# Digital Signals

**Assignment 4: Due 7/27/2016**

Nicholas Dwork

## 1 Final Projects

**Showing Moving Objects** From the *Making People Disappear* project, you know how to get an image of just the background. If you do a pointwise subtraction between an image and it's background, you'll get an image with large values just where there are movers.

**Part a** Use the data from the previous project. Make an image of the result of the subtraction.

**Part b** For those pixels in the subtraction image where the resulting magnitude is high, give them their original color. (Make the value of all other pixels 0.)

**Part c** There are morphological operations called `dilate` and `erode`. Use dilation and erosion to try to fill in any holes in the movers and eliminate movement in the trees.

**Lucas-Kanade Optical Flow** The goal of this project will be to have the computer determine the velocity of a mover. The key is to realize that if something is imaged at point $(x, y)$ in image 1 and that thing moves to point $(x + \Delta x, y + \Delta y)$ in image 2, then the image intensities will be the same. Think of a video as a 3D array, where the third dimension is time. Let $t$ denote the time of the first image, and $t + \Delta t$ be the time of the second image. Then

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t).$$

We can linearize the above equation using the multivariable chain rule (don't worry if you don't understand what this means; you can just use the equation shown later without understanding where it comes from). When we do that, we get the following equation:

$$I(x, y, t) = I(x, y, t) + I_x(x, y, t)\Delta x + I_y(x, y, t)\Delta y + I_t(x, y, t)\Delta t, \tag{1}$$

where $I_x$ is the derivative of $I$ in the $x$ direction, $I_y$ is the derivative of $I$ in the $y$ direction, and $I_t$ is the derivative of $I$ in the temporal dimension. Note that we have $I(x, y, t)$ on both sides of equation (1), so we can cancel that out:

$$-I_t(x, y, t)\Delta t = I_x(x, y, t)\Delta x + I_y(x, y, t)\Delta y, \tag{2}$$

Equation (2) is called the *Optical Flow Constraint*.

Recall the definition of the derivative of a function $f$ (again, you don't need to know this, and can just follow the formulas if you'd like):

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}.$$

We can approximate the derivative using a *forward difference approximation* as follows:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

for some finite $\Delta x$. Using this approximation, we can approximate the derivatives in equation (2). For example, the derivative in the x dimension for pixel $(i, j)$ is

$$I_x(i, j) \approx I^{(1)}(i + 1, j) - I^{(1)}(i, j),$$

where $I^{(1)}$ is the first image and $\Delta x = 1$ pixel.

Using the forward difference approximation (with $\Delta x = 1$ pixel, $\Delta y = 1$ pixel, and $\Delta t = 1$ frame), we can determine all values in equation (2) except for $\Delta x$ and $\Delta y$, which are our velocities! The vector $(\Delta x, \Delta y)$ is called the *Optical Flow Vector*. Note that equation (2) is one linear equation with two unknowns, so we don't have enough equations to solve for $\Delta x$ and $\Delta y$. We remedy this by assuming multiple pixels close to each other are moving with the same velocity.

**Part a** Use the data from the *Making People Disappear* project. Choose a pixel in the center of a moving person. Assume that the $3 \times 3$ box of pixels around your selected pixel are all moving with the same velocity. Now you have a linear system of 9 equations with 2 unknowns. Formulate this as $A\,\gamma = b$, and solve for $\gamma$.

**Part b** Repeat part (a) for a scene with multiple movers. Draw arrows on the figure indicating the velocity and direction of each mover.

**Image Denoising - Bilateral Filter** In class we saw that we can remove noise by doing a local-average, for example, with a box-car filter. This has the unfortunate side effect of blurring sharp edges in the image. Wouldn't it be better if we could blur intelligently? If there is an edge in the image, wouldn't it be better if we could blur with pixels just on the correct side of the edge? That is the goal of the *Bilateral Filter*.

**Part a** Download an image from the Internet that you like and convert it to grayscale. Add noise by creating a 2D array of random values (using numpy.random.normal) and adding it to the original image. This is your noisy image.

**Part b** The equation for a value in the filtered image is

$$\hat{I}(y) = \frac{1}{W_y} \sum_{x \in \Omega} I(x) f\left(\|I(x) - I(y)\|_2\right) g\left(\|x - y\|_2\right), \tag{3}$$

where $I$ is the noisy image, $\Omega$ is a neighborhood of $y$ (e.g. a $5 \times 5$ box centered on $y$), $f$ is a function that penalizes differences in intensity, $g$ is a function that penalizes differences in

distance (e.g. a Gaussian function), and $W_y$ is a normalizing constant. The value $W_y$ is chosen so that the weights $w = f\,g$ for all pixels in the neighborhood sum to 1.

Implement the bilateral filter to denoise your image; compare your denoised result to the original pristine image. For this project, we will let $f, g$ be zero-centered Gaussian functions defined as follows:

$$f(z) = \exp\left(-\frac{z^2}{2\,\sigma_f^2}\right).$$

**Part c** Compare the results of the bilateral filter to the boxcar average filter.

**Part d** Explain how the bilateral filter works. Why does it perform better than a boxcar average filter?

**Super-resolution** If one has multiple images of an object where the camera has been shifted slightly between images, those images can be combined into a single image with higher resolution. The algorithms for doing so are called *Super-resolution* algorithms. For this project, you will use the data located at the following link: `https://nicholasdwork.com/teaching/si2016/session2/hmwk4/crustacean.zip`

A camera took image `crustacean_1.jpg`, shifted by half a pixel in both directions, and then took the image `crustacean_2.jpg`. Let $x$ denote the high resolution image (extended into a 1D vector). Let $y_1, y_2$ denote the first and second captured image, respectively. Note that $x$ is twice the size of $y_i$. Then

$$y_1 = D_1\,B\,x_1$$
$$y_2 = D_2\,B\,x_2,$$

where $B$ is a $3 \times 3$ boxcar averaging blur, and $D_i$ is the appropriate downsampling matrix. These two equations can be combined into a single equation $A\,x = y$; solve for $x$. Compare your super-resolved image to the original high resolution image `crustacean_orig.jpg`.

Note that $B, D_1, D_2$ have a whole lot of zeros in them. Matrices with mostly zeros are called *sparse* matrices. There is a special variable in Python to store a sparse matrix; it stores the matrix in a way that's much more memory efficient. If your matrices are too large, it may be necessary to store them as sparse matrices.

**Image Fusion** For this project, we will use the data located at the following link: `https://nicholasdwork.com/teaching/si2016/session2/hmwk4/fusionData.zip`

You have been given an image captured with a visible camera (red, green, and blue), and an image captured with an infrared (IR) camera. As you can see, you can't see the gun in the RGB image. And you can't see details of the face in the IR image. The goal of this project is to see both the information from both images; that is, the goal is to see as much information as possible from both images in a single color image.

**Part a** One way to see details from both images is to visualize a point on the line connecting the two images (review the "alpha-blending" problem of homework 3). This works surprisingly well; give it a shot!

**Part b** We will now describe a more sophisticated algorithm that does a better job of retaining the information from both images.

Recall that if $Y$ is a grayscale image of size $M \times N$, then we can make a vector of size $MN$ by concatenating the columns of $Y$. This vector has the exact same information as the original image. If $Y$ is a color image, then we can make a vector of size $3MN$ where the first third is made of the red pixels, the second third made of the green pixels, and the last third made of the blue pixels. In this project, we are given four channels of data, and we can only display three channels of information. We can concatenate the data from all four images into a single vector of size $4MN$. Denote this vector as $y$.

We only have three channels to display the data from R,G,B, and IR; this is because we only have three cones in our eyes: red, green, and blue. So our final image (the fused image) will be of size $3MN$; denote this vector as $x$.

We want the vector $x$ that minimizes all of the following:

$$\|F_R - R\|_2^2, \quad \|F_G - G\|_2^2, \quad \|F_B - B\|_2^2, \quad \text{and} \quad \|f(F_R, F_G, F_B, R, G, B) - IR\|_2^2 \tag{4}$$

where $f(F_R, F_G, F_B, R, G, B) = 0.3\,F_R + 0.6\,F_G + 0.1\,F_B$. (You might be wondering where the $0.3, 0.6, 0.1$ came from. The amount of light that the eye detects is called *luminance*; it's the weighted average of each of the cones according to the weights specified.) The multiple objectives specified in equation (4) can be combined into a single expression: $\|A\,x - y\|_2^2$. Find $x$ and display it.

**Space Vehicle Propulsion** (Credit: Dr. Boyd) Vehicle propulsion can be modeled using the following discrete time linear dynamical system:

$$x(t+1) = Ax(t) + Bu(t), \quad y(t) = C\,x(t), \quad \text{for } t = 0, 1, 2, \ldots$$

where $A$ is called the *dynamics matrix*, $B$ is called the *input matrix*, $C$ is called the *output matrix*, and $t$ is an index of time. The *output* (which may incorporate velocity, angular velocity, and many other characteristics) is denoted by $y$. The vector $x$ is an intermediate entity in this model, and is referred to as the *state vector*. The *input* is denoted by $u$. A good metric for the amount of fuel consumed from time $t = 0$ to $t = T - 1$ is

$$\sum_{t=0}^{T-1} \|u(t)\|_2^2. \tag{5}$$

That is, the larger the input, the more fuel is consumed. We will call this value the "fuel consumed".

Generally, we would like to find the input sequence $u(t), t = 0, 1, \ldots, T-1$ that achieves our desired output with a minimum of fuel expended.

For this project, $A \in \mathbb{R}^{16\times16}, B \in \mathbb{R}^{16\times2}$, and $C \in \mathbb{R}^{2\times16}$. The system starts from the zero state, i.e. $x(0) = 0$. The goal is to find an input $u$ that results in $y(t) \to y_{des} = (1, -2)$ for all $t \geq T$ (i.e. exact convergence after $T$ steps). The data (and the initial code) for this project can be found at the following link: `https://nicholasdwork.com/teaching/si2016/session2/hmwk4/ss_small_input_data.py`

**Part a** Suppose that the system is in steady state, i.e. $x(t) = x_{ss}$, $u(t) = u_{ss}$, and $y(t) = y_{des}$. Find $u_{ss}$ and $x_{ss}$.

**Part b** Let $u^\star(t)$ for $t = 0, 2, \ldots, T-1$ be the input that minimizes (5) and yields $x(T) = x_{ss}$ (the value found in part a). Note that if $u(t) = u^\star(t)$ for $t = 0, \ldots, T-1$, and then $u(t) = u_{ss}$ for $t = T, T+1, \ldots$ then $y(t) = y_{des}$ for $t \geq T$. In other words, we have exact convergence to the desired output in $T$ steps, and we retain that desired output thereafter.

For the three cases $T \in \{100, 200, 500\}$, find $u^\star$ and the amount of fuel consumed for each case. For each of these three cases, plot $u$ and $y$ versus $t$.

**Urinalysis Color Matching** In this project, you will replicate the color matching algorithm described in [1]. You can see a video of how the device operates here: `https://www.youtube.com/watch?v=CnlnwLq95o0&feature=youtu.be`.

The slip-chip mechanism described in [1] is used to deposit urine onto the dipstick pads when then half circles are complete. You can see this happen in the frames of the data located at. `https://nicholasdwork.com/teaching/si2016/session2/hmwk4/urinalysisData.zip`.

The frames are 0.5 seconds apart. The urine is deposited onto each pad of the urinalysis dipstick when the half circles are complete. Once the urine is deposited onto each pad, the chemical processes begin; these processes may or may not change the color of the pad depending on the chemical makeup of the urine. The goal of this project is to determine the result of each chemical test by comparing the color of the test to the color in the color chart located in the zip file.

From top to bottom in the frames, the tests are GLU, BIL, KET, SG, BLO, pH, PRO, URO, NIT, LEU. The test must be read a specific amount of time after the urine is deposited. For each test, the readout times are 30, 30, 40, 45, 60, 60, 60, 60, 60, 120 seconds, respectively.

**Part a** As we discussed in class, each pixel of a color image is composed of three values: red, green, and blue. Thus, any color can be thought of as a vector in $\mathbb{R}^3$. For each chemical test: (1) identify the correct frame to analyze based on the amount of time that has elapsed, (2) determine the color of a pixel in the center of the pad, (3) compare that color vector to all relevant colors in the colorchart using the Pearson Correlation Coefficient to determine the result of the urinalysis test.

**Part b** Rather than using a single pixel in the center of the dipstick pad, average a $3 \times 3$ block of pixels to determine the resulting color of the test.

**Part c** Can you think of a more useful color comparison metric? Perform the same test using this other metric? Why is it more useful?

**Integrated Circuit Cell Placement** (Credit: Dr. Boyd) We consider an Integrated Circuit (IC) that contains $N$ cells or modules that are connected by $K$ wires. We model a cell as a single point in $\mathbb{R}^2$ (which gives its location on the IC) and ignore the requirement that the cells must not overlap. The positions of the cells are

$$(x_1, y_1), \quad (x_2, y_2), \quad \ldots \quad (x_N, y_N),$$

i.e., $x_i$ gives the $x$-coordinate of cell $i$, and $y_i$ gives the $y$-coordinate of cell $i$. We have two types of cells: fixed cells, whose positions are fixed and given, and free cells, whose positions are to be determined. We will take the first $n$ cells, at positions

$$(x_1, y_1), \quad \ldots \quad (x_n, y_n)$$

to be the free ones, and the remaining $N - n$ cells, at positions

$$(x_{n+1}, y_{n+1}), \quad \ldots \quad (x_N, y_N)$$

to be the fixed ones. The task of finding good positions for the free cells is called placement. (The fixed cells correspond to cells that are already placed, or external pins on the IC.) There are $K$ wires that connect pairs of the cells. We will assign an orientation to each wire (even though wires are physically symmetric). Specifically, wire $k$ goes from cell $I(k)$ to cell $J(k)$. Here $I$ and $J$ are functions that map wire number (i.e., $k$) into the origination cell number (i.e., $I(k)$), and the destination cell number (i.e., $J(k)$), respectively. To describe the wire/cell topology and the functions $I$ and $J$, we'll use the node incidence matrix $A$ for the associated directed graph. The node incidence matrix $A \in \mathbb{R}^{K \times N}$ is defined as

$$A_{kj} = \begin{cases} 1 & \text{wire } k \text{ goes to cell } j, \text{ i.e. } j = J(k) \\ -1 & \text{wire } k \text{ goes from cell } j, \text{ i.e. } j = I(k) \\ 0 & \text{otherwise} \end{cases}$$

Note that the $k^{\text{th}}$ row of $A$ is associated with the $k^{\text{th}}$ wire, and the $j^{\text{th}}$ column of $A$ is associated with the $j^{\text{th}}$ cell. The goal in placing the free cells is to use the smallest amount of interconnect wire, assuming that the wires are run as straight lines between the cells. (In fact, the wires in an IC are not run on straight lines directly between the cells, but that's another story. Pretending that the wires do run on straight lines seems to give good placements.) One common method, called quadratic placement, is to place the free cells in order to minimize the the total square wire length, given by

$$J = \sum_{k=1}^{K} \left(x_{I(k)} - x_{J(k)}\right)^2 + \left(y_{I(k)} - y_{J(k)}\right)^2$$

**Part a** Explain how to find the positions of the free cells that minimize the total wire length.

**Part b** In this part you will determine the optimal quadratic placement for a specific set of cells and interconnect topology. The file at the following link defines an instance of the quadratic placement situation: `https://nicholasdwork.com/teaching/si2016/session2/hmwk4/qplace_data.py`.

Specifically, it defines the dimensions $n$, $N$, and $K$, and $N - n$ vectors `xfixed` and `yfixed`, which give the $x$- and $y$-coordinates of the fixed cells. The file also defines the node incidence matrix $A$, which is $K \times N$. Find the optimal locations of the free cells. Make a plot showing the locations of the fixed cells and the locations of the free cells for your optimal placement (along with the wires).

**Cancer Diagnosis and Prognosis Prediction** *Multiple Linear Regression* is the process of finding an *affine* model that best fits the data. That is, we seek a vector of coefficients $x$ and offset $g \in \mathbb{R}$ such that $y \approx Ax + g\,1$, where each row of $A$ is comprised of a unique datapoint. Then, given a new datapoint $a_{\text{new}}$, we can predict an output with the expression $y_{\text{prediction}} = a_{\text{new}}^T x + g$. This method applies to those cases where there exists a set of $N$ quantified features; in this case, $x \in \mathbb{R}^N$.

In this problem, we will use Multiple Linear Regression to attempt to predict whether a tumor is benign or malignant, and whether the cancer will recur after treatment or not based on data from the University of Wisconsin Hospitals. Download and unzip the file located at the following

url[1]: `https://nicholasdwork.com/teaching/si2016/session2/hmwk4/cancerPredictionData.zip`.

**Part a** Separate out the data into two subsets of approximately 80% and 20% proportions. The 80% proportion will be used to determine the linear coefficients; it will be called the *training* dataset. The 20% proportion will be used to evaluate how well the linear coefficients work. It will be called the *test* dataset.

**Part b** For diagnosis, the outcomes are specified as *benign* or *malignant*. Assign a value of 0 to benign, and a value of 1 to malignant. Then, our prediction will yield a real number that is hopefully between 0 and 1; we can consider this number as a probability describing our measure of belief in the diagnosis of the tumor.

**Part c** Determine the linear coefficients using the training dataset. Evaluate its performance using the test dataset.

**Part d** Repeat the process for the prognosis data; that is, create a linear model to predict whether or not the cancer will recur.

**Part e** Which are the most important features? Are there any features that are useless? Make some plots showing the true outcomes versus the predicted outcomes for several features.

**Part f** An extension of Linear Regression is *Polynomial Regression*; in polynomial regression, one includes the values of features raised to integer powers and products of features in the model. Try a polynomial regression for some features that you think may be better fit with a polynomial model; did it help?

**Pharmaco-kinetics** Pharmaco-kinetics is the study of how drugs move through the body. In this problem, we will model the body as a set of compartments: brain, stomach, intestines, heart, lungs, and other (which includes muscle tissue, the lymphatic system, and kidneys). Each compartment is represented by a node in a directed graph. The weights of the edges of the graph dictate what percentage of medicine travels from one part of the body to another in an time cycle. For this problem, we will consider every five minute interval a time cycle. For example, figure 1 shows a model where 10% of the concentration of the drug in the heart moves to the brain.

**Part a** Let $x$ be a vector representing the amount of medicine in any compartment of the graph. For the model presented in figure 1, find a matrix $M$ such that $x_{t+1} = M\,x_t$. Note: this is a very simple example of a type of equation called a *Linear Dynamical System*.

**Part b** Suppose that it was desired to have at least a quantity of 0.1 mg of medicine present in each compartment of the body for a 10 hour period. How much would someone have to swallow in order to meet that goal?

[1]Data source: `https://archive.ics.uci.edu/ml/datasets.html`

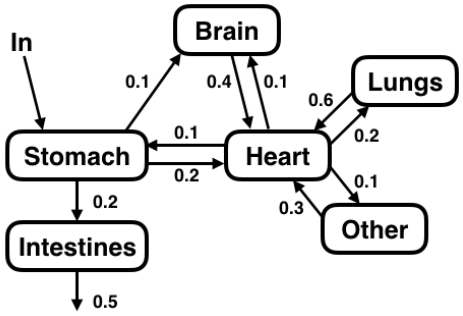


**Figure 1:** A compartmental model of the body. The input is oral consumption, which delivers medicine to the stomach.

**Part c** Instead of providing a patient with a pill, the medicine is delivered with an intravenous (IV) drip as shown in figure 2. The IV drip is computerized so that different amounts of the drug can be delivered in each time period. How much should the IV drip deliver in each cycle in order to attain a quantity of approximately 0.1 mg in each compartment after an hour of application?

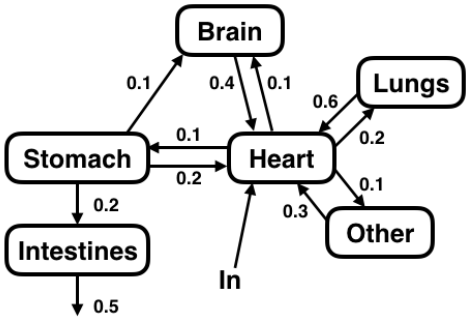


**Figure 2:** A compartmental model of the body. The input is an IV drip, which delivers medicine into the blood stream. We model this as an input into the heart.

**Part d** How soon after application can one attain a distribution of drug of approximately 0.1 mg in each compartment?

**Part e** Can you make it better? *Hint: you can.*

**Mixtures** Figure 3 shows a system of pipes and chambers. We assume that the fluids in each chamber are well mixed. The values $x_1, x_2$ represent two types of substances with 100% purity.

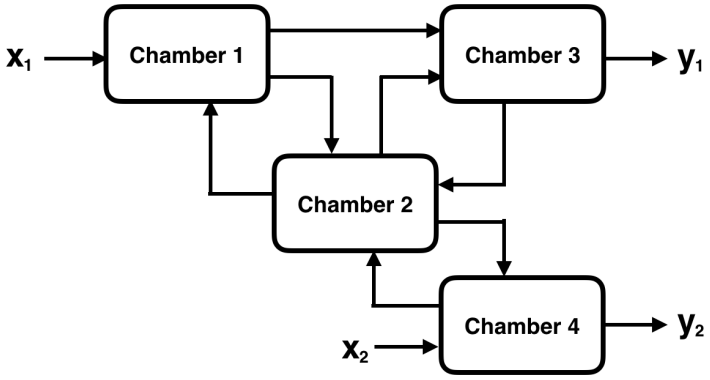


**Figure 3:** Fluids moving through chambers

We model this system as a directed graph (a set of nodes and edges where the edges only permit movement along one direction). The weights for each path are $Q_{12} = 0.6$, $Q_{13} = 0.3$, $Q_{21} = 0.1$, $Q_{23} = 0.2$, $Q_{32} = 0.3$, $Q_{42} = 0.2$. The outputs $y_1$ and $y_2$ are taps that can draw from chambers 3 and 4 as desired; they are both drawn with weights 0.1.

**Part a** Find a matrix $M$ such that $y^{(t+1)} = M\, y^{(t)}$.

**Part b** Find a sequence of inputs for $x_1$ such that the amount of $x_1$ in each chamber is 1 after 200 time steps (i.e. after 200 inputs).

**Part c** Find a sequence of inputs for $x_2$ such that the concentration of substance 1 in each chamber is $0.3, 0.4, 0.4, 0.2$ after 200 time steps.

**Structures** Figure 4 shows a structure. Make the lengths and angles appropriate for the problem to be interesting. The ground is assumed to be stable. The dark gray connectors to the ground are called *pin joints*; they exert a horizontal and a vertical force, but offer no rotational resistance. We consider forces (e.g. wind forces) as a force applied to a specific location. Note, since the ground is assumed stable, an earthquake is represented as a force applied down onto the structure.

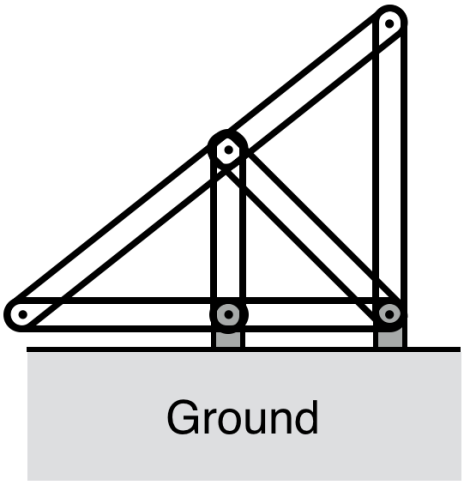


**Figure 4:** A structure

The force applied to each joint by a strut is a two element vector. In order for the structure to remain in equilibrium (unmoving), the sum of all forces at all joints must be 0. Note that if a strut applies a force at one end, it must apply an equal and opposite force at its other end (or else there would be acceleration).

**Part a** Consider several sets of forces applied at various nodes of the structure at differing angles. What are the maximum forces that each strut (each line segment in the structure) must be able to withstand given the sets of forces you've considered.

**Part b** Add an additional strut to the structure to make it more stable. How did you do?

**Image Inpainting** Sometimes, part of a picture gets corrupted. This may be because the picture is on an old medium (film, video tape) and the medium has errors in it. Or it may be done intentionally, perhaps by adding a logo or watermark onto an image. The goal of this project is to correct those corruptions. For this project, you will work with the image located at `https://nicholasdwork.com/teaching/si2016/session2/hmwk4/inpaintingData.zip`..

The files are as follows:

- `bac_small.jpg` - the uncorrupted image,
- `bac_small_corrupted` - the corrupted image,
- `bac_small_mask` - the mask showing which pixels are correct.

The mask shows you which pixels are correct, and which pixels are erroneous. Those pixels with a value of 255 in the mask are correct, and those with a value of 0 in the mask are incorrect.

To fix the corruptions, we will make use of a discrete approximation Laplacian function, defined below:

$$\begin{aligned}\mathcal{L}(x) &= \|D^{\text{horiz}}x\|_2^2 + \|D^{\text{vert}}x\|_2^2 \\ &= \sum_{i=1}^{M-1}\sum_{j=1}^{N-1}(X_{i+1,j}-X_{i,j})^2 + (X_{i,j+1}-X_{i,j})^2,\end{aligned}$$

where $X$ is the 2D array representing the image, $x$ is the column extended vector of the image, and the image is of size $M \times N$.

Let $y$ be a vector of just the unknown pixels. To perform image inpainting, we select those values of $y$ that minimize the Laplacian $\mathcal{L}$. That is, for a proper choice of $A$ and $b$, we select the $y$ that minimizes $\|A\,y - b\|_2$. (Note: there are many algorithms to perform image inpainting; this project describes one example of an image inpainting algorithm.)

**Part a** For a grayscale image, find $A$ and $b$.

**Part b** Perform image inpainting on a grayscale image. (To work with grayscale data, you can just work with one color channel of the data given to you.) Compare the result to the original.

**Part c** Perform image inpainting on a color image. Compare the result to the original.

## References

[1] Gennifer T Smith, Nicholas Dwork, Saara A Khan, Matthew Millet, Kiran Magar, Mehdi Javanmard, and Audrey K Ellerbee Bowden. Robust dipstick urinalysis using a low-cost, micro-volume slipping manifold and mobile phone platform. *Lab on a Chip*, 16(11):2069–2078, 2016.