A canonical program library

Jon F. Claerbout

ABSTRACT

A *canonical computer program* is defined to be one based on a linear operator, that also meets certain programming standards.

INTRODUCTION

Report and journal articles may provide a satisfactory account of research, but they rarely enable anyone else to replicate results quickly. At SEP we often have difficulty reconstructing the work of bygone graduates. It would help if programs were left in usable form.

Programming standards: pro

Programs become much more usable if they meet some sort of programming standards. Programming standards make your programs more accessible to other people. Writing your programs according to accepted standards prepares you to take advantage of other work done to the same standards. Also, it attracts contributions by others to your programs. Programming standards give a program longevity. When computers and operating systems change, as they always do, libraries that meet consistent standards are much more likely to be converted than isolated programs of unknown standards.

Programming standards: con

Demanding programming standards would be a good idea if it were clear what the standards should be. But most of our research is nonroutine. Enforced programming standards could interfere with the progress of innovative research.

PROGRAMMING STANDARDS

A sensible philosophy is to watch for standards and use them when they exist. Examples of usable standards are:

- The fortran subroutine
- GKS (the Graphical Kernel System)
- SEG tape format standards
- Disco
- seplib
- sy system

Seplib (cubelib)

More than 50% of the programs written at SEP in the last year have been written to meet the *seplib* standard. We simultaneously use the name *seplib* to refer to a package of programs, utility subroutines and programming aids written to meet these standards. The seplib standard is misnamed. It should have been called *cubelib* because each data set is considered to be a cubic lattice with the number of points on each axis given by 'n1', 'n2', and 'n3'. (Axis 1 is the Fortran column axis). The first element of the cubic lattice has a location in physical space denoted by 'o1', 'o2', and 'o3'. Points on the cubic lattice are separated by physical intervals 'd1', 'd2', and 'd3'. Axes of the lattice are labeled 'label1', 'label2', and 'label3'. These names are *external* to the programs. They are names in a data base (called a history file) that the programs read when they need information about the lattice. Many programs use other names internally, such as 'nt' for 'n1'.

A model seplib process reads an input history file, appends processing information and default parameter overrides, writes an updated output history file and then processes the data cube. This mild programming discipline permits the seplib user to string together several processes end-to-end with a useful feature "pipes" of the UNIX (trademark AT&T) operating system. For example a succession of linear operators can be applied and the user does not need to allocate storage for intermediate results.

Sy system

The principal dissatisfaction with the *seplib* standard is that it does not adequately handle irregularly sampled data sets. For irregularly sampled data sets there is another system in use at SEP called the "sy system," the name denoting SEGY format. This system was invented by Einar Kjartansson and further developed by Shuki Ronen and Stew Levin.

Canon program library

I am gathering programs to meet a new standard applicable only to programs that implement linear transformations. I estimate that about half of the programs written for seismic data analysis do linear transformations, so I hope the library will grow substantially.

I am calling the program library the "canon library". The word "canon" means "a law or a body of laws of a church". I have chosen to call the library the "canon" library because linear operators do have a body of laws ruling their behavior. To qualify for the canon library a program must pass tests based on properties of linear operators. (Linear operators sometimes have "canonical" forms but such forms have nothing to do with the canon library). Canon library programs meet 4 global standards and 3 more local standards.

GLOBAL CANON LIBRARY STANDARDS

Because of the large variations in operating systems, programming standards that can claim to be global are about limited to subroutines with fortran-style argument passing. The canon global standards are:

1. Distributive law $A\,(x+y\,)=Ax+By$
2. Conjugate transpose routine must be provided.
3. Pseudoinverse option must be provided (if only the transpose).
4. Each routine must pass the dot product test, i.e. $(A\,'y\,)'x=y\,'(Ax\,)$

Why insist on the transpose?

Geophysical modeling is often the mathematical transpose to the corresponding geophysical data processing. I illustrated in SEP 42 that for many of the linear operators in common geophysical use, the transpose is so closely related to the operator itself that they may as well both reside in the same body of code.

Fourier transformation is a linear operator and signals and noises are often characterized in both the time domain and the Fourier domain. More generally, for each matrix or linear operator A, and transformation $y = Ax$ there is a space of inputs (x) and a space of outputs (y). Meaningful signals and noises are often identified in the x-space or the y-space of any of the A operators that we use in exploration geophysics. Optimization procedures that attempt to estimate missing data, enhance signals, or cope with noises typically require that the user provide two programs, one to apply the linear operator, and one to apply its transpose. So a program library of linear operators that includes the transposes is automatically of value to people doing optimization work. The optimizers themselves need not learn the details of the linear operators they are using.

Fourier transform example

Many Fourier transform programs *pad* the data to a length of the form 2^{integer} *before* the Fourier operation. Notice that the operation of padding can be regarded as multiplying by a rectangular matrix containing an identity matrix I on top with a zero matrix beneath. Padding followed by Fourier transformation can be expressed in matrix algebra by

$$\mathbf{Prog} \quad = \quad \mathbf{F} \begin{pmatrix} \mathbf{I} \\ 0 \end{pmatrix} \tag{1}$$

The conjugate program is now

$$\mathbf{Prog}^* \quad = \quad \begin{pmatrix} \mathbf{I}^* & 0 \end{pmatrix} \mathbf{F}^* \tag{2}$$

So we see that the conjugate program *truncates* the data *after* the Fourier operation. Likewise, if a program pads real data values to complex values with zero imaginary parts, then the conjugate program truncates the complex values back to reals.

So we see that the mathematical conjugate suggests a program that does some of the friendly things you might want a program to do anyway. (Plot programs abhor complex numbers and data sets typically have a number of data points that is a multiple of 10, not a power of 2).

Forward vs inverse

There is often confusion about what constitutes the *forward* Fourier transformation and what constitutes *inverse* transformation. That confusion is even worse in seismic data processing. On the one hand it is natural to say that *modeling* should be called the "forward" transformation and on the other hand, it is natural that geophysicists have

given names to the horses in their stable of processing programs. Modeling names and processing names point to processes that are often conjugates or inverses of each other. So which one should be called the "forward" operator? Philosophical questions like this have been known to divide churches, start civil wars, and separate blood relatives. The canon program library standards wisely say nothing about the definition of any linear operator, only that after you have defined one, you must define its transpose.

The dot product test

The associative property of linear algebra says that

$$y'(Ax) \quad = \quad (y'A)x \tag{3a}$$

which is the same as

$$y'(Ax) \quad = \quad (A'y)'x \tag{3b}$$

Take the vectors x and y to contain random numbers. In this identity, the part $(A'\ y)$ uses the transpose operator and the part (Ax) uses the operator itself. So it is a test of the validity of a transpose program that (3b) is satisfied. Experience with a wide variety of operators shows that (3) is normally satisfied to near the computing precision. Indeed when we switched from the VAX to the Convex, I was surprised to find that equation (3) was often satisfied down to *the least significant bit.* I don't doubt that larger rounding errors could occur, but so far, every time I have encountered a relative discrepancy of 10^{-5} or more, I was later able to uncover a conceptual error or a programming error.

Don't be alarmed if the operator you have defined has truncation errors. Such errors in the definition of the original operator should be identically matched by truncation errors in the conjugate. If your code passes the dot product test, then you should be able to pass back and forth between data space and model space as many times as you like without fear of growth of truncation errors.

Equation (3) also has a more physical interpretation. Say that x is a vector in model space and y is a vector in data space. Let x_1 be a random model vector and y_2 be a random data vector (from a different model). From the model x_1 we can find data $y_1=Ax_1$. Likewise, from the data y_2 we can find a model $x_2=A'y_2$. Inserting in (3) yields

$$x'_2\, x_1 \quad = \quad y'_2\, y_1 \tag{4}$$

So the inner product test relates the two cases.

LOCAL CANON LIBRARY STANDARDS

The rest of this paper defines local standards that are peculiar to Stanford at this time. Never-the-less, it is included because it suggests goals that may be met at other sites by other means.

Main programs find inputs and parameters, and should compute defaults for everything that can be defaulted. Main programs are highly dependent on the local operating system. Scientifically, main programs are less than inspiring. But experience shows that a good library of main programs is like a good "tool box." It enhances productivity. Outsiders are impressed by the speed and ease that complicated structures get built.

The local standard demands that each program

5. be usable as a main or as a subroutine.
6. demonstrate its impulse response with Wiggle or Graph.
7. demonstrate its pseudoinverse with Wiggle or Graph.

"Wiggle" and "Graph" are plot programs on the local graphics terminals.

The local standards are loosely attached to the directory (/usr/src/sepsrc/canon) containing the local shared canon program library. Besides the sources of various canonical programs, this directory contains various supports. There is a text file explaining how to install programs (ReadMe). There is a command script (dot.sh) for doing the dot product tests. There is a command table (fakefile) for compiling and performing impulse response tests.

Conj and inv

It is required that each canonical program declare the parameters:

conj=0	default (i.e. the operator itself)
conj=1	apply the conjugate transpose operator.
inv=0	default (i.e. the operator itself)
inv=1	apply a pseudoinverse operator (maybe only the conjugate).

When both 'conj=1' and 'inv=1' the program should supply the conjugate of the inverse (which is the same as the inverse of the conjugate). Unitary and near-unitary operators are prevalent in seismic data analysis. Mathematically, the inverse of a unitary operator is the same as its conjugate transpose. So, for a unitary operator, specifying both 'conj=1' and 'inv=1', is equivalent to specifying both equal to zero.

Many linear operators do not have an inverse, but they always have a pseudoinverse because you can define the *pseudo*inverse any way you like. The transpose matrix

is always a possible pseudoinverse. A better pseudoinverse is to divide each row of the transpose by a row sum of $(A^T A)$, i.e. scale the transpose so that the zero frequency response is exactly inverted. This is not the place for a more detailed account of pseudoinverses.

Canonical programs generally test on the condition whether $conj \equiv inv$. This test determines the space of the inputs and outputs. For example, the Normal Moveout program looks like

```
real tt(n1), zz(n1)
if( conj ≡ inv )
      call read('t-space', n1, tt)
else
      call read('z-space', n1, zz)

call nmo( conj, inv, n1, tt, zz)

if( conj ≡ inv )
      call write('z-space', n1, zz)
else
      call write('t-space', n1, tt)
```

You will notice in the above example that a single subroutine provides for the linear operator, its transpose, and its pseudoinverse. There is not a separate subroutine for each task. This may seem surprising, but it describes every program put in the library so far. The reason seems to be that it is usually easy to code the transpose operator along with the operator itself. And the pseudoinverse usually follows just after the transpose. Also, there is the issue of program maintenance. When improving an operator, it is easiest to maintain consistency if the operator and its transpose share as much code as possible, which means putting them in the same subroutine.

Tetch

The following local command line says that a history file named "in.h" (which contains seplib parameters and a pointer to a data lattice) will be slant stacked over 75 Snell parameters (p-values) and the output history file will be named "out.h".

```
<in.h Slantstack n2=75 > out.h
```

The following local command line does slant stack and follows up by inverse slant stack.

```
<in.h  Slantstack  n2=75 | Slantstack inv=1 > pseudoin.h
```

The second invocation of Slantstack required us to develop a new procedure for extracting information from history files. The problem that needed a new solution was to

enable the second invocation of Slantstack to find the number of traces input to the first invocation of Slantstack. Thus it had to search the history file for the last value of n2 before the first invocation of Slantstack. The local name for the new subroutine is called "tetch".

Nonsquare operators and SAW (Sep Auto Writer)

Many matrices are not square. Programs in the canon library will frequently have occasion to handle inputs of different dimensionality than outputs. The SEP autowriter provides fetches with defaults and error checks. When the linear operator is "rectangular" then the dimensions of inputs and outputs get interchanged if $conj \neq inv$. This resembles the NMO example earlier. Take an example where 'nx' denotes the number of traces and 'nvel' denotes the number of points in velocity space. In the syntax of SAW we have

```
if conj≡inv
        from history:   integer  n2:nx
        from tetch:     integer  n2:nvel
        to history:     integer  n2=nvel
else
        from history:   integer  n2:nvel
        from tetch:     integer  n2:nx
        to history:     integer  n2=nx
```

To escape such verbosity, *saw* can produce the above 8 lines from the single line

```
from conj: integer n2:nx:nvel
```

as usual in *saw*, defaults are possible, i.e.

```
from conj: integer n2 : nx=nvel+1 : nvel=nx
```

When the computation of defaults is more complicated, the verbose form may be inescapable, such as where n1=function(n1,n2) and n2=function(n1,n2).

Defaults

Defaults can cause a perfectly good program to fail dot product tests. It is the tester's responsibility to be sure that all process parameters are the same on the conj=1 invocation as the conj=0 invocation, and this normally means overriding the defaults. Although defaults could be forced to be conjugate, there are many reasons for allowing defaults to be nonconjugate. For example, notice that the matrix operator (I,0) that truncates is transpose to the operator that pads. It is natural for a Fourier transform program to *default* to pad to a power of two, whether or not the conjugate is being invoked. But in normal use, or when taking a dot product test, the second invocation

should truncate from a power of two back to the original data size.

PRESENT STATUS

At the present time, the canon program library has little to covet. There are only six installed programs.

Interp2	JFC	linear interpolate 2-axis with exact inverse
NMO	JFC	Normal moveout
Phase	Al-Yahya	Gazdag
Radial	JFC	radial traces
Stolt	VanTrier	Stolt
Unmo	JFC	Unitary NMO

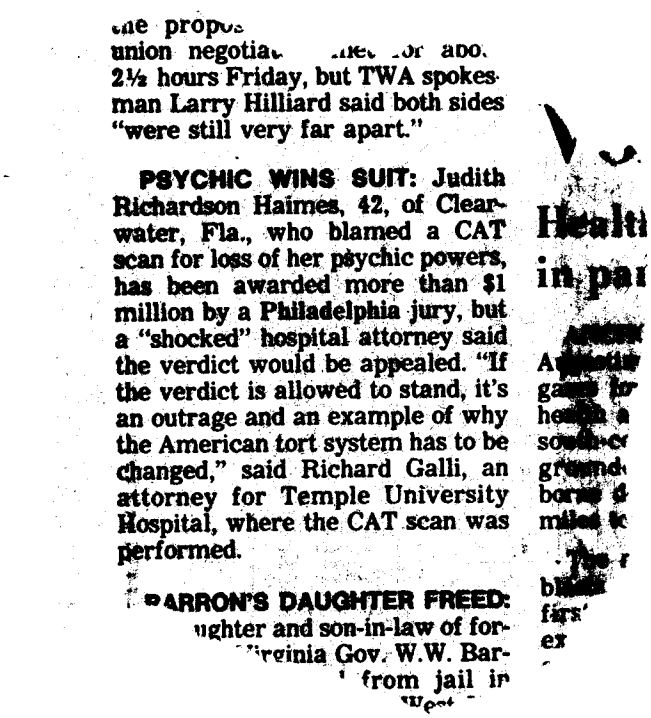
ACKNOWLEDGEMENT

I conceived and implemented seplib based on Rob Clayton's "getpar". Stew Levin completed the project and brought the code to a high standard. The piping aspect never would have been successful without him. Kamal Al-Yahya wrote "tetch."

REFERENCES

Claerbout, J.F., 1985, What is the transpose?, SEP-42, pp 113-128

The Dangers of Tomography



...ne propo...
union negotia... ...ier ...or abo...
2½ hours Friday, but TWA spokesman Larry Hilliard said both sides "were still very far apart."

PSYCHIC WINS SUIT: Judith Richardson Haimes, 42, of Clearwater, Fla., who blamed a CAT scan for loss of her psychic powers, has been awarded more than $1 million by a Philadelphia jury, but a "shocked" hospital attorney said the verdict would be appealed. "If the verdict is allowed to stand, it's an outrage and an example of why the American tort system has to be changed," said Richard Galli, an attorney for Temple University Hospital, where the CAT scan was performed.

...ARRON'S DAUGHTER FREED: ...ughter and son-in-law of for... ...rginia Gov. W.W. Bar... ...from jail in ...West...

Healt...
in pa...