example:

```
name    dummy_exercise
arg1    1D Array of signal samples : numpy.ndarray
arg2    Sample rate : float
output  1D array of innovation times : numpy.ndarray
```

Functions that do not adhere to these prototypes will lose a portion of the marks allocated for code correctness.

In total, the code submission for this coursework should consist of a single python script called 'submission.py' containing function definitions for each of the requested functions.

1.2.3 Project files contents

The supplied project folder contains a number of files listed here for reference:

1. **submission.py** - The empty python script to be filled with requested functions
2. **data.py** - Python module that contains some data required for the exercises
3. **image_fusion.py** - Python module containing functions and variables needed for the final exercise on super resolution
4. **Pipfile** and **Pipfile.lock** - Pipenv files. These can be used for setting a python environment with all the appropriate modules installed. See Pipenv for more details.
5. **project_files_&_data** - Folder containing data for the project. It is not necessary to access this folder manually.

A Python "pipenv" is provided in the project repo should students wish to use them. Use of any of this helper tool is not required but provided for the convenience of students.

1.2.4 Supplementary code

Any code not explicitly asked for in the code submission may be placed in an appendix of the main report if it is a useful aid to understanding your report. You should avoid placing all your code in the appendices.

2 Preliminaries

The classic sampling theorem states that any bandlimited function $x(t)$ such that $X(\omega) = 0 \; \forall \; |\omega| > \omega_{max}$ can be exactly recovered from its samples given that the rate $2\pi/T$ is greater or equal to twice the highest frequency component. The continuous time signal is recovered with $x(t) = \sum_{n \in \mathbb{Z}} y[n] \text{sinc}(t/T - n)$ where $\text{sinc}(t) = \sin(\pi t)/\pi t$ and $y[n] = x(nT)$. A fun extension of this theorem has recently been developed, where it is made possible to sample signals that are not bandlimited but completely determined by a finite number of parameters. Such signals are called Finite Rate of Innovation (FRI) signals. In particular, it is made possible to sample and exactly recover streams of Diracs, differentiated Diracs and piecewise polynomials using a smoothing kernel $\varphi(t)$ that satisfies the Strang-Fix conditions. In this case, the samples are given by $y[n] = \langle x(t), \varphi(t - n) \rangle$. The main aim of this project is to make the student familiar with these recently developed sampling theorems. Notice that this project refers mainly to Chapter 5 of the lecture notes.

3 Strang-Fix conditions

One of the main conditions on the sampling kernel $\varphi(t)$ is that it should be able to reproduce polynomials. Therefore, there exist coefficients $c_{m,n}$ such that

$$\sum_{n\in\mathbb{Z}} c_{m,n}\varphi(t-n) = t^m \quad m = 0, 1, \ldots, N. \tag{1}$$

In our context, we assume that $\{\varphi(t-n)\}_{n\in\mathbb{Z}}$ spans a Riesz space. Therefore there exists a dual-basis $\{\tilde{\varphi}(t-n)\}_{n\in\mathbb{Z}}$ that satisfies $\langle\varphi(t), \tilde{\varphi}(t-n)\rangle = \delta_0$. The coefficients $c_{m,n}$ can thus be found by applying the inner product with the dual-basis $\tilde{\varphi}(t-n)$ on both sides of equation (1). It results that $c_{m,n} = \langle t^m, \tilde{\varphi}(t-n)\rangle$. Notice that a scaling function that generates wavelets with $N+1$ vanishing moments is able to reproduce polynomials of degree N. It can therefore be used in our case as a sampling kernel.

Exercise 1. **The Daubechies scaling function**

a. *Which Daubechies scaling function should be used in order to reproduce polynomials of maximum degree* 3*?*

b. *Write a function that calculates coefficients to reproduce polynomials of degree* 0 *to* 3*:*

```
name    get_cmn_db
arg1    shift number : Int
arg2    polynomial order : Int
output  coefficient : float
```

c. *Compute the coefficients to reproduce polynomials of degree* 0 *to* 3 *for* $n = 0, \ldots, 32-L$ *where* L *is the support of the kernel. Plot the results with the shifted kernels and the reproduced polynomials as in Figure 5.2 of the lecture notes.*

Guideline:

- *Choose the right Daubechies scaling function. For example, assume it is 'dB2'.*
- *To generate the* $\varphi(t)$ *with a resolution* $\frac{1}{2^J} = \frac{1}{64}$*, do as follows:*

```
> import pywt
> db2 = pywt.Wavelet('db2')
> phi, psi, x = db2.wavefun(level=6)
> phi = np.pad(phi,(0,1984))
```

- *The array* `phi` *is in fact a 6-iteration approximation of* $\varphi(t)$ *padded with zeros to a length of 2048. The shifted version* $\varphi(t-nT)$ *where* $T = 1$ *is obtained by shifting* `phi` *by* $64n$ *points. To do this you can either reassign with index slices or by using* np.roll*, though be aware that this rotates the array like* circshift *in MATLAB, not simply shifting it.*
- *Note that a period of* $T = 1$ *is equivalent to* 64 *indices due to the chosen resolution.*

Exercise 2. **The B-Spline scaling function**

a. *Which B-Spline scaling function should be used in order to reproduce polynomials of maximum degree 3?*

b. *Find the dual-basis of the scaling function and write a function that calculates coefficients to reproduce polynomials of degree 0 to 3:*

```
name    get_cmn_bspline
arg1    shift number : Int
arg2    polynomial order : Int
output  coefficient : float
```

c. *Compute the coefficients to reproduce polynomials of degree 0 to 3 for $n = 0, \dots, 32 - L$ where L is the support of the kernel. Plot the results with the shifted kernels and the reproduced polynomials as in the previous exercise.*

Hint:

- *Use the Daubechies formula for spectral factorization in order to find the dual-basis.*

---

4 The annihilating filter method

Assume we observe a discrete signal $\tau[m]$ of the form $\tau[m] = \sum_{k=0}^{K-1} a_k t_k^m$ and assume we want to retrieve the locations t_k and the weights a_k. One efficient way of determining these parameters is through the annihilating filter. Consider the discrete filter $h[m]$ given by z-transform

$$H(z) = \prod_{k=0}^{K-1} (1 - t_k z^{-1}). \tag{2}$$

Clearly, the convolution between h and τ will result in a zero output as the t_ks are the roots of the filter (for more details, we refer to Section 5.2.1 of the lecture notes). Therefore h is called annihilating filter of τ. In practice, the filter can be found by posing $h[0] = 1$ and solving the following system of equations

$$\begin{bmatrix} \tau[K-1] & \tau[K-2] & \dots & \tau[0] \\ \tau[K] & \tau[K-1] & \dots & \tau[1] \\ \vdots & \vdots & \ddots & \vdots \\ \tau[N-1] & \tau[N-2] & \dots & \tau[N-K] \end{bmatrix} \begin{pmatrix} h[1] \\ h[2] \\ \vdots \\ h[K] \end{pmatrix} = - \begin{pmatrix} \tau[K] \\ \tau[K+1] \\ \vdots \\ \tau[N] \end{pmatrix}.$$

The knowledge of the filter is sufficient to determine the t_ks as they are the roots of the polynomial in (2). The weights a_k can be determined using K samples of $\tau[m]$. These form a classic Vandermonde system

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ t_0 & t_1 & \dots & t_{K-1} \\ \vdots & \vdots & \ddots & \vdots \\ t_0^{K-1} & t_1^{K-1} & \dots & t_{K-1}^{K-1} \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{K-1} \end{pmatrix} = \begin{pmatrix} \tau[0] \\ \tau[1] \\ \vdots \\ \tau[K-1] \end{pmatrix}$$

which has unique solution given that the t_ks are distinct.

Exercise 3. **The annihilating filter**

a. Write a function that finds the annihilating filter $h[m]$ for a given set of signal moments $\tau[m]$:

```
name    annihilating_filter
arg1    1D array of signal moments : np.ndarray
arg2    filter order, K : Int
output  annihilating filter, h[m] : np.ndarray
```

You may assume that there are always at least $2K$ signal moments

b. Write a function that determines the innovation parameters a_k and t_k for $k = 1, \ldots, K$ from the annihilating filter generated by the function in part (a):

```
name    innovation_params
arg1    annihilating filter, h[m] : np.ndarray
arg2    filter order, K : Int
output  2D array, a_k along first row, t_k along second row  : np.ndarray
```

c. Apply your functions to the signal moments, $\tau[m]$, provided in the project folder. Retrieve the locations t_k and the amplitudes a_k. Remark: we have set $K = 2$. You can load moments as a numpy array from the data *module using* data.tau.

---

5 Sampling Diracs

All the necessary pieces for implementing an end-to-end algorithm capable of recovering sampled streams of Diracs have been written. Indeed, consider a signal of type $x(t) = \sum_{k=0}^{K-1} a_k \delta(t - t_k)$. Then the locations t_k and the amplitudes a_k can be recovered from the samples by applying the annihilating filter method to

$$\begin{aligned} s[m] &= \sum_n c_{m,n} y[n] \\ &= \langle x(t), \sum_n c_{m,n} \varphi(t-n) \rangle \\ &= \int_{-\infty}^{\infty} x(t) t^m dt \\ &= \sum_{k=0}^{K-1} a_k t_k^m \end{aligned}$$

where the last equality derives from the fact that $x(t)$ is a stream of Diracs and $\int_{-\infty}^{\infty} f(t)\delta(t - t_k)dt = f(t_k)$.

Exercise 4. **Sampling Diracs**

a. Create a stream of Diracs where $K = 2$. Filter the Diracs with an appropriate Daubechies scaling function, then sample the resulting signal. For the scaling function, using a sampling period of $T = 1$. Plot the resulting samples.

b. Reconstruct the continuous time Diracs from the samples. Compare this result with the ground truth.

Exercise 5. **Reconstruction**

We have sampled a stream of Diracs $(K = 2, T = 1)$ *with a 'db4' Daubechies scaling function. The observed samples are available from the* data *module as* data.samples.

a. Use the reconstruction algorithm from the previous question to exactly recover the locations and the amplitudes of all the Diracs.

6 Reconstruction in the presence of noise

The methods developed so far are not resilient to noise, so we want to use total-least squares and the Cadzow routine to make them more resilient to noise.

Exercise 6. **Sampling in noise**

a. Using the routine you developed in Exercise 4, create a stream of Diracs where $K = 2$. *Sample the signal with an appropriate Daubechies scaling function that can reproduce polynomial up to degree* $N - 1$ *and* $N > 2K$ *using a sampling period of* $T = 1$. *Compute the moments* $s[m] = \sum_n c_{m,n} y[n]$ *with* $m = 0, 1, .., N - 1$ *and* $N > 2K$. *Add zero-mean Gaussian noise to the moments* $s[m]$ *leading to* $\hat{s}[m] = s[m] + \epsilon[m]$. *Use the command np.random.randn to generate Gaussian noise. Choose different values for the variance* σ^2.

Guideline:

- *To generate zero-mean Gaussian noise you can use the numpy randn function:*

```
> import numpy as np
> noise = np.random.randn(N)
```

where N *is the number of samples of noise*

Exercise 7. **Reconstruction in noise**

1. *Write a function that performs reconstruction on a set of signal moments using TLS denoising:*

```
name    tls
arg1    1D array of signal moments : np.ndarray
arg2    Number of Diracs, K : Int
output  Dirac times and magnitudes : np.ndarray * np.ndarray
```

2. *Write a function that performs reconstruction on a set of signal moments using Cadzow denoising:*

```
name    cadzow
arg1    1D array of signal moments : np.ndarray
arg2    Number of Diracs, K : Int
arg3    Number of iterations : Int
output  Dirac times and magnitudes : np.ndarray * np.ndarray
```

3. *Given the noisy moments generated in the previous exercise, use the TLS and Cadzow methods to retrieve the original two Diracs. Compare and contrast your results. Try different values of* N.

7 Application: image super-resolution

You have just implemented a new sampling algorithm capable of perfectly recovering certain classes of signals that have been sampled with a rate below the Nyquist rate. This new scheme has a wide variety of applications ranging from wideband communications to image super-resolution. It is this last application that we are now asking you to implement.

Suppose that we have access to N low-resolution cameras that are all pointing to the same fixed scene. The positions of each camera is *unknown*. The goal of image super-resolution is to generate a single high-resolution image, called the super-resolved image, by fusing the set of low-resolution images acquired by these cameras. The super-resolved image (SR) has therefore more pixels than any of the low-resolution image (LR), but more importantly, it has also more details (*i.e.* it is not a simple interpolation). This is made possible because each camera observes the scene from a different viewpoint.

Image super-resolution can usually be decomposed into two main steps (see Figure 1):

1. Image registration: the geometric transformations between the LR images are estimated so that they can be overlaid accurately.

2. Image fusion and restoration: the data from each LR image is fused on a single high-resolution grid and missing data is estimated. Restoration finally enhances the resulting image by removing any blur and noise.

In the next exercise, you are asked to write in Matlab the image registration function of an image super-resolution algorithm. The approach used to register the LR images is based on the notion of continuous moments and is explained next. The fusion/restoration step is already written for you.

Suppose that the i-th camera observes the scene:

$$f_i(x,y) = f_1(x + dx_i, y + dy_i) \quad i = 1, \ldots, N,$$

where $f_1(x,y)$ is the scene observed by the first camera taken as the reference. We therefore have $(dx_1, dy_1)^T = (0,0)^T$. We have implicitly assumed here that the geometric transformations between the views are 2-D translations. At each camera, the point spread function $\varphi(x,y)$ of the lens is modeled by a 2-D cubic B-spline and blurs the observed scene $f_i(x,y)$ (see Figure 2). B-splines are functions with compact support and satisfy Strang-Fix conditions. Therefore, similarly to Equation (1), there exists a set of coefficients $\left\{c_{m,n}^{(p,q)}\right\}$ so that:

$$\sum_{m\in\mathbb{Z}}\sum_{n\in\mathbb{Z}} c_{m,n}^{(p,q)} \varphi(x-m, y-n) = x^p y^q,$$

The blurred scene $f_i(x,y) * \varphi(-x,-y)$ is then uniformly sampled by the CCD array which acts as an analog to discrete converter. The output discrete image of the i-th camera has then the samples $S_{m,n}^{(i)}$:

$$S_{m,n}^{(i)} = \langle f_i(x,y), \varphi(x-m, y-n) \rangle$$

By using a similar development as written in Section 5, it can be shown that:

$$m_{p,q} = \int\int f(x,y)\, x^p y^q dx dy \tag{3}$$

$$= \sum_m \sum_n c_{m,n}^{(p,q)} S_{m,n} \tag{4}$$

Equation (3) is the definition of the geometric moments $m_{p,q}$ of order $(p+q)$, $p,q \in \mathbb{N}$, of a 2-D continuous function $f(x,y)$. Equation (4) shows that it is possible to retrieve the exact

moments of the observed continuous scene from its blurred and sampled version by computing a linear combination with the proper coefficients. Knowing those moments, we can compute the barycenter of $f(x, y)$:

$$(\overline{x}, \overline{y})^T = \left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}} \right)^T$$

If $f_i(x, y) = f_1(x + dx_i, y + dy_i)$, then $(dx_i, dy_i)^T$ can be retrieved from the difference between the barycenter of f_1 and the barycenter of f_i:

$$(dx_i, dy_i)^T = (\overline{x}_i, \overline{y}_i)^T - (\overline{x}_1, \overline{y}_1)^T$$

Exercise 8. Image super-resolution

In this exercise, you have access to $N = 40$ *low-resolution color images of* 64x64 *pixels. The goal is to generate a super-resolved color image of* 512x512 *pixels. The images observe the same scene and have been taken from different viewpoints. We assume that the geometric transformations between the views are 2-D translations.*

a. *Write a function that registers each image with respect to the first image. This function should return the estimated shift of each image:*

```
name    get_barycenters
output  barycenters_x and barycenters_y relative  : np.ndarray * np.ndarray
        to first low resolution image
```

Note: the function has no arguments. $barycenter_x[k, l]$ *should be the barycenter in the x axis for the* l^{th} *layer of the* k^{th} *low resolution image relative to the* 1^{st} *low resolution image. Thus* $barycenter_x[0, l] = 0$. *As stated in the guidelines, you should threshold each low resolution image with an appropriate threshold to increase SNR (use pywt.threshold with* mode='hard'*).*

You can see the results of your registration by running image_fusion.superres(get_barycenters)

b. *What is the correct threshold for pixel values and how did you determine this? Use a principled way of determining the correct threshold.*

What is the PSNR achieved with your implementation?

c. *Can you pre/postprocess the data to achieve a higher PSNR?* ***You must submit the original implementation of get_barycenters but you can discuss any improvement you make in the written report***

Guidelines and tips:

1. *The main python module is* image_fusion.py. *This module handles loading relevant data, image fusion, and plotting. You can edit this file if you want but all submissions will be tested against the original.*

2. *Functions written in your submission script can use variables and methods from* image_fusion.py *by accessing them with the following convention* image_fusion.x *where* x *is a variable in* image_fusion *or* image_fusion.f() *where* f *is a function in* image_fusion

3. *The variables and functions from* image_fusion *of interest to you are:*

name	Purpose
n_sensors	Number of low resolution images
n_layers	Number of layers per image (RGB)
lr_ims	Low resolution images in a list, each image has the layer axis last
hr_ref_im	High resolution reference image, layer axis last
lr_size	Width/height of low resolution images
hr_size	Width/height of high resolution reference image and target resolution
coeff_0_0 coeff_0_1 coeff_1_0	Three matrices of size $64x64$: You will use these to compute the continuous moments from the samples of your LR images. *Coef_0_0* is used to compute the continuous moment $m_{0,0}$, *Coef_1_0* is for $m_{1,0}$ and *Coef_0_1* is for $m_{0,1}$.
superres(get_barycenters)	A function that performs image fusion. To test your barycenter finding function, pass it to superres and superres will generate the corresponding reconstruction and plot it. The PSNR and the super resolved image are returned from this function.

These can be accessed using dot notation. For example, if you want to use the superres function you can call it like so: image_fusion.superres(get_barycenters)

4. *Consider first the Red layer of each LR images, compute the continuous moments and find the 2-D translations. Then only, consider the Green layer, and then the Blue one. For each LR image, you should then have retrieved three similar 2-D translations.*

5. *In order to reduce the noise created by the background of the image and get accurate moments, it is required that you select a threshold for each layer and set to 0 all the samples that are below it.*

6. *All images are scaled such that sample values are between 0.0 and 1.0*

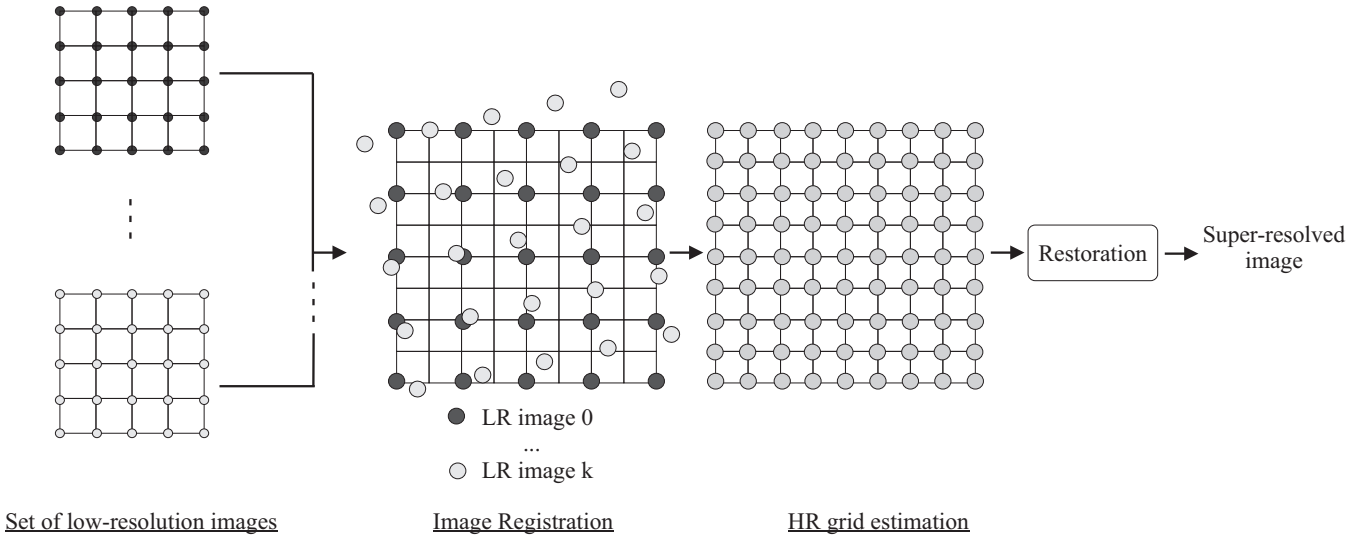
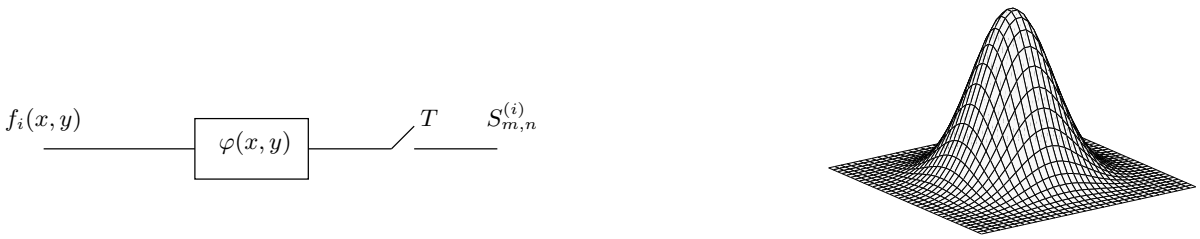


Figure 1: Image super-resolution principles



(a) Acquisition model.

(b) 2-D B-spline $\varphi(x,y)$.

Figure 2: Model for each of the N cameras.