1. Simple Fitting

Generate some data of the form

$$y(x) = a \exp(\alpha x) + k$$

for the following choice of constants: $a = 2$, $\alpha = -0.75$, $k = 0.1$, in the range $x \in [0, 4]$. Add zero-mean Gaussian noise with amplitude 0.1 to this signal to create a noisy version of the signal.
Then, compare the results of fitting the noisy data using a least-squares fit to the analytical form, as well as splines (from `scipy.interpolate`) and polynomial fits (from numpy) of orders 0, 1 and 2.

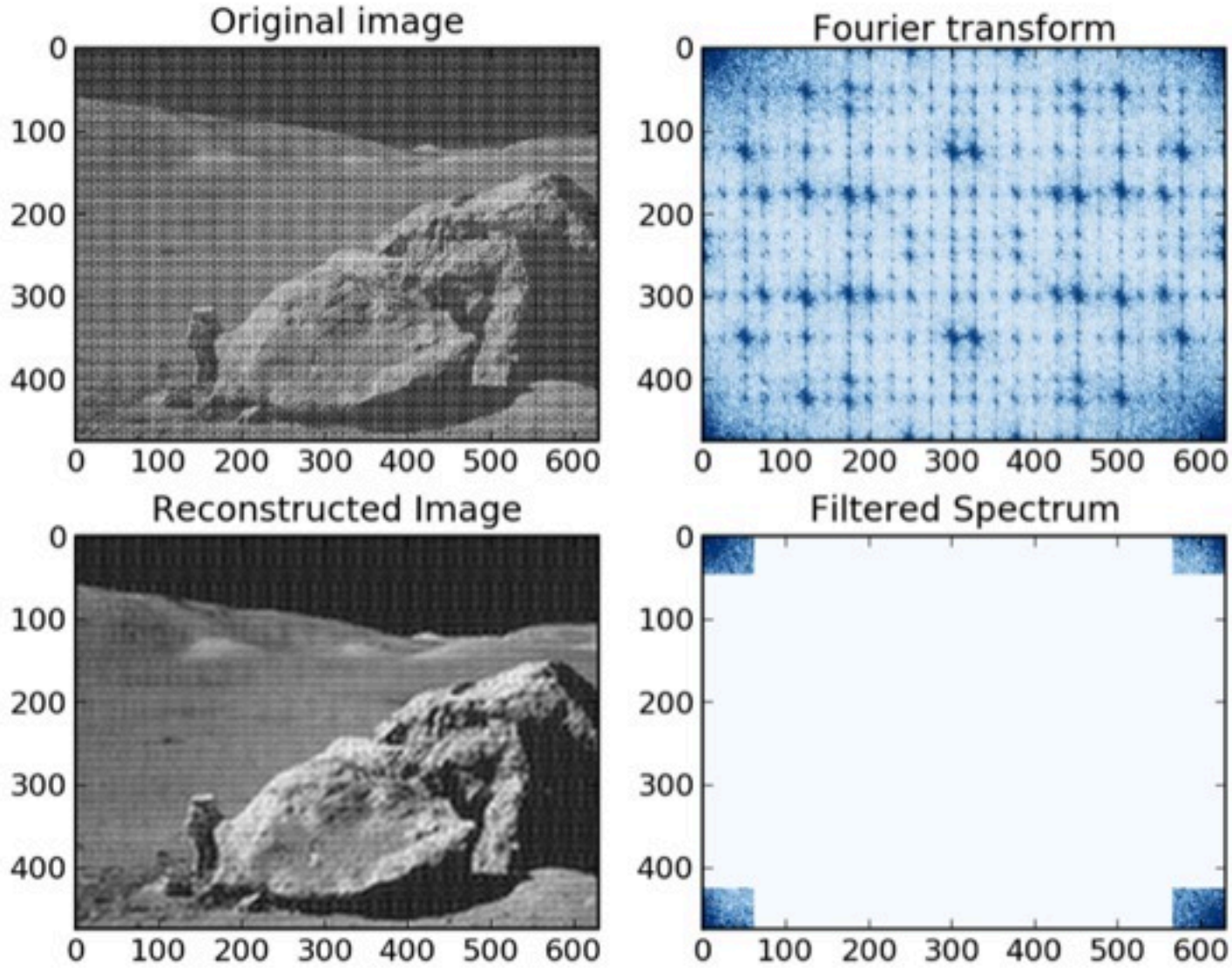
2. FFT Denoising

background:

The convolution of an input with with a linear filter in the temporal or spatial domain is equivalent to multiplication by the Fourier transforms of the input and the filter in the spectral domain. This provides a conceptually simple way to think about filtering: transform your signal into the frequency domain, dampen the frequencies you are not interested in by multiplying the frequency spectrum by the desired weights, and then apply the inverse transform to the modified spectrum, back into the original domain. In the example below, we will simply set the weights of the frequencies we are uninterested in (the high frequency noise) to zero rather than dampening them with a smoothly varying function. Although this is not usually the best thing to do, since sharp edges in one domain usually introduce artifacts in another (e.g. high frequency "ringing"), it is easy to do and sometimes provides satisfactory results.

After reading the image file moonlanding.png, try to produce images like the ones in the figure on the next page.

We will describe the process here and provide some hints as to what you need to think about. The image in the upper left panel of the Figure is a grayscale photo of the moon landing. There is a banded pattern of high frequency noise polluting the image. In the upper right panel we see the 2D spatial frequency spectrum. The FFT output in the numpy.fft module is packed with the lower frequencies starting in the upper left, and proceeding to higher frequencies as one moves to the center of the spectrum (this is the most efficient way numerically to fill the output of the FFT algorithm). Because the input signal is real, the output spectrum is complex and symmetrical: the transformation values beyond the midpoint of the frequency spectrum (the Nyquist frequency) correspond to the values for negative frequencies and are simply the mirror image of the positive frequencies below the Nyquist (this is true for the 1D, 2D and ND FFTs in numpy).
You should compute the 2D spatial frequency spectra of the luminance image, zero out the high frequency components, and inverse transform back into the spatial domain. You can plot the input and output images with plt.imshow(), but you should observe that if you show the power spectrum (the absolute value of the FFT) directly, you will only see white, and not the image in the Figure's upper right panel. This is due to the fact that the power spectrum has a small number of pixels with extremely high amplitude, which completely swamp the contrast (you can verify this by playing with a histogram of the data). You will thus need to clip the limits of the color range to a small range of values where most of the power actually lives (say 95%).



hints:

- The upper right panel is obtained after finding the range of values that contain 95% of the power. For this, the `mlab.prctile()` function will be useful.
- The `numpy.fft` module contains the necessary FFT routines for this exercise.
- In Python, a complex number z has `z.real` and `z.imag` attributes for its real and imaginary parts.

Problem 3

The inverse of a square matrix A is the matrix B such that AB = I, where I is the identity matrix consisting of ones on the diagonal, and zeros everywhere else. In *numpy*, the inverse can be calculated with the *numpy.linalg.inv* method.

We would like you reproduce the *numpy.linalg.inv* routine. Provided a square matrix, your routine *my_matrixinv* should verify the matrix is invertible (i.e. the determinant != 0), and then return the matrix inverse.