Due Date: 17:00, Monday, January 26, 2015
TA: Yi Shen (yishen@sep.stanford.edu)

Lab 2 - Operators, adjoints and dot product test

Your name[1]

ABSTRACT

We will look at three types of operators: finite difference operators, recursive operators and summation transform operators. We will examine the implementation of each type and run the operators on a model to create some datasets. You will code the adjoint of each operator and then test the accuracy of the adjoint using the dot product test. Finally, you will apply the adjoints to the datasets in an attempt to get the models back.

FORWARD OPERATORS

The previous lab highlighted how forward processes can be done efficiently with operators by implementing a function or subroutine that loops through the elements of the model and the data and applies only the non-zero matrix elements. In this lab we will look at ways of achieving this efficient implementation by examining three common types of operators: finite difference operators, recursive operators and summation transform operators.

Finite difference operators

Finite difference operators are very common since they give an easy way to apply derivatives on a sampled signal. The mapping matrix is square (except for the boundaries) which results in a model space and data space of equal sizes. Moreover, finite difference operators are very local: each point in the output depends on the corresponding point in the input and it's close neighbors. Therefore, instead of looping over both model space and data space, we can loop over one of them and shift the index to get the necessary neighboring values.

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

Recursive operators

Some processes have triangular mapping matrices where the coefficients of one row is the scalar multiplication of the previous row in addition to a diagonal elemenent. These processes can by implemented using recursive operators. In a recursive operator, the output of the previous loop is used in the current loop. This method can be efficient because we reuse previous results instead of computing them again. Some examples of recursive operators are leaky integration and lowcut filters. The downside of recursive operators is that the loop has to be executed sequentially and code parallelization becomes difficult.

Summation transform operators

Summation transform operator relate the model space to the data space by summation/spraying surfaces. Each point in the model space is computed by summing all the data points along a certain surface or vice-versa. These summation/spraying surfaces are defined by a mapping equation which relates the model space axes to the data space axes. To implement summation transform operators, we loop over all but one of the variables of the mapping equation. Instead of looping over the last variable, we compute its value using the mapping equation and then find the index that corresponds to that value. You have the choice of which mapping variable to compute. This variable could be either a model space axis or a data space axis.

YOUR ASSIGNMENT

You will work with three operators on three different datasets. The forward operators are provided for you and the `Makefile` already has rules to generate the datasets and their figures. You are also provided with the dot product test. Each operator and dot product test is composed of a program and a module. The programs are complete and do not need to be modified. However, all the modules are missing parts that you need to complete. You will first apply the forward on a dataset. Then, you will code the adjoint operator and run the dot product test (after your complete its module). Finally, you will apply the adjoint operator on the output of the forward operator in an attempt to get the original models back. Modify the `Makefile` to add rules that generate the required tests, results and 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.

Finite difference operators

We will use the second order Laplacian operator as an example of finite difference operators. Use the brick dataset, shown in Figure 1, to test this operator.

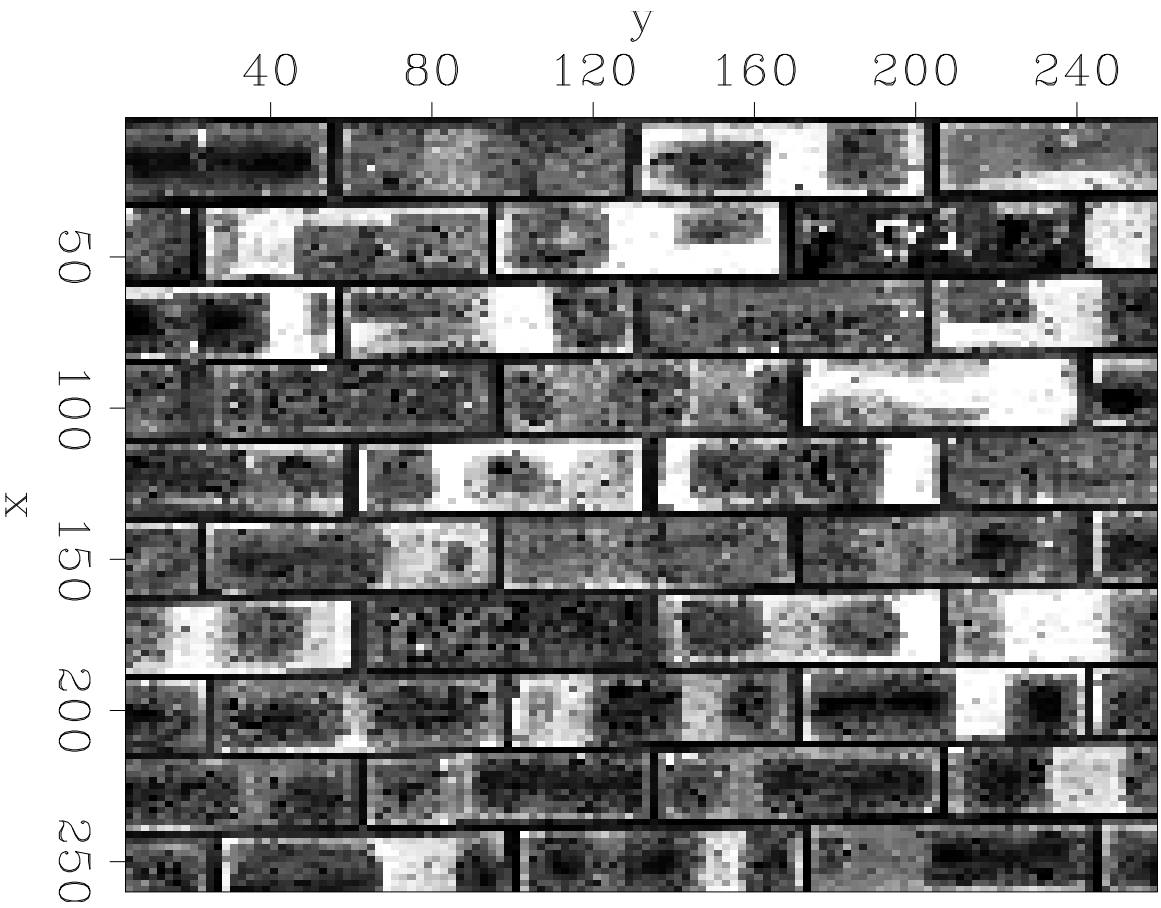

Figure 1: Bricks

Questions

1. Apply the forward Laplacian operator on the brick dataset and plot the results. Describe the effect of the forward operator.

YOUR ANSWER.

2. Complete the dot product test module and then code the adjoint of the Laplacian operator. Run the dot product test for the Laplacian operator. What are your test results?

YOUR ANSWER.

3. Do you need to get an exact match to consider the adjoint correct? Why?

YOUR ANSWER.

4. Apply the adjoint on the output of your forward operator and plot the results. Describe the effect of the adjoint operator. Is this adjoint a good approximation to the inverse?

YOUR ANSWER.

Recursive operators

We will use the lowcut operator as an example of recursive operators. Use the bay dataset, shown in Figure 2, to test this operator.

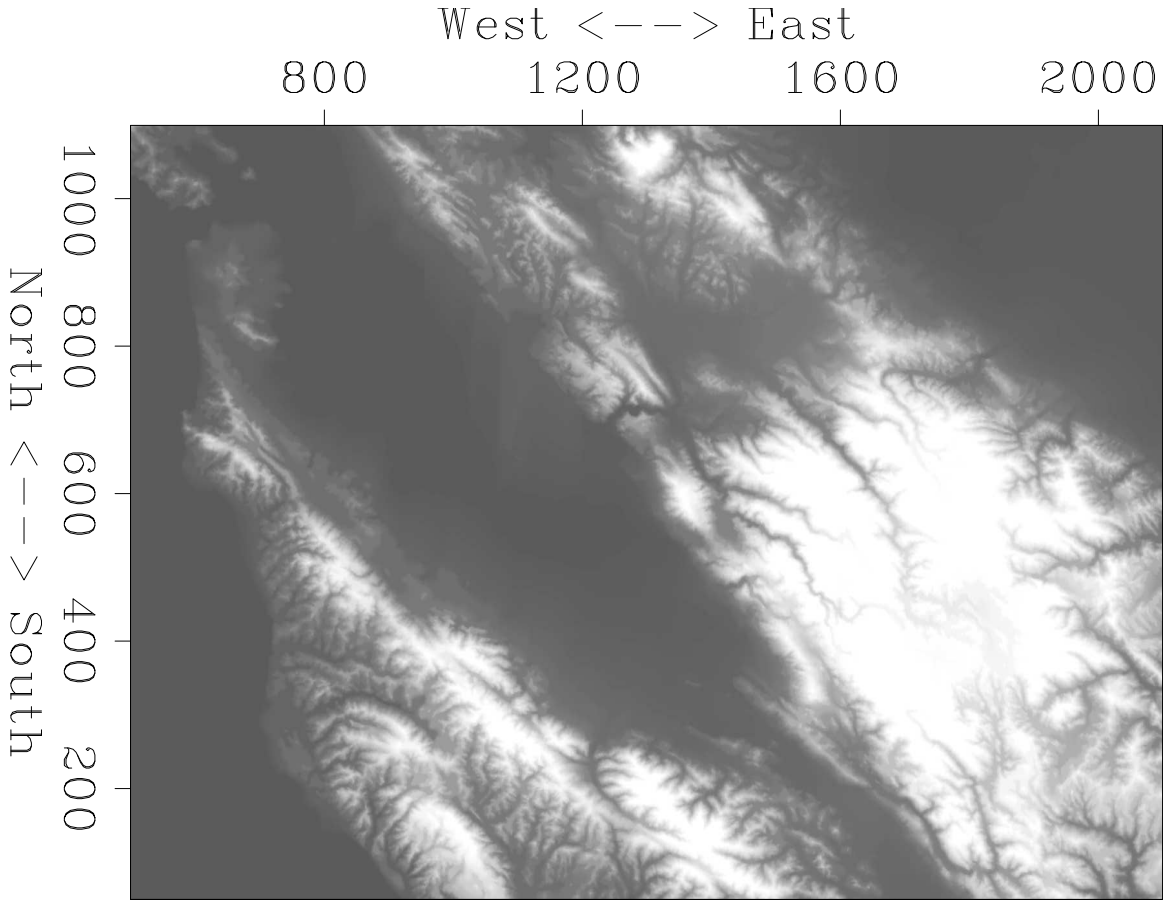

Figure 2: Bay area topography

Questions

5. Apply the forward lowcut operator on the bay dataset and plot the results. Describe the effect of the forward operator.

YOUR ANSWER.

6. Code the adjoint of the lowcut operator and run the dot product test. What are your test results?

YOUR ANSWER.

7. Apply the adjoint on the output of your forward operator and plot the results. Describe the effect of the adjoint operator. Is this adjoint a good approximation to the inverse? What effect does the value of ρ have on the output?

YOUR ANSWER.

Summation transform operators

We will use the linear Radon operator as an example of summation transform operators. In this operator, the model space axes are intercept and slope whereas the data space axes are time and distance. Therefore, going from model space to data space is done by spray each point into a line where the adjoint sums along a line in the data space to compute a point in the model space. The mapping equation can be written as:

$$t = px + z, \tag{1}$$

where t is time, p is slope, x is distance and z is intercept. The operator loops over p, x, z and computes t. Use the spike dataset, shown in Figure 3, to test this operator.

Questions

8. Apply the forward linear Radon operator on the spike dataset (use the parameters in the file `Par/lradon.p`) and plot the results. Describe the effect of the forward operator.

YOUR ANSWER.

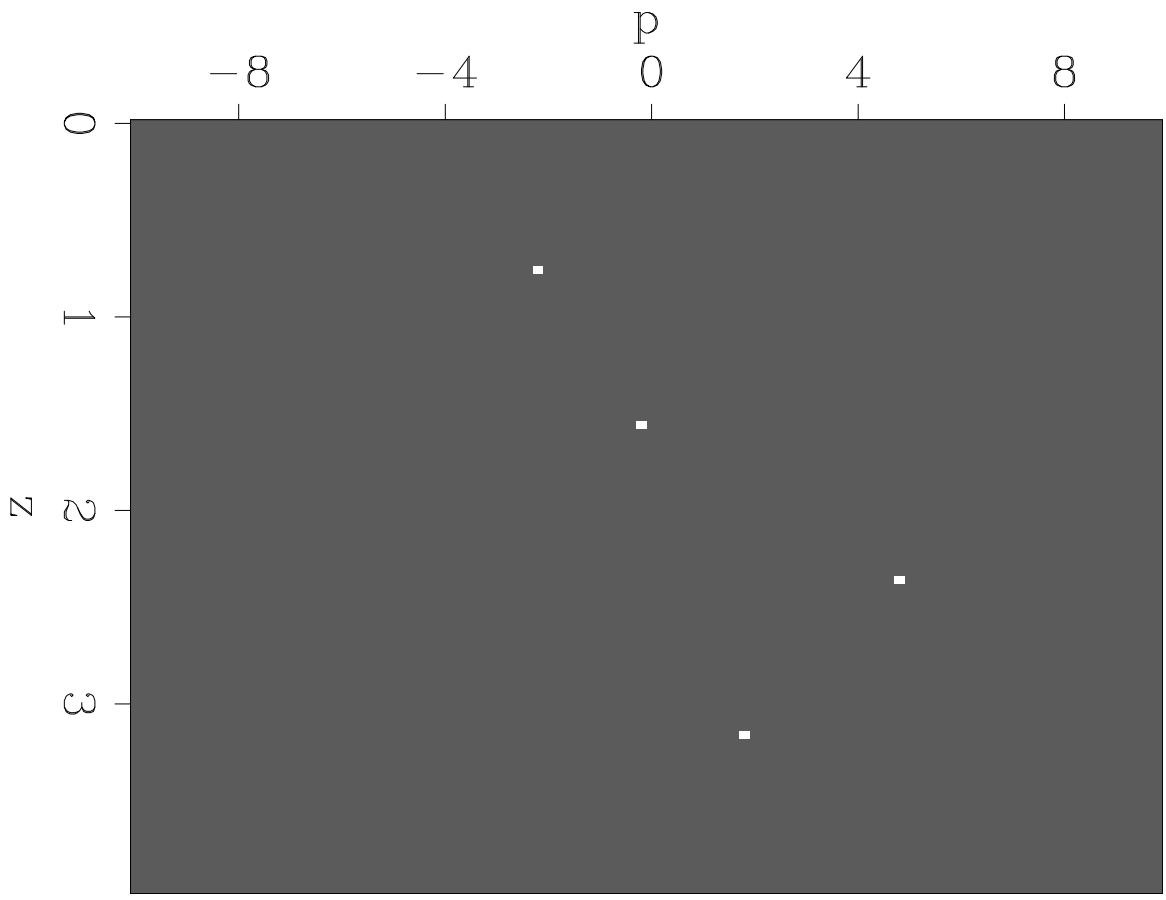

Figure 3: Linear-Radon model

9. Code the adjoint of the linear Radon operator and run the dot product test. What are your test results?

YOUR ANSWER.

10. Apply the adjoint on the output of your forward operator and plot the results. Describe the effect of the adjoint operator. Is this adjoint a good approximation to the inverse?

YOUR ANSWER.

Extra Credit

1. In the adjoint results of the linear Radon operator, why do we get fan like shapes? What do the slopes represent?

YOUR ANSWER.

2. Describe an experiment where the adjoint linear radon operator becomes very close to the inverse.

YOUR ANSWER.

3. The module of each operator has a function and a subroutine and the function calls the subroutine. What is the point of this setup?

YOUR ANSWER.

EXTRA LAB (WIZARDS ONLY)

In Lab1, you've been asked to write the pseudo-code to define time-variable convolution with a filter linearly interpolated from a coarse mesh. In this lab, write the pseudo-code of its ajoint. PLEASE LIMIT YOURSELF TO ONE HOUR.

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`.