

## Image In-painting by Linear Programming

This problem is about in-painting (or de-noising) an image with a large number of missing pixels at random locations via solving a linear program. The idea is to fill in the missing pixels so that the resulting image has a small total variation (explained below).

We will use Matlab's linear programming solver `linprog` which is designed to handle vector-valued variables. On the other hand, images are matrices. Therefore, we need to treat images as vectors, and at the end transform the solution vector back into an image.

Let the matrix  $X^0 \in \mathbb{R}^{n \times n}$  represents a square gray-scale image (rectangular and color images can be similarly handled as well) whose pixel values are only partially and approximately available on a subset of indices (or pixel locations). Let the vector  $\mathbf{x}^0 \in \mathbb{R}^{n^2}$  be the vectorized version of  $X^0$ , obtained by stacking up the  $n$  columns of  $X^0$  (in natural order) to form a long  $n^2$ -vector  $\mathbf{x}^0$  (in a mathematics notation  $\mathbf{x}^0 = \text{vec}(X^0)$ ; or in MATLAB `x0 = X0(:)`).

We know neither  $X^0$  nor  $\mathbf{x}^0$ . The data available to us is a matrix  $\hat{X} \in \mathbb{R}^{n \times n}$  corresponding to the vector  $\hat{\mathbf{x}} \in \mathbb{R}^{n^2}$  whose elements satisfy

$$\hat{\mathbf{x}}_i = \begin{cases} \mathbf{x}_i^0 + \text{noise}, & i \in \Omega, \\ 255, & \text{otherwise,} \end{cases}$$

where  $\Omega$  is a subset of  $\{1, 2, \dots, n^2\}$ , and 255 is the largest pixel value in an 8-bit gray scale color scheme, representing a lost pixel value that is just filled with a white dot. See Figure 1, for example.

Left: Original. Right: Contaminated



Figure 1: Available data for an image.

Can we reconstruct  $\mathbf{x}^0$ , rather accurately, from the partial and approximate data  $\hat{\mathbf{x}}_\Omega$  (consisting of all the available pixel values with indices in  $\Omega$ )? The answer is a sound YES provided that the subset  $\Omega$  is “sufficient” in a certain sense and the noise level is sufficiently low. How do we do it?

Let  $m = |\Omega|$  be the total number of indices in  $\Omega$ , and  $S \in \mathbb{R}^{m \times n^2}$  be the sub-matrix of the  $n^2$  by  $n^2$  identity matrix consisting of the rows whose indices are in  $\Omega$ . We will solve the following  $\ell_1$ -minimization problem to reconstruct the

image:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \|E\mathbf{x}\|_1 \\ \text{s.t.} \quad & S\mathbf{x} = \hat{\mathbf{x}}_\Omega \\ & \mathbf{x} \geq 0. \end{aligned} \tag{1}$$

What are the sizes of  $S$ ? How many nonzero elements it has?

The constraint equation  $S\mathbf{x} = \hat{\mathbf{x}}_\Omega$  insists that the recovered image (represented by  $\mathbf{x}$ ) should match the pixel values of  $\hat{\mathbf{x}}$  on the subset  $\Omega$  which should be correct within a noise level. In addition, we impose nonnegativity because pixel values are nonnegative. By solving this problem, we fill in the missing pixel values while making the objective function  $\|E\mathbf{x}\|_1$  small. The function  $\|E\mathbf{x}\|_1$  is called a total variation (or TV) of the image which is represented by  $\mathbf{x}$ . We choose to minimize this TV function simply because natural images usually have small total variations.

We now discuss the matrix  $E$ . First consider the one-dimensional difference matrix

$$D = \begin{pmatrix} 1 & -1 & 0 & & \\ 0 & 1 & -1 & 0 & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{pmatrix} \in \mathbb{R}^{(n-1) \times n}. \tag{2}$$

For a 2-dimensional image  $X$ , good results can be obtained by taking differences in both horizontal and vertical directions, giving two matrices  $DX$  and  $XD^T$ . We vectorize both  $DX$  and  $XD^T$ , and then stack the two into a long vector. By taking the  $\ell_1$ -norm of the long vector, we get a version of two-dimensional TV function  $\|E\mathbf{x}\|_1$  in (1) (other versions do exist).  $E$  is the matrix representation of the above actions on the image  $\mathbf{x} = \text{vec}(X)$ ; that is,

$$E\mathbf{x} = \begin{bmatrix} \text{vec}(DX) \\ \text{vec}(XD^T) \end{bmatrix}.$$

To have an explicit form for the matrix  $E$ , it is most convenient to use matrix Kronecker product “ $\otimes$ ” (learn more of Kronecker products by yourself), which gives

$$E = \begin{bmatrix} I \otimes D \\ D \otimes I \end{bmatrix}, \tag{3}$$

where  $I$  is the  $n \times n$  identity matrix, and  $D$  is defined in (2). In MATLAB, the command `kron` performs Kronecker products (study it). What are the sizes of  $E$ ?

The optimization problem (1) is not yet a linear program, but can be equivalently transformed into one. To do so, we first simplify the objective function by writing  $E\mathbf{x} = \mathbf{u} - \mathbf{v}$  for  $\mathbf{u}, \mathbf{v} \geq 0$  where  $\mathbf{u}$  represents the positive part of  $E\mathbf{x}$  (i.e.,  $\mathbf{u}_i = (E\mathbf{x})_i$  if the latter is positive and  $\mathbf{u}_i = 0$  otherwise). Similarly,  $\mathbf{v}$  represents the absolute value of the negative part. Under this transformation,

$$\|E\mathbf{x}\|_1 = \sum_i |(E\mathbf{x})_i| = \sum_i \mathbf{u}_i + \sum_i \mathbf{v}_i = e^T \mathbf{u} + e^T \mathbf{v},$$

where  $e$  is the vector of all ones of an appropriate length.

With the simplified objective function along with the condition  $E\mathbf{x} = \mathbf{u} - \mathbf{v}$  for  $\mathbf{u}, \mathbf{v} \geq 0$ , we arrive at a linear program below, which is equivalent to (1),

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{u}, \mathbf{v}} \quad & e^T \mathbf{u} + e^T \mathbf{v} \\ \text{s.t.} \quad & E\mathbf{x} - \mathbf{u} + \mathbf{v} = 0 \\ & S\mathbf{x} = \hat{\mathbf{x}}_\Omega \\ & \mathbf{x}, \mathbf{u}, \mathbf{v} \geq 0, \end{aligned} \tag{4}$$

which has 3 nonnegative, vector-valued variables  $\mathbf{x}, \mathbf{u}, \mathbf{v}$ .

In order to call the Matlab linear program solver `linprog`, we need to put (4) into a standard form acceptable to `linprog`; that is

$$\begin{aligned} \min_{\mathbf{z}} \quad & f^T \mathbf{z} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{z} = \mathbf{b} \\ & \mathbf{z} \geq 0, \end{aligned} \tag{5}$$

where  $f^T = [0^T \ e^T \ e^T]$ ,  $\mathbf{z} = [\mathbf{x}^T \ \mathbf{u}^T \ \mathbf{v}^T]^T$ , and

$$\mathbf{A} = \begin{pmatrix} E & -I & I \\ S & 0 & 0 \end{pmatrix}. \tag{6}$$

What is  $\mathbf{b}$  in (5)? What are the sizes of  $f$  in (5)? What are the sizes of  $I$  and  $\mathbf{A}$  in (6)?

## Tasks

Write a MATLAB function to compute a reconstructed image (matrix)  $X$ , from the available data matrix  $\hat{X}$  and the index set  $\Omega$ , by solving the standard form linear program (5), or another equivalent form of it, using `linprog`. Specifically, your function has the interface

```
function X = myInpaint(Xh, Omega)
```

where the input arguments represent data  $\hat{X}$  and  $\Omega$ , respectively, and the output is the reconstructed image. The function performs the following steps:

- Figure out relevant sizes ( $n$ ,  $m$  and so on) from the two input arguments.
- Construct the  $E$  matrix according to (3) and the  $S$  matrix.
- Construct the  $\mathbf{A}$  matrix according to (6), and the  $f$  and  $\mathbf{b}$  vectors.
- Call MATLAB `linprog` to solve the linear program (5) to get  $\mathbf{z}$ .
- Extract the image  $\mathbf{x}$  from  $\mathbf{z}$  and reshape it into a matrix  $X$  for output.

Be sure that you use sparse matrices to construct  $E$ ,  $S$  and  $\mathbf{A}$ . For example, instead of using `eye` to generate identity matrices, you should use `speye` to generate sparse identity matrices.

Run the test script `test_inpaint.p` on two data sets: `data1_inpaint.mat` and `data2_inpaint.mat`. Without your code, the script runs the instructor's code `yzInpaint.p` only. All these files are available online. First experiment on the first data set (containing a 256 by 256 image) until your code works. Then try the second data set (containing a 512 by 512 image). **Finally, run the test script on both images to get a total score.**