Digital Signals

Assignment 3

Nicholas Dwork

1 Main Problems

Problem Recall that for a very long time, philosophers held a widespread agreement that the definition of knowledge was "a justified true belief." Create your own Gettier case that violates this definition. Can you create a better definition of knowledge where your Gettier case is no longer considered knowledge?

Problem Consider the optimization problem below

$$\text{minimize } \|Ax - b\|_2^2 + \gamma \|Dx\|_2^2$$

where A, b, γ, D are known, and x is the optimization variable. Find matrix $\tilde{A}$ and $\tilde{b}$ such that the following optimization problem is equivalent to the one above.

$$\text{minimize } \|\tilde{A}x - \tilde{b}\|_2$$

Provide an analytic expression for the optimal value of x in terms of $\tilde{A}$ and $\tilde{b}$.

Problem The video at the following link is of a real helicopter flying around; there are no special effects in the video.

`https://www.youtube.com/watch?v=qgvuQGY946g`

Why does it look like the blades aren't spinning?

Problem Find a subset of $\mathbb{R}^2$ that satisfies each of the following:

- Closed under pointwise addition but not under scalar multiplication
- Closed under scalar multiplication but not under pointwise addition

Problem Let $A \in \mathbb{R}^{M \times N}$. Show that the null space of A is a vector space.

2 Final Projects

FP 1 *Visualizing the Line Between Images* If you are given two points (or vectors) p_1, p_2, then you can parameterize the line segment between them as follows:

$$f(\alpha) = \alpha\, p_1 + (1 - \alpha)\, p_2,$$

where $\alpha \in [0, 1]$. What value does the function take on when $\alpha = 0$, $\alpha = 0.5$, $\alpha = 1$.

The set of all images is a cone (a cone is another type of set; it can be thought of as a vector space without any negative values; it is closed under non-negative linear combinations). We can use the above parameterization to make a line segment between images (gnarly, I know). In this project we are going to visualize this line. Download two images from the Internet that you like. Change these images so that they are the same size. Make an array of α values between 0 and 1 in steps of 0.1. Compute $f(\alpha)$ for each value of α where p_1 is one image and p_2 is the other. Save each image, and make an animated gif of your images (you can do so at `www.makeagif.com`).

Note that when you save the images, you're going to want to save them with a naming convention that stores the images in order. Suppose you choose to do this project with a `for` loop (which would be a good thing to do). Let `i` be the index of the for loop. You want the output image to be named `img_i.jpg`, where instead of `i` you replace it with the index corresponding to that image.

FP 2 *Making People Disappear* I have taken a video of imagery from a stationary platform. You can download individual frames from this video here:

`www.stanford.edu/~ndwork/si2016/hmwk3/peopleWalking.zip`

The goal of this project is to create a video of only the background (without any people in it). Note that there is no single frame like this; we're going to have to do some image processing to achieve our goal.

Part a Work completely in grayscale (that is, work with 2D images rather than 3D images; one way to convert from 3D to 2D is to only use one color channel). Think of a video as a 3D array, where each slice of the array is a single image. Make a new image where the value of each pixel equals the temporal mean of that pixel in all frames. That is, if the pixel we are computing is the $(i, j)^{\text{th}}$ pixel, the value of that pixel is the mean of the $(i, j)^{\text{th}}$ pixel in all frames.

Part b Perform the same procedure, but rather that using the temporal mean, use the temporal median.

Part c Repeat steps (a) and (b), but this time work with color imagery.

Part d What's the smallest number of frames you need for this to work with the temporal mean and temporal median? Which one works better? Why?

FP 3 *Showing Moving Objects* From the previous 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.

FP 4 *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 Δx. 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 Δx and Δy, 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 Δx and Δy. 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 3×3 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 γ.

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

FP 5 *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

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

where I is the noisy image, Ω is a neighborhood of y (e.g. a 5×5 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?

FP 6 *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:

`www.stanford.edu/~ndwork/si2016/hmwk3/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 3×3 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.

FP 7 *Image Fusion* For this project, we will use the data located at the following link:

`www.stanford.edu/~ndwork/si2016/hmwk3/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 image is to visualize a point on the line connecting the two images (see the "Visualizing the Line Between Images" project). 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:

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

where $f(F_R, F_G, F_B, R, G, B) = 0.3(F_R - R) + 0.6(F_G - G) + 0.1(F_B - B)$. (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.

FP 8 *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:

`www.stanford.edu/~ndwork/si2016/hmwk3/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.

FP 9 *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.

`www.stanford.edu/~ndwork/si2016/hmwk3/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 colorchart 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 3×3 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?

FP 10 *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^{th} row of A is associated with the k^{th} wire, and the j^{th} column of A is associated with the j^{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:

`www.stanford.edu/~ndwork/si2016/hmwk3/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).

FP 11 *Another Image Encryption* In class we learned one way to embed a code into an image. This project will show you another way to do that. As we previously discussed, each pixel of an image is a number that ranges from 0 to 255. This is because we store the data in a pixel as a byte, which is an ordered set of eight bits (a bit is a single digit that can store either 0 or 1). The conversion from binary to decimal is as follows:

$$x_7\, x_6\, x_5\, x_4\, x_3\, x_2\, x_1\, x_0 = 2^7 x_7 + 2^6 x_6 + \cdots + 2^0 x_0.$$

For example:

$$10110010 = 2^7 + 2^5 + 2^4 + 2^1 = 178.$$

The quantity x_0 is called the *Least Significant Bit* or LSB. Note that the gray of 178 is extremely close to the gray of 179. So if we added or subtracted 1 to the value it wouldn't change the image much.

To encode a message into an image, subtract 1 from all odd values. Then, add the message bit (1 or 0) to each pixel in sequence until the message is completed. Do not add anything to the remaining pixels.

To convert from ASCII to binary (and vice-versa) you can use the following website:

`http://www.unit-conversion.info/texttools/convert-text-to-binary/#data`

Part a Decrypt the message in the image located at

`www.stanford.edu/~ndwork/si2016/hmwk3/bacEmbedded.png`

Part b Encrypt a message into any image that you like. Verify that you can decrypt the message from the image.

Part c An audio stream is a one-dimensional array of values that range from 0 to 255 for the same reasons as imagery. Use this same process to encrypt a message into an audio stream. Verify that you can decrypt the message correctly.

FP 12 *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

`www.stanford.edu/~ndwork/si2016/hmwk3/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 an 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.

Part d An audio stream is a 1D array of data. "Inpainting" can be done with audio data as well. Create a short audio stream, and set small portions of it to 0. Then inpaint the stream; how does it sound compared 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.