An Object-Oriented Toolbox for Studying Optimization Problems

H. Lydia Deng, Wences Gouveia & John Scales

To be included in the proceedings volume of the Interdisciplinary Conference on Methodology Computation and Integrated Applications
University of Aarhus, Denmark, May 1995.

Center for Wave Phenomena
Colorado School of Mines
Golden, Colorado 80401
303/273-3557
An Object-Oriented Toolbox for Studying Optimization Problems

H. Lydia Deng, Wences Gouveia & John Scales

Center for Wave Phenomena, Colorado School of Mines, Golden, Colorado 80401, USA

Abstract

We have developed the CWP object-oriented optimization library (COOOL) as a tool for studying optimization problems and to aid in the development of new optimization software. COOOL consists of a collection of C++ classes (encapsulations of abstract ideas), a variety of optimization algorithms implemented using these classes, and a collection of test functions, which can be used to evaluate new algorithms. To use one of the optimization algorithms, a user codes an objective function (and its derivatives if available) according to a simple input/output model — in any language. Moreover, the object classes themselves can facilitate the development of new optimization and numerical linear algebra software since the routine aspects of such coding will benefit from the reusability of code inherent in the object-oriented philosophy. After giving a brief description of the library, we illustrate its application on a problem from our current research.

Overview

A wide variety of problems in science can be thought of as solutions to optimization problems: the maximizing or minimizing of a function, possibly subject to constraints or penalties. In geophysics, for example, because we can never truly “know” the earth’s subsurface properties except by direct observation. Instead, we seek interpretations that are most likely in some sense. In real applications, the functions characterizing the likelihood of such interpretations are often complicated entities existing in high-dimensional spaces. Knowing just which optimization tool is appropriate to solve a given problem is half the battle. By having a varied set of tools available one might be able to “browse” through various choices to find the most desirable match between an algorithm and the problem at hand. Further, all optimization problems share certain basic features. They all must have some concept of a set of unknown parameters (model), and they all involve communication between a computational module whose job it is to
update models (optimization method), and an objective function, whose job it is to measure which models are good and which are bad.

Our motivation in putting together this library was threefold. First, we wanted to have a varied collection of optimization routines and realistic test functions so that we could study the computational complexity of problems of the sort encountered in geophysics. Second, we wanted to have a programming environment that facilitated the rapid development of new algorithms so that there would be maximum re-use of existing code and maximum flexibility in mating optimization codes with objective functions on a variety of hardware platforms. Finally, we wanted to encourage the development of a test-bed of geophysical optimization problems in order to stimulate interest in geophysical optimization problems among applied mathematicians.

Object-Oriented Programming (OOP) represents an approach to software development in which the system is organized as a collection of objects that are encapsulations of data structures and behavior. There are many languages supporting the OO paradigm. In C++, which we used to build COOOL, classes are user-defined types to represent objects in the software. COOOL is a collection of C++ classes, each of which serves for some specific functions in an optimization. Taking advantages of encapsulation, sharing, and inheritance (Pohl, 1993) features provided by the OO paradigm, COOOL are designed to facilitate maximal flexibility and re-usability.

Optimizing with COOOL

Design of COOOL

The three criteria behind the development of COOOL are:

1. It should provide a consistent application programming interface (API) for solving optimization problems.
2. It should allow incremental development of the library.
3. It should allow for application packages to be easily built from the existing library.

By “consistent”, we mean that optimization algorithms as well as objective-function formats should be relatively transparent to application users. This provides the flexibility for users to choose optimization methods easily and concentrate on the specific problem rather than struggling to fit the requirements of the various optimization algorithms.

Figure 1 shows the structure that COOOL uses to achieve these goals. Level 0 contains generic classes, such as Vector, Matrix, List, Astring, etc. These classes are basic element of COOOL for handling algebraic computations. Special classes, such as SpaMatrix and DiagMatrix, handle sparse and diagonal matrices efficiently. Level 1 is the main level of COOOL, containing classes for three major components in any optimization problems: unknown parameters (Model), objective functions (ObjFcn), and optimization algorithms (Optima). All these classes can be easily accessed (either adding or modifying) with
minimum influence on others. Level 2 is the application level where problemspecific packages can be built with existing classes from lower levels. For example, we should be able to build a package for a certain travel-time inversion problem with the flexibility of choosing any of the mathematical optimization methods.

![Diagram showing levels of COOOL](image)

**Fig. 1.** Structure of COOOL. Level 0 contains generic algebraic data structures. Level 1 contains prototypes for three necessary components in any optimization problems: **Model**, **ObjFcn**, and **Optima**. Both levels are extensible with minimum influence on existing classes. Application packages at Level 2 can be easily built with lower-level classes.

**Optimization Methods in COOOL**

Figure 2 shows a classification of the optimization methods included in the preliminary release of the library. We divide the methods into two main classes: linear solvers and local optimization methods. Global optimization methods will be part of a future release.

**Linear Solvers.** All of the linear solvers in COOOL are iterative. Descriptions of most algorithms can be found in (Scales & Smith, 1994). Presently COOOL implements two flavors of row action methods: **ART** (Algebraic Reconstruction Technique) and **SIRT** (Simultaneous Iterative Reconstruction Technique). These methods are especially attractive when problems are too big to fit into memory. When the non-zero matrix elements fit into memory, **CGLS** (Conjugate Gradient Least Squares) is very attractive. Finally, we include a version of the **IRLS**
(Iteratively Re-weighted Least-Squares) algorithm to efficiently solve linear systems in the $\ell_p$ norm. This algorithm takes advantage of certain approximations to achieve nearly the speed of conventional least squares methods while being able to robustly handle long-tailed noise distributions.

**Local Optimization Methods.** The algorithms presented here can be applied to quadratic and non-quadratic objective functions alike. The term *local* refers both to the fact that only information about a function from the neighborhood of the current approximation is used in updating the approximation, and that these methods usually converge to extrema near the starting models. As a result, the global structure of an objective function is unknown to a local method. Some of these techniques, such as Downhill Simplex and Powell’s method do not require explicit derivative information of the objective function. Others, such as the quasi-Newton methods require at least the gradient. In the latter case, if analytic expressions are not available for the derivatives, a module for finite-difference calculation of the gradient is provided. COOOL also includes non-quadratic generalizations of the conjugate gradient method incorporating two different kind of line search procedures.

**Objective Functions**

Objective functions are those functions to be minimized (or maximized). Several analytical test functions are implemented in the library. They include a generalized $N$-dimensional quadratic function, Rosenbrock’s function, and a two-dimensional multi-modal analytical function.

For realistic problems, the formulation of objective functions may vary tremendously. COOOL provides a means of communication between optimization algorithms and user-defined objective functions. A user can write the objective function in a stand-alone file using any language and the COOOL only needs the executables. If gradient information is available, COOOL also accepts gradient vectors from the user-defined file. This communication model is illustrated in Fig. 3. COOOL sends out a flag to the user-defined objective function file, indicating whether the function value or its derivatives is needed, followed by a set of model parameters. The objective function reads the flag and the model from standard input, and writes the result to standard output; COOOL will then capture these information and put them to an optimization algorithm for an updated model. This simple communication model allows us to consider algorithms as diverse as downhill simplex, Newton’s method, and simulated annealing within a unified framework.

A key point in COOOL is that communication between optimization algorithms and objective functions is transparent to users. For using COOOL, a user only needs to construct several objects choosing from the library; namely, a Model, an ObjFcn, and an Optima. The interior computation and communication among these objects are handled by COOOL. A pseudo-code for solving an optimization problem using COOOL is shown in Fig. 4. Although COOOL is written in C++,
Fig. 2. An overview of the optimization algorithms contained in COOOL. Included are linear solvers for rectangular linear systems using both least squares and general $\ell_p$ norms and local methods based on direct search and quasi-Newton methods. Monte Carlo global optimization methods will be part of a future release of COOOL.

Fig. 3. A model of communication between an objective function and an optimization class. The user-defined stand-alone objective function are responsible for evaluating the value or gradient for a set of parameters according to the message sent by the optimization algorithm.
little knowledge of C++ is needed for using it. Evaluations of objective functions may be the most computationally intensive step for many optimization problems; COOOL allows users to code them in the most efficient language available.

```c
main(argc, **argv)
{
    // Construct a constrained Model with n unknowns
    Model m(n, upper, lower, Δm);
    // Construct an ObjFcn: user-defined function, statics
    ObjFcn *f = new ObjFcn("statics", ...);
    // Construct an Optima: choose an optimization method
    Optima *opt = new ConjugateGradient(f, imax, tol, ...);

    // Initialize the model to m₀
    m = m₀;

    // COOOL returns an optimal model optm
    Model optm = opt → optimizer(m);
}
```

Fig. 4. Pseudo-code for using COOOL to solve an optimization problem. Constraints of model parameters are separated from optimization algorithms. The boundary [lower, upper] and grid-size Δm are specified when constructing the Model object.

An Application of COOOL to Residual Statics Correction

Next we show an application of COOOL to a problem from our current research. Residual statics are time shifts observed in reflection seismic data and are attributed to localized heterogeneities in the near surface. These time shifts result in a decrease of the stacking quality and can therefore be estimated by optimizing some measure of stacking quality such as the stacking power function. The stacking power function is the sum of squares of the stacked sections over a time-window which contains most significant events,

\[
SP(m) = \sum_{c} \sum_{t} \left( \sum_{offset} Trace(m) \right)^2 \tag{1}
\]

where \( m \) is the parameter vector of time-shifts.

In seismic exploration, the function \( SP(m) \) in (1) is a high dimensional function with many local maxima. A picture of a two-dimensional slice through such a beast is shown by Smith et al. (1992). In this section, we show two examples of using COOOL for studying such optimization problems.
Multi-grid Residual Statics Correction

The left figure of Fig. 5 shows a stacked section acquired over a permafrost zone without statics corrections. The input for COOOL is the filtered pre-stacked data within the time-window, indicated by the black bar in Fig. 5. The right figure of Fig. 5 shows the result of applying the COOOL non-quadratic version of ConjugateGradient to optimize the stacking power in successively wider frequency bands. It is clear that the continuity of the reflections was restored, and the resolution and signal-to-noise ratio improved. This multi-grid approach was suggested by Bunks, et al. (1995).

![Input data](image)

**Fig. 5.** The left figure shows a stacked section acquired over a permafrost zone without statics corrections. The right figure shows the stacked section after surface-consistent statics are corrected. Statics are estimated using COOOL by optimizing the stacking power in successively wider frequency bands. The black bar on the right shows the time window of data used to compute the statics corrections.

Multi-Resolution Analysis for Simplifying an Objective Function

Multiresolution analysis (MRA) projects the input signal to a set of nesting subspaces which represent a series of successive coarser resolution levels. Using MRA, we intend to study the behavior of complex objective functions at various resolution levels.

Consider a simple version of the statics problem: two identical traces with an unknown shift. For this problem, the objective function (1) is one-dimensional. Maximizing this stacking power is equivalent to minimizing the following mean-squared error function,

$$ E(\delta) = \sum_{i=0}^{N-1} (P_0(i - \delta) - P_1(i))^2, \quad (2) $$
where $P_0(t)$ and $P_1(t)$ are the data traces, $N$ is the number of samples per trace, and $\delta$ is the unknown time-shift. For two traces containing a shifted Ricker wavelet, Fig. 6 shows mean-squared error functions when input data are decomposed to successively coarser resolution levels. The bases for the decomposition is a symmetric, shift-invariant wavelet bases developed by Saito and Beylkin (1993). A detailed discussion on this work can be found in (Deng, 1995). Randomly choosing 50 initial models between $[-0.2, 0.2]$ s, Figure 7 shows histograms of the converged time-shifts obtained by a non-quadratic ConjugateGradient tool provided in COOOL. These results show that the MRA increases the chance for local search methods to find the global extrema.

![Graph showing mean-squared error functions](image)

**Fig. 6.** The mean-squared error functions for two shifted traces containing Ricker wavelets at various resolution levels. Input is decomposed by an MRA using a symmetric, shift-invariant wavelets bases.

**Conclusions and Future Directions**

The CWP Object-Oriented Optimization Library (COOOL) is a collection of C++ classes for studying and solving optimization problems. It was developed using the freely available GNU compiler gcc. The library contains the basic building blocks for the efficient design of numerical linear algebra and optimization software; it also comes with a variety of unconstrained optimization algorithms and test objective functions drawn from our own research. The only requirement for using one of the optimization methods is that a simple model of communication be followed. This allows us to use exactly the same code to optimize functions tailored for a variety of hardware, no matter what programming language is used. Further, since we have provided class libraries containing building blocks for general purpose optimization and numerical linear algebra software, the development of new algorithms should be greatly aided.
Fig. 7. Histograms of the obtained time-shifts of 50 conjugate-gradient optimization experiments for data at corresponding resolution levels as in Fig. 6. Initial models are chosen randomly between $[-0.2, 0.2]$ s. The horizontal axis is the number of shift-samples, where the sample interval is 0.01 s, and the grid size of the histograms is 4 samples. All 50 experiments found the true solution when the data are decomposed to resolution level 5.

COOOL is now freely available via anonymous ftp at

hilbert.mines.colorado.edu/pub/cwpcode/coool,

and on the WWW at

http://cwp.mines.edu:3852/cwpcode/coool

Postscript and HTML versions of a technical report on COOOL by the authors are available by anonymous ftp or WWW from the same addresses. Any bug reports or suggestions should be sent to

optima@dix.mines.edu.

So far we have treated only unconstrained problems, or rather, whose constraints can be incorporated directly into the objective function. Allowing more general kinds of constraints will be the subject of future enhancements.

Acknowledgment

Our inspiration for implementing optimization software in an object oriented fashion is Martin Smith. We have been influenced by the classes Martin Smith
and Terri Fischer developed for the Uni-Processor Genetic Algorithm (UGA). The communication between the objective function and the optimization algorithms is managed via EXPECT, written by Don Libes of the National Institute of Standards and Technology (NIST) and adapted to our purposes from UGA. We would like to thank Pechorageofika for the permission to use the seismic data set in this work, and Gregory Beylkin for his help on the MRA.

This work was partially supported by the sponsors of the Consortium Project on Seismic Inverse Methods for Complex Structures at the Center for Wave Phenomena, Colorado School of Mines, and the Shell Foundation.

References


