SEPVector: a C++ inversion library

Eileen Martin, Robert G. Clapp, Huy Le, Chris Leader, and Dave Nichols

ABSTRACT

SEPVector is a library of C++ classes, methods, and simple interfaces for solving geophysical inverse problems. From the beginning this library was designed to allow users with relatively little coding expertise to work on heterogeneous computer architectures, a feature that is becoming increasingly critical to modern library development. Verification of the code features is done through a thorough set of unit tests. Although it is written in C++, Fortran users can easily learn to use the SEPVector interface through a series of simple examples included in the package.

INTRODUCTION

As hydrocarbons become more difficult to find, ever more sophisticated imaging techniques are needed to produce accurate images of the subsurface. One approach to improved imaging is the use of algorithms, such as Reverse Time Migration (RTM) (Baysal et al., 1983), that more accurately describe wave behavior. As important, is the recognition that imaging is fundamentally an inversion problem. Inversion has been used extensively to produce velocity, or more generally earth property, models for years using ray based techniques (Stork, 1992; Clapp, 2001) and more recently with the growing use of waveform inversion (Woodward, 1990; Sirgue and Pratt, 2004). There is also growing use of inversion in place of more conventional migration techniques (Clapp, 2005; Valenciano, 2008; Tang, 2011).

The growing size of seismic datasets, more accurate imaging algorithms, and the expanding use of inversion all lead to significantly increased compute requirements. At the same time the last fifteen years as seen significant changes in computer architecture including the (re)introduction of vector units (up to 16 in length), multiple cores (up to 12 per socket), and the (re)introduction of co- processors such as General Purpose Graphical Processing Units (GPGPU) and Xeon Phi.

The last twenty years have seen several attempts at SEP and other institutions in building an object oriented inversion library that separates the math (optimization) from the physics. Nichols et al. (1993) was an attempt at using C++ and Schwab (1998) in Java but both failed to catch on due to the predominant use of Fortran in SEP. Fomel and Claerbout (1996) built a heavily used library on top of Fortran90, but it is difficult to use in complex inversion problems due to Fortrans object oriented limitations and its single thread nature. Clapp (2004) built a python based

library that proved successful for cluster based applications, but was not practical on smaller problems. Attempts to use outside frameworks (Gockenbach and Symes, 1996; A. D. Padula and Symes, 2009) have not proven to catch on at SEP.

In this paper we introduce a new inversion library, SEP vector. The library is written in C++, uses modern programming techniques, such as unit tests for reliability. It is designed to take advantage of modern architecture, such as being vector and multicore aware on conventional CPUs. In this paper we begin by describing the core classes of the library, we then discuss our testing environment, and present some simple examples, before concluding with the future directions we plan to take the library.

DATA STRUCTURES AND INHERITENCE

One of the largest scale features of the SEPVector library is that all of the data structures and solvers are classes contained in the SEP namespace. The namespace will make it easier for users to combine this library with other libraries (e.g. a visualization library), especially if the other library has any classes of the same name. Each instantiation of the classes is an object with some associated data and methods that it can call. Some classes need similar methods, so in our code similar classes inherit methods and a description of their data from more general abstract classes.

Most codes have some amount of error checking along the lines of asking whether the dimension of the input match those of the operator, but what happens when a vector of the wrong units just happens to have the right length? This can especially be an issue when chaining together many operators or concatenating operators. Our solution, inspired by (A. D. Padula and Symes, 2009), associates each vector with a space, and each operator with a domain space and a range space so that it is easy to check compatibility of vectors with each other and with operators.

First let's look at the structure of vectors, the fundamental data structure of SEPVector. As seen in Figure 1 each vector object stores two pieces of data: a pointer to an actual data container, and a pointer to the space the vector lives in. When a vector is created, it is actually the space that creates the data container, ensuring that these two are compatible. The data container objects manage data through the Boost MultiArray library, which gives an easy interface to access and modify multi-dimensional arrays. We chose this widely-distributed package because it has undergone significant testing and optimization, which can be critical for efficiently working with high-dimensional data.

At the high level, our code is structured to deal with objects like vectors, spaces, data containers, operators, and solver steppers. Each of these is an example of an abstract class, or a class that is never meant to be instantiated without more details about its implementation. For example, one type of space is a space that handles floating point vectors for in-core computations, but even that is not enough detail. As seen in Figure 2, we must further specify whether these vectors are true vectors,

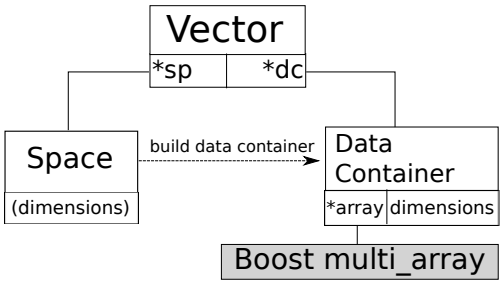

Figure 1: A diagram of the structure of a vetor. Notice that each vector has a pointer to a space and a data container that is build by the space. Many details of the data container are handled by the Boost multi_array library. [**NR**]

matrices, or some higher dimensional tensor, then we have a concrete class, derived from the abstract space class, which we can now create specific instances of. Notice that all of the in core float spaces would share some methods, for example, checking compatibility of the spaces.

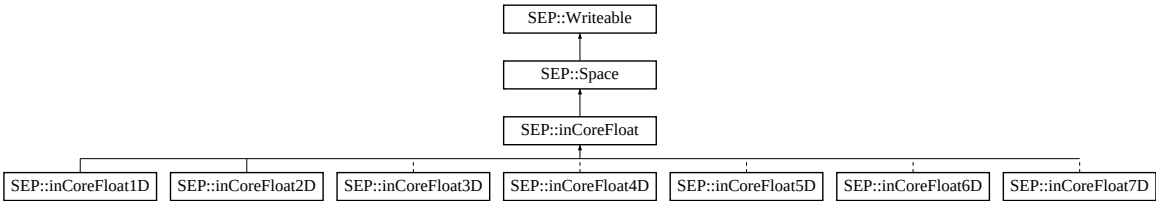

Figure 2: The inheritence structure of vector spaces for in core floating point computations starts with three layers of abstract classes, then each of the concrete derived classes. Notice that dotted lines signify concrete classes that are not fully implemented. [**NR**]

Another example of these abstract classes being used is in our structure for defining operators. Each forward map has an associated domain and range space, as seen in Figure 3. These spaces can be thought of as model space and data space because the model and data vectors are elements of these spaces, respectively. All operators are maps and have a forward operator requiring domain and range spaces, so they inherit this forward operator from their parent map class. They also have their own adjoint operator, as is shown in Figure 3. We have provided some basic operators, which are concrete classes that inherit from the operator class, but as is demonstrated in the example codes, users can specify their own concrete operator classes.

TESTING ENVIRONMENT

To test the library, we have created a series of unit tests that can be easily compiled and run. Each test runs some basic operations, and has specific error messages and

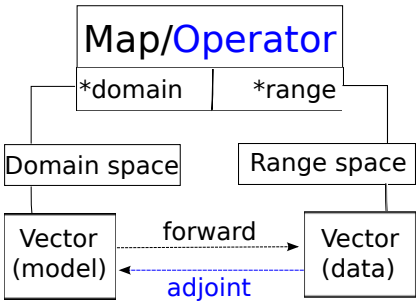

Figure 3: A diagram of the structure of an operator, which is a type of map. Note that maps and operators have a domain space and a range space. Maps only have a forward operator, but operators also have an adjoint. [**NR**]

output to help the developer pinpoint issues. If there are no problems found by the tests, they only print the test number and a short summary of the test to the screen. The user can also run the tests with *valgrind* to check whether memory is being managed properly. The tests build up from very simple operations to more comprehensive ones as follows:

1. Creates and copies space, creates of vector, assigns data to vector, zeroes data from vector, and checks compatibility of different spaces.
2. Clones a vector and takes the linear combination of two vectors in the same space.
3. Tests scaling for both $y = \alpha x$ and $x = \alpha x$.
4. Tests negating a vector for both $y = -x$ and $x = -x$.
5. Tests equivalent scenarios: linear combination versus scaling after adding so $\alpha x + \alpha y = \alpha(x + y)$, and scaling versus additive inverse $-1x = -x$.
6. Tests dot product on a known pair of vectors, and that $(x, y) = (y, x)$
7. Dot product test on first and second derivative operators to check that $(x, Ay) = (y, A^H x)$
8. Dot product test on matrix vector multiplication operator for both $(x, (I + A)y) = (y, (I + A)^H x)$ and $(x, Ay) = (y, A^H x)$.
9. Tests that the conjugate gradient solver is working for a test problem with a known solution.

Many of these tests run on specific known data like a vector of consecutive integers. The advantage is that solutions for any length vector can be calculated by hand, so

we know what to expect. The most general test we have is the dot product test, and test 7 and 8 serve as a nice example of how a user can run the dot product test on any operator they come up with.

GEE EXAMPLES

Flexibility and usability dictate that writing and constructing problems within this library should be as readable and intuitive as possible. The majority of the SEP software legacy has been Fortran based; source-codes provided with reports are largely written in Fortran90 and make use of simple solver modules that were written a decade ago, when multi-core performance and heterogeneous environments were less of a concern. Furthermore, C++ codes, relative to Fortran90, can appear more daunting and less readable, especially to users with a limited coding background. Concepts like pointer dereferencing, inheritance and abstract functions are either less visible or entirely absent in Fortran90, thus to be user-friendly these will be as hidden as possible.

A series of examples from Claerbout and Fomel (2014) were chosen and implemented using this new vector library. This book contains a number of simple yet important geophysical concepts which are familiar to many users. Consequently seeing how the same problems look in this library compared to the Fortran code will provide invaluable insight into how to begin using the library for personal purposes. Especially: how to frame a program, the use of data containers, the use of operators and combining these classes into a solver.

Included herein is a basic inverse hyperbolic radon code, as seen in Claerbout and Fomel (2014). The notion of this is simple - an adjoint hyperbolic radon transform is applied to several spikes in the tau-p space (model space, for this problem) to produce a series of hyperbolae in that data-space (x-t.) Simply applying the forward transformation to these data produces a model-space output which is poorly focused and contains multiple data truncation artifacts. Posing this recovery as an inverse problem with a conjugate-directions solver produces a cleaner, more representative series of spikes in the tau-p space.

This demonstrates the use of spaces, vectors, data containers, maps, adjoints and solvers in a short program. Adapting this problem to larger datasets and more complex operators will be more straight-forward after building this example.

CONCLUSIONS

In response to more sophisticated seismic imaging procedures, growing datasets and more heterogeneous computing architectures, we wrote SEPVector, an object oriented inversion library coded in C++. This code was build from the ground up with modern architectures in mind, and is thoroughly verified through a series of unit tests. Because

many researchers in SEP primarily code in Fortran, we include a series of familiar examples that can be compared to Fortran code, as well as extensive easy to navigate documentation with the code. The primary tools for this library are in place, but there is still work to be done to expand its capabilities and make it more user friendly.

FUTURE WORK

The framework for multi-CPU core inversion is in place, but there is much work to be done to extend this library. Current efforts are being put into adding GPU capability, so extending how arrays are stored and transferred will be necessary. How much of the current library capability will be adapted for GPU use is still under debate.

More common operators and solvers are being written to save users time when it comes to, say, taking derivatives, interpolating or calling a conjugate directions solver. Also more advanced but common operators, such as wave-equation propagators (both multi-core and GPU) will be written and incorporated into the library in the most user-friendly manner.

ACKNOWLEDGEMENTS

This work was supported in part by the Department of Energy Computational Science Graduate Fellowship, provided under grant number DE-FG02-97ER25308.

REFERENCES

A. D. Padula, S. D. S. and W. W. Symes, 2009, A software framework for abstract expression of coordinate-free linear algebra and optimization algorithms: ACM Transactions on Mathematical Software, **36**, 8:1–8:36.

Baysal, E., D. D. Kosloff, and J. W. Sherwood, 1983, Reverse time migration: Geophysics, **48**, 1514–1524.

Claerbout, J. and S. Fomel, 2014, Geophysical image estimation by example: Stanford University.

Clapp, M. L., 2005, Imaging under salt: Illumination compensation by regularized inversion: PhD thesis, Stanford University.

Clapp, R. G., 2001, Geologically constrained migration velocity analysis: PhD thesis, Stanford University.

———, 2004, A python solver for out of core, fault tolerant inversion: SEP-Report, **117**, 183–190.

Fomel, S. and J. Claerbout, 1996, Simple linear operators in Fortran 90: SEP-Report, **93**, 317–328.

Gockenbach, M. S. and W. W. Symes, 1996, Hilbert class library: A library of abstract c++ classes for optimization and inversion: Computers & Mathematics with Applications, **32**, 1–13.

Nichols, D., H. Urdaneta, H. I. Oh, J. Claerbout, L. Laane, M. Karrenbach, and M. Schwab, 1993, Programming geophysics in C++: SEP-Report, **79**, 313–471.

Schwab, M., 1998, Enhancement of discontinuities in seismic 3-D images using a Java estimation library: **99**.

Sirgue, L. and R. G. Pratt, 2004, Efficient waveform inversion and imaging: A strategy for selecting temporal frequencies: Geophysics, **69**, 231–248.

Stork, C., 1992, Reflection tomography in the postmigrated domain: Geophysics, **57**, 680–692.

Tang, Y., 2011, Imaging and velocity analysis by target-oriented wavefield inversion: PhD thesis, Stanford University.

Valenciano, A. A., 2008, Imaging by wave-equation inversion: PhD thesis, Stanford University.

Woodward, M., 1990, Wave equation tomography: PhD thesis, Stanford University.