MATH-UA 252/MA-UY 3204 - Fall 2022 - Homework #7

Problem 1. Let $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$, where $m < m$. In general, underdetermined linears systems admit either no solutions, or infinitely many solutions. For instance, if our goal is to find some x satisfying $Ax = b$, and we try to solve the least squares problem:

$$\text{minimize} \quad \|Ax - b\|_2^2 \tag{1}$$

we run into trouble with the pseudoinverse $A^\dagger = (A^\top A)^{-1} A^\top$, since $A^\top A$ is rank deficient (why?). The standard way to get around this problem is to *regularize* the problem. As you saw in the midterm, if you instead solve the regularized least squares problem:

$$\text{minimize} \quad \|Ax - b\|_2^2 + \sigma \|x\|_2^2 \tag{2}$$

where $\sigma > 0$.

Download the Nile time series data set `nile.csv`. The first column is the year and the second column is the river flow. Load this data set into numpy and eliminate some values (e.g., assume 10 entries in the middle of the time series are missing). We will try to use a discrete cosine transform (DCT) basis to recover the missing values. Let n be the number of points in the time series, and use the following code:

```
X = scipy.fft.dct(np.eye(n), norm='ortho')
```

to create an orthogonal DCT matrix. Note that the columns of X will correspond to cosines of progressively higher frequency (you can plot a few to see). Let f be the vector of flow rates. Then $\alpha = X^{-1} f = X^\top f$ gives the coefficients of f in the DCT basis.

Let I be an index vector giving the *non*-missing values of the time series. Then $X_I \in \mathbb{R}^{|I| \times n}$ restricts the columns of the DCT matrix to those known values. The linear system $X_I \alpha = f_I$ is an underdetermined linear system. Try to solve this problem using least squares and regularized least squares. Plot the resulting flow rate versus the year. Does this do a good job of filling in the missing values? Does ℓ_2 regularization help? Explain your findings.

Problem 2. The *basis pursuit* problem is the following minimization problem:

$$\begin{aligned} &\text{minimize} && \|x\|_1 \\ &\text{subject to} && Ax = b, \end{aligned} \tag{3}$$

where $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, and $\|x\|_1 = \sum_{i=1}^n |x_i|$. Typically, the linear system $Ax = b$ is *underdetermined*, so that $m < n$. Note that minimizing $\|x\|_1$ promotes *sparsity* in x—there will tend to be more zero entries. Our goal will be to solve:

$$\begin{aligned} &\text{minimize} && \|\alpha\|_1 \\ &\text{subject to} && X_I \alpha = f_I, \end{aligned} \tag{4}$$

in order to compute a sparse solution to the underdetermined linear system.

1. Prove that the basis pursuit problem (3) is equivalent to the following standard form LP:

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^n u_i + \sum_{i=1}^n v_i \\ &\text{subject to} && \begin{bmatrix} \Phi_I & -\Phi_I \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = f_I, \\ & && u \geq 0 \\ & && v \geq 0 \end{aligned} \tag{5}$$

2. Solve this LP using `scipy.optimize.linprog`.

3. Compare your results with what you got in Problem 1.

Problem 3. Show that the dual LP of the basis pursuit problem is:

$$\begin{aligned} \text{maximize} \quad & f_I^\top \lambda \\ \text{subject to} \quad & -\mathbb{1} \leq \Phi_I^\top \lambda \leq \mathbb{1}, \end{aligned} \tag{6}$$

where $\mathbb{1}$ is a vector with each of its components equal to 1.

Problem 4. Set up and solve the dual LP for the Nile time series basis pursuit problem. Explain how to recover the primal variables from the dual variables and do so. Check that your results agree.

Bonus problem. Now we want to use basis pursuit to do image inpainting. We will eliminate a rectangular region from a grayscale image and set up a basis pursuit problem to recover the missing values using the values in a small margin around the rectangle.

You can get a test image of a raccoon with the following code snippet:

```
img = (scipy.misc.face()/255).mean(2)
```

You should assume the region we delete consists of indices in the rectangle $[i_0, i_1) \times [j_0, j_1) \subseteq \mathbb{Z}^2$. For recovery, we let $p > 0$ and consider the indices:

$$I = \Big([i_0 - p, i_1 + p) \times [j_0 - p, j_1 + p)\Big) \setminus \Big([i_0, i_1) \times [j_0, j_1)\Big) \subseteq \mathbb{Z}^2. \tag{7}$$

As before, we'll use the DCT basis. But this time, since we're working with a two-dimensional signal, we'll use a DCT basis for each dimension. This can be accomplished using a Kronecker product. Let $m = i1 - i0$ and $n = j1 - j0$. To set up a 2D DCT basis on an $(m + 2p) \times (n + 2p)$ grid, use:

```
dct = lambda n: scipy.fft.dct(np.eye(n), norm='ortho')
X = np.kron(dct(m + 2*p), dct(n + 2*p))
```

Note *very* carefully: this assumes a row-major ordering of the grid nodes. Here's how the 2D and 1D orderings relate:

```
# generate some random low-frequency DCT coefficients in 1D
alpha = np.zeros((m + 2*p)*(n + 2*p))
alpha[alpha.size//8] = np.random.randn(alpha.size//8)

# compute the linear combination---this is still 1D
f = X@alpha

# reshape the values of f into a 2D array
F = f.reshape(m + 2*p, n + 2*p)

# plot the values
plt.figure()
plt.imshow(F)
plt.show()
```

After setting up and solving the (primal) basis pursuit problem for inpainting, compare the original and recovered images. Make plots showing what you come up with. Additionally, compute the root mean-squared error (RMSE) and see how it varies for different rectangles and different values of p. What do you observe?

Two more things to experiment with:

1. Try inpainting a smoother two-dimensional signal. An interesting choice would be a random combination of the columns of X (the 2D DCT basis). In this case, in principal, exact recovery should be possible under certain circumstances.

2. Try using a basis other than the DCT. A good choice is a 2D Wavelet basis, such as the Haar basis. See, e.g. this Wikipedia page.

How do the results compare?