Due Date: 17:00, Friday, January 16, 2015
TA: Yi Shen (yishen@sep.stanford.edu)

Lab 1 - Introduction to optimization and operators

Your name[1]

ABSTRACT

Here we will look at the general formulation of a linear optimization problem. We will define the model space, data space and the mapping matrix and focus on the properties of that matrix. Then, we will look at operators and how to analyze them. Finally, we will apply a few roughening operators to the topography map of the Bay area and interpret and analyze both the results and the operators.

LINEAR OPTIMIZATION

In any scientific experiment, we try to measure a certain property or attribute. If we are lucky, that attribute can be measured directly. Unfortunately, this is rarely the case. In general, we end up measuring another attribute that is not exactly what we want, but is related to it by physical equations. Going from the attribute that we want to the attribute that we measure is considered the forward problem. The goal of optimization is to go in the inverse direction, i.e. to compute what we want from what we measure using some physical equations.

Before we can invert our measurements, we first need to formulate the forward problem. In the framework of optimization, we define the model space to be the attributes that we want and the data space to be the attributes that we measure. Everything else, including the physical equations, acquisition parameters or any auxiliary parameters, will be included in the mapping matrix. Once we defined these components, we can write any forward process as follows:

$$\mathbf{d} = \mathbf{F}\mathbf{m}, \tag{1}$$

where the vector $\mathbf{d}$ represents the data space, the vector $\mathbf{m}$ represents the model space and the matrix $\mathbf{F}$ represents the mapping matrix. This formation is only valid if the data space is linearly related to the model space, i.e. if the matrix $\mathbf{F}$ is independent of the model vector $\mathbf{m}$. Notice that the size of the matrix $\mathbf{F}$ is determined by the size

[1]**e-mail:** youremail@stanford.edu

of the model and data space. For instance, if the size of the model space is $\mathrm{Nm} \times 1$ and the size of data space is $\mathrm{Nd} \times 1$, then the size of the mapping matrix is $\mathrm{Nd} \times \mathrm{Nm}$. Keep in mind that the model space and the data space are always vectors of size $\mathrm{N} \times 1$, regardless of the physical dimensionality.

The mapping matrix $\mathbf{F}$ is the matrix that converts a model into data. A big portion of the problem can be understood by only looking at the size of that matrix and which elements are non-zero, without actually looking at those non-zero values. For instance, a tall-thin matrix means that data space is larger than the model space, hence the mapping from the model to data includes a lot of spraying or scattering. On the other hand, a short-wide matrix means that data space is smaller than the model space, hence the mapping from the model to data includes a lot of summing or stacking. If the matrix is diagonal, then each data point is only a scaled version of one corresponding model point, which is called scaling. If the matrix has non-zero values in non-diagonal elements, then each data point is a linear combination of several model points, which is called filtering. If each column is the same as the previous column but shifted down by one element, the process is considered stationary. In the worst-case scenario, all the elements are non-zero, but this rarely happens.

OPERATORS

The straight forward way to apply a forward problem is to save the matrix $\mathbf{F}$ and then apply a matrix-vector multiply. However, in most of the problems that we will consider, the matrix is very sparse, i.e. most of its elements are zeros. Moreover, many processes are stationary where the same coefficients are repeated over and over in every column. So, instead of saving the whole matrix, we can implement a function or subroutine that loops through the elements of the model and the data and compute only the non-zero matrix elements on the fly. This efficient implementation is called an operator. In simple cases, each operator loop is similar to applying one row (or column) of the matrix. Notice that we never have to save or compute any unnecessary elements. Also, since operators do not use matrix-vector multiply, we do not have to convert the data and model to vectors.

However, since we cant see the matrix directly, it might be tricky to understand the behavior of the operator by just looking at the code, especially for complicated operators. Therefore, an easy way to understand an operator is to find its filter response (also called point-spread function or PSF). Computing the filter response is done by applying the operator on a model with a spike at one location only. The output corresponds to the column of the matrix $\mathbf{F}$ of that location. It is easier to see the properties of an operator from the filter response rather than from the code. If the operator is a scalar, the output should have values at one location only. If the operator is a filter, the output should have values at more than one location. If the operator is stationary, the filter response would be the same for all locations of the spike (the model edges might be an exception).

YOUR ASSIGNMENT

You will first answer questions regarding the general formulation, coding and cost of setting up a linear optimization problem. Then, you will work on the Bay area topography map. The `Makefile` already has rules to generate the dataset and its figure, as shown in Figure 1. You are also provided with three programs and modules that apply three different types of roughening operators. The programs and modules are complete and do not need to be modified. You only need to modify the `Makefile` to add rules that generate the required results and their figures for each question. Make sure to include the correct and complete set of prerequisites for each `Makefile` rule (including the executable programs). Also, do not forget to add the PDF figures to the default target and include the figures in this paper.

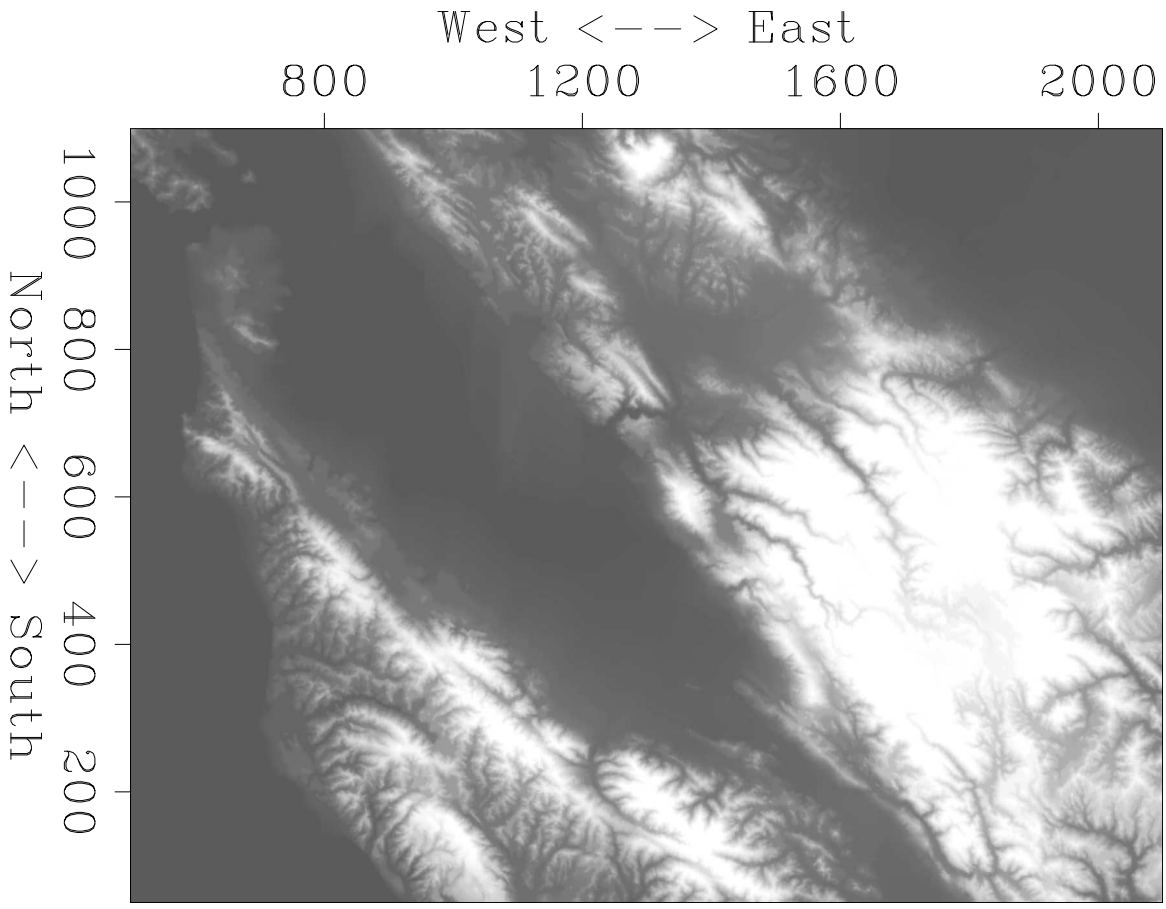

Figure 1: Bay area topography

Questions

General questions

1. Write a pseudo-code for a matrix-vector multiply using loops.

YOUR ANSWER.

2. How do we know if a matrix or operator is linear?

YOUR ANSWER.

3. Given a simple line fitting problem where you have the values $\mathbf{y} = \{y1, y2, y3\}$ and the coordinates $\mathbf{x} = \{x1, x2, x3\}$ and we want to fit a line with slope a and y-intercept b such as $\mathbf{y} = a\mathbf{x} + b$

   (a) What is the data space, model space and the mapping matrix (write out all the elements)?

YOUR ANSWER.

   (b) Given that we are fitting a line, is $\mathbf{y}$ linearly related to $\mathbf{x}$? Show your proof.

YOUR ANSWER.

4. In the relationship $\mathbf{y} = a\mathbf{x}^2$, is $\mathbf{y}$ linearly related to $\mathbf{x}$? If not, how can you reformulate it to make it so?

YOUR ANSWER.

5. Assume that the data is three dimensional $d(x_1, x_2, t)$ and model is two dimensional $m(y_1, y_2)$:

   (a) How do you convert the model and data to the form $\mathbf{d} = \mathbf{F}\mathbf{m}$?

YOUR ANSWER.

   (b) What would the sizes of $\mathbf{d}$, $\mathbf{F}$ and $\mathbf{m}$ be? Use a prefix N to indicate size, .e.g. Nx_1.

YOUR ANSWER.

6. Assume that the data is one dimensional $d(x_1)$ and model is also one dimensional $m(x_1)$ and the operation we want to apply is first-order backward finite difference (look it up if you do not already know it). Show the computational cost (size of memory required, number of summations and multiplications) of doing a matrix-vector multiply.

YOUR ANSWER.

7. Redo the same previous question but with operators instead of matrix-vector multiply.

YOUR ANSWER.

Coding

Study the Makefile and get a feel for how it works. The idea with these labs is to make the codes and results reproducible. Once you are done with the necessary changes you should be able to type `make burn` and then `make` to rebuild all results. This is how you will be graded for these portions of the assignments.

1. Add rules to the `Makefile` that applies each roughening operator (`Rough1`, `Rough2` and `Rough3`) to the topography data. Then, add rules to generate a figure for each result and include them in this paper. Comment on the effect of each operator.

YOUR ANSWER.

2. Add rules to the `Makefile` that generates a small model with a spike at some location (hint: check out the command `Spike`) and then apply each roughening operator to compute its filter response. Then, add rules to generate a figure for each result and include them in this paper. Do the filter responses agree with your previous observations?

YOUR ANSWER.

Be sure to add your figures to the line that begins `default:`. Now you should be able to type `make burn` and then `make` to recreate everything.

Extra Credit

1. What are these three roughening operators?

YOUR ANSWER.

2. Other than just looking at the filter response, how else would you further analyze it (just mention the method)?

YOUR ANSWER.

3. Complete the comments in the source code of the first operator's program and module as well as the `Makefile`

EXTRA LAB (WIZARDS ONLY)

In chapter 1 of GIEE you first learn about filtering. Next you learn about linear interpolation. This is an exercise where you try to put the two together. I give you lots of hints, then we talk about it. This is the first year this assignment is given, so, I have little idea how difficult you will find it. PLEASE LIMIT YOURSELF TO ONE HOUR. Do not expect to finish. Please turn it in on time. We will discuss it in subsequent class periods until we all know the answer.

Background

We want to have filters that change their impulse response with time. The obvious code seems to require big memory with size being this product: data length times maximum filter lag. This isn't bad in one physical dimension, such as time, but is oppressive in higher spatial dimensions, such as in (t, x), or (t, x, z). But here, as warm up, we try to put it together in time alone.

Fortunately, applications calling for time (and space) variable filters generally want the filter to change slowly with time. Thus we envision parameterizing filters by a coarse mesh (widely spaced time points). From this coarse mesh, linear interpolation finds the effective filter at each point in time.

Exercise

Write pseudo-code, or code in any language you choose, to define time-variable convolution with a filter linearly interpolated from a coarse mesh. Ignore the adjoint. You

are invited to discuss this problem with others before class. LIMIT YOUR TIME ON THIS PROBLEM TO ONE HOUR.

HINTS

The equation for filtering data $x(t)$ with filter $b(\tau)$ is

$$y(t) \quad = \quad \int b(\tau)\, x(t-\tau) \tag{2}$$

Practitioners will love it if we give them a filter changing with time:

$$y(t) \quad = \quad \int b(\tau, t)\, x(t-\tau) \tag{3}$$

The convolution part looks like

```
do t {
do tau {
        if operator itself
                y(t) += b(tau,t) × x(t-tau)
        if adjoint
                b(tau,t) += y(t) × x(t-tau)
        }}
```

The declarations at the beginning of the code can make it work like like $\mathbf{y} = \mathbf{Bx}$ or like $\mathbf{y} = \mathbf{Xb}$.

How do we think about $b(\tau, t)$? It's evidently an array, but not really. It's more like an outer product. Using matrices, we might write it as

$$b(\tau,t) \quad = \quad \begin{bmatrix} \vdots \\ \mathbf{b}_i \\ \vdots \end{bmatrix} \begin{bmatrix} 1.0 & .9 & .8 \cdots & 0.0 \end{bmatrix} + \begin{bmatrix} \vdots \\ \mathbf{b}_{i+1} \\ \vdots \end{bmatrix} \begin{bmatrix} .0 & .1 & .2 & \cdots & 1.0 \end{bmatrix} \tag{4}$$

where the idea is that $\mathbf{b}_i$ represents the filter at the previous mesh level and $\mathbf{b}_{i+1}$ at the next level. Filter lag is on the vertical axis while time t is on the horizontal.

DONE

When you are all finished modifying the source files, `Makefile` and latex file, make sure that will paper will compile corretly from a cleaned directory using the default targets only. In other words, the following sequence of commands should produce

your a PDF version of your paper without any problems: `make burn`; `scons -c`; `make`; `scons`.

Once you make sure everything is working properly, clean up your directory by typing `make burn`; `scons -c`.

Reference

GIEE chapter 8.1

`http://sep.stanford.edu/sep/jon/wjt/chapter8-1.pdf`