CESLib: an object library for building scalable inversion applications

Musa Maharramov

ABSTRACT

Application of various inversion techniques to practical problems depends on our ability to quickly adapt existing algorithms to different optimization methods, model and data spaces, operators and discretization schemes. This paper discusses a new object-oriented Fortran library for computational earth sciences (CESLib). I outline scalable model-space and operator hierarchies and an optimization abstraction mechanism that are implemented in the library, and demonstrate a specific application to joint time-lapse inversion. In particular, I demonstrate how using the implemented object framework reduced the amount of time and effort in converting a single-model full-waveform inversion application into a simultaneous inversion package without affecting low-level code for computationally-intensive processing.

INTRODUCTION

While often the least visible part of a research, development of scalable and efficient software is key to success in testing and applying new methods of computational geophysics. Techniques such as full-waveform inversion (Fichtner, 2010) that integrate multiple methods and tools benefit from software modularity, encapsulation and extensibility (Rouson, 2014). Object orientation obviates duplication of low-level codes, and provides for easy integration of new functionality that conforms to common interface conventions. However, although many benefits of object-oriented design are undisputed, selection of a suitable type hierarchy and interface paradigms, and the underlying programming language and library dependencies poses a significant challenge at an early stage of any object-oriented scientific library development. More specifically, typical questions arising at early stages of development are:

1. Can we avoid developing a new library from scratch but simply extend an existing library, such as TriLinos (The Trilinos Project, 2014)?
2. What are the advantages and disadvantages of an object-oriented design? Can we achieve our objectives by using modularization facilities of non-object-oriented languages such as Fortran 95 (Metcalf, 2011)?
3. Which programming language(s) to use—e.g., C++ (Stroustrup, 2013) or Fortran 2008 (Metcalf, 2011)?

4. What is the right balance between abstraction and implementation? More specifically, should we adopt the approach that requires every base class to be purely abstract with no significant type-bound members of known shape and no implemented type-bound procedures? Should we employ a compromise based on endowing even the base classes with key features of a *specific* but fairly general implementation?

As is amply demonstrated by a great variety of programming paradigms deployed by successful scientific software projects, there are no universal answers to these questions.

The object of this document is to describe my new library `CESLib` (Computational Earth Sciences Library) that I used for time-lapse full waveform inversion problems of (Maharramov, 2014b). I provide a rationale for my answers to each of the above key questions, backing them with demonstrations of specific features of the new library.

CESLIB

The library was conceived as an extension of my earlier general-purpose object-oriented library `exp_tk` (Maharramov, 2012), intended specifically for seismic modeling and inversion applications. CESLib is implemented in Fortran 2008 as a dynamically or statically linked library with Fortran modules on 64-bit Intel architectures. Compilation of the latest source code requires version 14.0.2 or later of the Intel Studio Suite (Blair-Chappel and Stokes, 2012), however, a version of the library that can be built with version 13 of the compiler is maintained as well.

One of the compelling reasons for choosing Fortran 2008 was the desire to reuse older Fortran code that can be easily encapsulated in new modules and types (Rouson, 2014). Another reason was the continued performance edge enjoyed by modern Fortran compilers over C++ compilers, even when no obvious language-specific constraints exist to justify the edge (Markus, 2012). The third important reason for choosing Fortran 2003/2008 was the availability of powerful array features such as pointers to array slices. While some of these features have an equivalent C++ workaround, continuous enhancements to the Fortran standard (Metcalf, 2011) provide sufficient reason for staying with the language.

A functional waveform inversion framework should include at least two key components: a wave propagation modeling library and an optimization library. While the latter can be considered an "atomic" sub-library that may not have many dependencies, the former depends on a multitude of data structures and algorithms for discretization and solution of wave propagation equations on various computational domains (Fichtner, 2010). Very large optimization problems (to the order of 10^9) solved in full-waveform inversion make it impractical to use most of the existing optimization libraries such as LAPACK95 (Barker et al., 2001) as back-end solvers. Third-party *iterative* solvers may be used; however, certain specifics of the

full-waveform optimization, such as use of a custom line search algorithm (Sirgue, 2003), mean it is easier to implement a small dedicated optimization library rather than adapt a portion of some existing library.

Rather than providing detailed descriptions of all library features, in the following sections I focus on key aspects of the library that rationalize my answers to the questions listed in the Introduction.

CLASS HIERARCHY

The library provides the following *base* classes:

1. `Dataset` is the base class for any data structures containing contiguous arrays of single-precision real numbers of arbitrary dimension, supporting parallel file input/output.
2. `Medium`, an extension of `Dataset`, is the base class for any (subsurface) models, such as acoustic slowness, elastic moduli, and various anisotropic models.
3. `Field` is the base class of any single and multi-component wave fields.
4. `Lattice` is the base class for extrapolation grids.
5. `Functional` is the base class for any objective function used in non-linear optimization.
6. `Optimizer` is the base class for gradient-based optimization, such as non-linear conjugate gradients (Nocedal and Wright, 2006).

There are a number of auxiliary objects that I do not list here as those are not important for the discussion. Note the absence of an abstract "vector" class. An abstract `vector` class that could be *anything*[1] that supports basic linear operations was part of the `epx_tk` framework (Maharramov, 2012). However, none of the applications of `epx_tk` or `CESLib` require vectors that cannot be represented as contiguous arrays of real numbers that could fit in random access memory. Based on previous experience with `exp_tk` and discussions with industry representatives (Vu, 2013), the abstract vector was dropped from `CESLib`. Note this decision should by no means be considered as a recommendation for avoiding purely abstract vector classes. Certain application, such as processing of *seismic gathers*, may require manipulation of data vectors that cannot fit in memory, or use heterogeneous or non-contiguous storage (e.g., residing in both CPU and GPU memory). The fact that any vector is a one-dimensional array does not constrain the dimensionality of application vectors. Fortran pointers to array slices (Metcalf, 2011) provide an easy array reshaping functionality without

[1]not just an array of numbers

memory reallocation. Note, however, that this feature is not unique to Fortran and can be achieved in C++ as well using static type recasting.

In the following sections I describe a few of the key library features that demonstrate some of the mentioned technologies, and discuss their effect on productivity.

ARRAY SLICES

Pointing to array slices is a key feature that is used throughout the optimization framework. For example, a `Medium` object has a type member `p` (property):

```
real, dimension(:), pointer, public :: p
```

This member is inherited by the `AcMedium` (acoustic medium) type. The dimensionality of the actual data array `p` is dynamically changed by declaring, e.g., a three-dimensional "slowness" pointer `slow` and pointing it to `p`:

```
real, dimension(:,:,:), pointer :: slow
...
slow(1:nz,1:nx,1:ny) => m%p(1:nxyz)
```

Note this allows referencing both `p` and `slow` in any code that uses the `AcMedium` type (run-time class). Array slices allow a straightforward implementation of anisotropic elastic models, as different property arrays can point to different slices of the `Medium` member `p`. However, elastic and anisotropic models are not currently implemented in `CESLib`.

I minimize the use of pointers to array slices within the loops of wave extrapolation codes as a repetitive use of dynamic array offsets may slightly degrade the performance

OPTIMIZATION

The key optimization type is `Functional` that represents a nonlinear minimization functional. Since the library specializes in derivative-based methods, the type has three *deferred* type-bound members `eval`, `evalG`, `evalH` that compute the value of the functional for a specified argument (that is a one-dimensional array!), both value and the gradient of the functional, and the dot product of the Hessian with a specified vector. The latter is computed in high-order adjoint state methods (Plessix, 2006). The user is expected to provide this functionality by extending the abstract `Functional` class and defining their own evaluation methods. My implementation of the full-waveform inversion (Maharramov, 2014b) defines `eval` and `evalG` and provides a dummy `evalH` as only the first-order adjoint-state method is used.

One important auxiliary object type used by `Functional` is the `LineSearch` type. By default, one extension of the `LineSearch` type is provided (`MoreThuente`) that implements the More-Thuente line search algorithm (Nocedal and Wright, 2006).

AGGREGATION

The joint time-lapse full-waveform inversion method of (Maharramov, 2014b) requires *simultaneous* minimization of the joint objective function

$$\alpha\|\mathbf{M}_b\mathbf{u}_b - \mathbf{d}_b\|^2 + \beta\|\mathbf{M}_m\mathbf{u}_m - \mathbf{d}_m\|^2 + \tag{1}$$

$$\delta\|\mathbf{WR}(\mathbf{m}_m - \mathbf{m}_b - \Delta\mathbf{m}^{\mathrm{PRIOR}})\|^2 \;\rightarrow\; \min. \tag{2}$$

The object-oriented technique of *aggregation* (Rouson, 2014) allows for a straightforward extension of a single-model full-waveform inversion code to solve the optimization problem (1,2). In my implementation, type `Func1` extends `Functional` and is used for minimizing one of the terms in (1). I implement a joint functional using an extension of `Functional` that contains two independent *instances* of `Func1` for the baseline and monitor inversion:

```
 type, extends(Functional) :: Func2
    private
!
!   baseline and monitor functionals
    class(Func1), pointer :: pmon, pbase
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   type members required for difference regulariation
    real :: regalpha = 0 ! regularization parameter
!
    integer :: regop = 2 ! regularization operator
!
!   regularization (weighting) matrix
    real, dimension(:), pointer :: regmask
!
!   these are required for difference regularization
    real, dimension(:),pointer :: diff1, diff2
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
    contains
      procedure :: eval  => evalFunc2  ! value only
      procedure :: evalG => evalFunc2G ! value and gradient
      procedure :: evalH => evalFunc2H ! dummy
...
  end type Func2
```

Two instances of the `Func1` class are allocated independently prior to the initialization of a joint functional instance, and assigned to the member pointers `pmon` and `pbase` during the initialization of the joint functional. Note that implementing `evalFunc2` only requires two separate invocations of `pbase%eval()` and `pmon%eval()`, and summing the results. Implementing `evalFunc2G` requires separate invocations of `pbase%evalG()` and `pmon%evalG()` as well as the trivial evaluation of the difference regularization term (2). Note that the model vectors of `pmon` and `pbase` need to point to two contiguous slices of a single array in order for the joint model to be used in the optimization framework.

CONCLUSIONS AND PERSPECTIVES

`CESLib` is a scalable computational framework for solving forward and inverse problems of wave-propagation modeling. One of the deciding factors in implementing the new object library was extensibility for solving a hierarchy of optimization problems similar to (1,2) via aggregation. Conversion of a single-model full-waveform inversion into a joint inversion code required only the implementation of difference regularization in addition to the modest overhead of extending the base functional type to a joint type.

The upcoming release of `CESLib` will include the pseudo-acoustic modeling method of (Maharramov, 2014a).

ACKNOWLEDGEMENTS

The author thanks David Nichols, Phuong Vu and Stewart Levin for a number of useful discussions.

REFERENCES

Barker, V. A., L. S. Blackford, J. Dongarra, J. D. Croz, S. Hammarling, M. Marinova, J. Wasniewski, and P. Yalamov, 2001, LAPACK95 User's Guide: SIAM.

Blair-Chappel, S. and A. Stokes, 2012, Parallel programming with Intel Parallel Studio XE: Wrox.

Fichtner, A., 2010, Full seismic waveform modelling and inversion: Springer.

Maharramov, M., 2012, Identifying reservoir depletion patterns with applications to seismic imaging: SEP Report, **147**, 193–218.

——— 2014a, Artifact reduction in pseudo-acoustic modeling by pseudo-source injection: SEP Report, **152**, 97–108.

——— 2014b, Joint full-waveform inversion of time-lapse seismic data sets: SEP Report, **152**, 19–28.

Markus, A., 2012, Modern Fortran in practice: Cambridge University Press.

Metcalf, M., 2011, Modern Fortran explained: Oxford University Press.

Nocedal, J. and S. J. Wright, 2006, Numerical optimization: Springer.
Plessix, R.-E., 2006, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications: Geophys. J. Internat., 495–503.
Rouson, D., 2014, Scientific software design: The object-oriented way: Cambridge University Press.
Sirgue, L., 2003, Inversion de la forme donde dans le domaine fréquentiel de données sismiques grands offsets: PhD thesis, Queens University, Canada.
Stroustrup, B., 2013, The C++ programming language: Addison-Wesley.
The Trilinos Project, 2014, Trilinos release 11.8. [Online; accessed May-15-2014].
Vu, P., 2013, Personal communication.