iNEOS: An Interactive Environment for Nonlinear Optimization*

Marcel Good † Jean-Pierre Goux ‡ Jorge Nocedal §
Victor Pereyra ¶

May 22, 2000

Abstract

In this paper we describe iNEOS, an Internet-based environment which facilitates the solution of complex nonlinear optimization problems. It enables a user to easily invoke a remote optimization code without having to supply the model to be optimized. An interactive communication between client and server is established and maintained using CORBA. We test the system in a simulation designed to identify material parameters of a piezoelectric crystal.

*This work was supported by National Science Foundation grant CDA-9726385, by Department of Energy grant DE-FG02-87ER25047-A004, and by Sandia National Laboratories LDRD 97-0301.

†Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208, mgood@ece.nwu.edu

‡Optimization Technology Center, Northwestern University, Evanston, IL 60208, goux@ece.nwu.edu

§Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208, nocedal@ece.nwu.edu

¶Weidlinger Associates, Los Altos, California, victor@ca.wai.com

1 Motivation and Overview of iNEOS

Nonlinear optimization techniques are used in a wide range of engineering and scientific applications (see e.g. [11]). In many of these applications, off-the-shelf optimization codes are interfaced with complex proprietary codes that perform computationally intensive simulations. Due to the variety of nonlinear optimization codes available to the user [10] and the diversity of the data structures they employ, it is time consuming and difficult to install and compare different solvers.

In this paper we describe iNEOS, an Internet-based environment which facilitates the process of solving such nonlinear optimization problems. It allows a user (client) to perform a simulation that requires interactive access to a remote optimization solver (the server), and does so without revealing any important information about the simulation. We demonstrate the effectiveness of iNEOS by performing an interactive identification of parameters in a model of piezoelectric crystals.

This work was motivated by the success of the NEOS server,

`http://www-neos.mcs.anl.gov`

see also [4]. The NEOS server solves a wide range of optimization problems remotely, via the Internet, and provides several interfaces to the user. Over the past few years, the NEOS server has proved to be very useful for hundreds of users from academia, industry, and government, and is regarded as a successful demonstration of the possibility of networked access to software and data.

The current implementation of NEOS has, however, some limitations that make it impractical for an important class of applications. It forces the user to submit all the data of the problem, and do so in formats specific to each area of optimization. To solve nonlinear optimization problems, the user has to submit AMPL [5], Fortran or C files describing the objective function and constraints. This mode of operation prevents many potential users from accessing the servers, for one of the following reasons:

- The data files or codes specifying the model (objective function and constraints) are often proprietary or confidential.
- The evaluation of the objective function and constraints is too time consuming to be delegated to a NEOS server.
- The model is written in several languages or in a format that is not acceptable to the NEOS servers.

By taking advantage of the particular structure of nonlinear optimization algorithms, it is possible to circumvent these difficulties. In nonlinear optimization, an improved solution can be generated just by knowing the current iterate and the numerical values of the function and gradient at the current point. These arrays of real numbers do not reveal the nonlinear model to an observer of the data submissions.

Consider, for example, an unconstrained optimization problem,

$$\min f(x) \tag{1}$$

where f is a scalar function of n variables. This problem can be solved using a quasi-Newton iteration of the form

$$p_k = -H_k g_k, \qquad x_{k+1} = x_k + \alpha_k p_k, \tag{2}$$

where H_k is an approximation to the inverse Hessian of f, g_k is the gradient of f evaluated at the current iterate x_k, and α_k is a steplength determined by a line search procedure. All that is needed to compute a new estimate x_{k+1} is to provide x_k, g_k and f_k, and to supply values of f and g for all trial values generated in the line search procedure; see for example [13]. After x_{k+1} has been computed, a new Hessian approximation H_{k+1} is generated based on the differences $x_{k+1} - x_k$ and $g_{k+1} - g_k$.

Some of the nonlinear optimization codes in NEOS, such as TRON [7] and L-BFGS [2], already have such internal client-server design. In these codes a driver computes the function and gradient values at the current iterate, and then calls the optimization solver, which returns a better estimate of the solution. iNEOS has been designed to exploit this structure in an interactive way. The task of evaluating the objective function and gradient (i.e., the simulation) remains in the hands of the user, and iNEOS provides a new approximate solution which is computed on the server; see Figure 1.

The interactive environment performs the following steps, starting from an initial estimate x_0 provided by the user:

Repeat until a convergence test is satisfied or an error message is generated:

1. The client computes the function and gradient (simulation phase) for the current estimate of the optimal solution, and sends these values over the network to the server.

2. Based on this information, the server computes a new trial point (optimization phase) and sends it back to the client over the network.

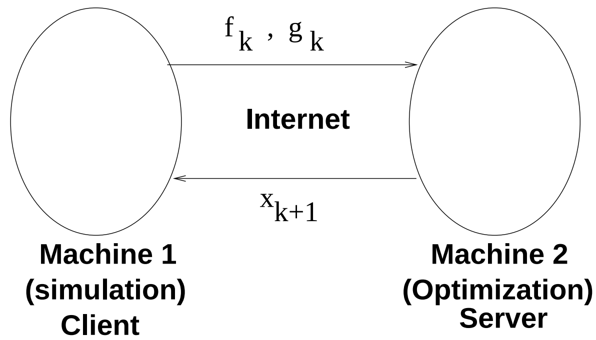

Figure 1: Overview of iNEOS.

3. Repeat steps 1 and 2 until a new iterate with a lower function value has been computed.

End Repeat

This environment requires stable communication between client and server every time that a new trial point of the optimization calculation is computed. One must keep both processes (client and server) alive and in synchronization until the optimization is completed. We have chosen to use CORBA [14] to implement this environment, as described in the next section.

At present, iNEOS is capable of solving bound constrained optimization problems of the form

$$\min f(x) \quad \text{subject to} \quad l \leq x \leq u, \tag{3}$$

where l and u are n-vectors of bounds. The description of the interactive environment given above applies to (3), provided the vectors l and u are transmitted by the client during the first invocation of the server. The optimization is performed by means of L-BFGS-B [2, 17], a limited memory quasi-Newton method. Solvers for general nonlinear optimization problems (with equality and inequality constraints) will be added in the future. They will require the transmission of second derivatives from the client to the server.

2 Implementation

Several technologies can be used to implement iNEOS. A prototype implementation was built using the Nexus [12] communication library. However, the lack of object-oriented design, the absence of built-in support for multiple clients, and deployment difficulties led us to choose CORBA which is superior in all these aspects.

CORBA is well established in the computing community, and has been successfully used for business applications and legacy system integration. It has yet to prove its relevance in scientific computing applications due to its lack of performance compared to libraries like MPI [8]. But since the main goal of iNEOS is to enable optimization technology in a user-friendly and reliable manner, the communication overhead imposed by CORBA is not a major concern. In the applications we have in mind, the simulation requires minutes or hours, and the amount of data transmitted between client and server is not large. When solving the bound constrained optimization problem (3), a total of $2n+1$ floating point numbers are submitted over the Internet during each data exchange, where n denotes the number of variables. We envision solving problems of up to 1 million variables. Even in this case, most of the time in a typical iNEOS session will be spent in the simulations performed on the client's computer.

Of paramount importance is the ease of deployment of iNEOS and the robustness of the data exchange between client and server. CORBA's IDL allows us to describe the interface of a server as a set of methods to be called on objects. The CORBA libraries handle network issues and marshal data between client and server. This allowed us to develop a consistent object oriented design of iNEOS. CORBA also provides additional services like security, which will be discussed later.

We should mention that iNEOS could have also been built using DCOM [9], the object oriented component technology developed by Microsoft that provides support for distributed components as well as multiple clients. It would allow us to treat optimization solvers as components, facilitating the addition of future solvers. Nevertheless, we wished to develop iNEOS for Unix systems, and even though DCOM has been ported to Solaris, it would have to be bought and installed on the client machine. This was the main reason for not using DCOM, which would have been our technology of choice on Windows platforms.

2.1 Architecture

Figure 2.1 shows the architecture of iNEOS and the components involved in the process.

The Factory is a generic object, which will allow us to easily expand the system in the future. All that is needed to add a solver is to create a

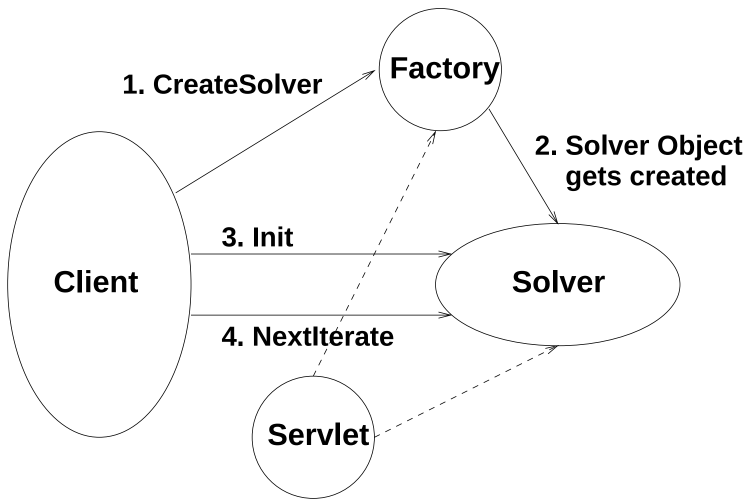

Figure 2: iNEOS Architecture.

new object class, and make the factory aware of it. The interface with the Factory object is shown below.

```
interface SolverFactory : HttpObject
{
   exception FactoryException {};

   Solver CreateSolver(in string type)
      raises(FactoryException);
   string CreateSolver2(in string type)
      raises(FactoryException);
};
```

The SolverFactory interface inherits from HttpObject (not shown), which enables the Servlet [15] to communicate with every running object on the server. In this way, the use can retrieve information about the server as well as the Solver using a regular Web browser. The servlet runs on an Apache [1] Web Server, and facilitates the retrieval of log information documenting the progress of the optimization. This setup makes iNEOS more accessible to the user. The Solver object for the L-BFGS Solver is shown below.

```
interface LifeCycleObject : HttpObject
{
void free();
};
```

```
        /* The Solver interface is the common base interface for all
           Solvers. It implements a common method to act on solvers.
*/

interface Solver : LifeCycleObject
{
        string GetName();
                long GetJobID();
};

        /* The LBFGSSolver interface is used by the LBFGS solver
           object. The methods are LBFGS specific.
        */

interface LBFGSSolver : Solver
{
exception SolverException
{
TASKTYPE task;
};

Vector Init(in long n, in long m, in Vector x_start,
                          in Vector l, in  Vector u,
                          in BoundTypes nbd, in double f,
  in Vector g, in double factr,
  in double pgtol, out TASKTYPE task,
  in short iprint)
  raises(SolverException);

Vector NextIterate(in double f, in Vector g, out TASKTYPE task)
  raises(SolverException);
};
```

The Solver interface inherits from two interfaces, which specify the common part of every solver. LifeCycleObject is used to destroy a solver when it is not anymore used. As we can see from the LBFGSSolver interface, the communication between client and solver is reduced to mainly two methods, which take care of exchanging the data between client and solver. The usage of the server object is further simplified by a procedural C/C++ library, which hides the CORBA details from a client. This API can also be used with Fortran applications, which would otherwise not be possible due to the lack of a CORBA binding for Fortran.

3 Application to a Parameter Identification Problem

To demonstrate the viability of iNEOS as an optimization problem-solving environment, we use it for the determination of parameters in a piezoelectric crystal model.

Piezoelectric transducers convert electrical signals to mechanical signals and vice versa. They serve as transmitters and receivers in imaging systems for sonar, medical, and non-destructive evaluation applications. One of the most technically demanding applications is ultrasound medical imaging. Today nearly all of the major ultrasound system companies are experimenting with finite element models that describe the transient response of a piezoelectric material. Most have enjoyed only limited success at significant development or simulation costs, and it is therefore important to improve the accuracy of the models as much as possible.

In this study we will use least squares techniques to determine the material parameters in a model of a homogeneous piezoelectric crystal, given a set of measured impedances. The nonlinear least squares problem can be posed as

$$\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} w_i (I_i^o - I_i^c(\boldsymbol{\alpha}))^2 \tag{4}$$

subject to

$$\underline{\alpha}_i \leq \alpha_i \leq \bar{\alpha}_i. \tag{5}$$

Here I_i^o, I_i^c, are the observed and calculated complex fast Fourier transforms of the impedance samples respectively, and $0 \leq w_i$ are some weights. The vector $\boldsymbol{\alpha}$ has ten components that represent the parameters describing the elastic, electromagnetic and coupling properties of the homogeneous piezoelectric crystal sample. Since there is only a narrow range of the parameters that lead to valid physically possible materials, the upper and lower bounds (5) are imposed.

The partial derivatives of the impedance with respect to the parameters,

$$\mathbf{J}(\boldsymbol{\alpha}) = -\frac{\partial I_i^c(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} \tag{6}$$

are approximated by finite differences. A full description of the model and the simulation methods is given in [3].

3.1 Utilization of iNEOS

Our goal is to improve upon the accuracy of an initial choice of parameters. From the point of view of nonlinear optimization this is a small problem, since the number of parameters is 10, but the function evaluation is expensive. In addition, the model is proprietary, making this application ideal for testing iNEOS.

As mentioned earlier, the optimization code employed in our experiments is L-BFGS-B [2], which is written in Fortran, using a reverse communication structure. The modeler provides an initial choice of the parameters, and evaluates the objective function and its gradient for every trial value of the parameters, and L-BFGS-B provides an improved guess of the parameters. iNEOS establishes and controls communication between client (modeler) and server (optimizer).

Since the objective here is to validate the method, the experiments will be performed in the following controlled environment. For a given setting of the parameters α (the target), the finite element code is used to produce the resulting impedance. Then we perturb the initial values of the parameters by 7.5%, and ask whether we can recover the initial values using the least squares approach. We wish to reconstruct each of the parameters to at least 1% of accuracy. The client computer was a 300 MHz Pentium II, running Solaris, and was located in California; the server was a Sun Ultra5 running Solaris and located in Illinois. The client had a firewall that required a small modification of the way we handle object references; see [6]. Each evaluation of the objective function f and gradient g requires approximately 10 minutes in the client machine. The fitted real and imaginary parts of the impedance are shown in Figures 3 and 4.

The interactive optimization was successfully completed; for a more detailed description of these experiments see [3].

4 Final Remarks

The experiment reported in the previous section demonstrates that the current implementation of iNEOS is capable of solving challenging problems in computational optimization. The interface with the remote solver is very simple due to iNEOS' intuitive API, and overcomes one of the main limitations of the standard NEOS servers. iNEOS can be extended, with relatively little effort, so as to handle optimization problems with general constraints. The transmission of second derivatives of the objective function and constraints from the client to the server would permit the use of some of the

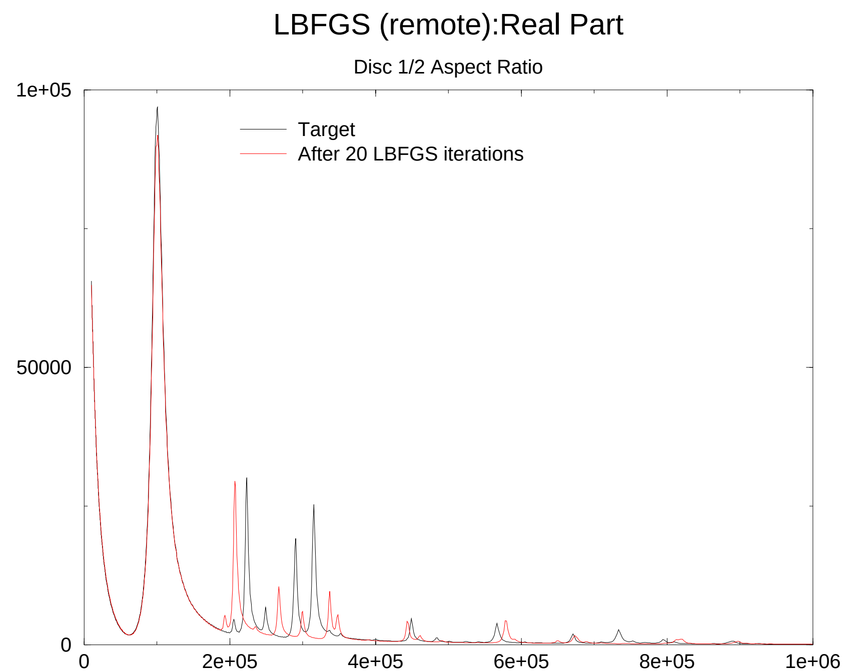

Figure 3: Real part of the impedance.

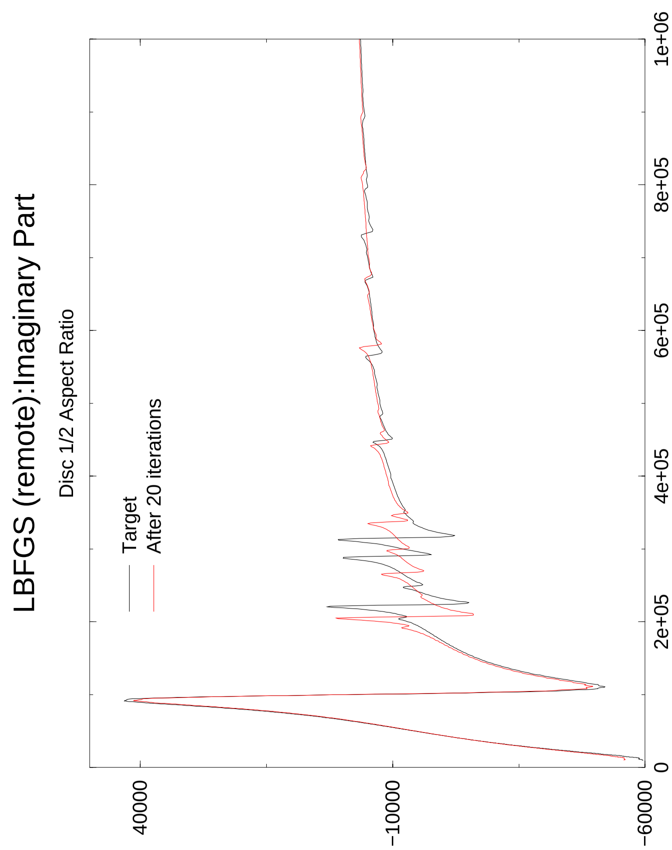

Figure 4: Imaginary part of the impedance.

most powerful nonlinear programming solvers.

Several enhancements of the current system can improve its security, reliability, portability, and ease of use.

CORBA provides all the basic services for building highly reliable distributed systems over the Internet. For iNEOS to be heavily used, it must be deployed on a reliable platform that can handle requests from numerous users, and it must deal well with failures of a single server as well as security. An array of "Application Server Solutions" are currently available in the market, and most of them are CORBA-based. Such Application Servers allow deployment of an application into a framework which handles clustering, fail over, load balancing, and security. Since iNEOS is CORBA-based, the step of migrating it into such an Application Server Environment is rather simple.

Currently, the iNEOS API still reveals specific details of a particular solver. Since not every solver requires the same amount of data, the integration of new solvers into iNEOS would be facilitated by the development of a common solver interface that represents data in the calling sequence in XML [16]. This would greatly facilitate the testing of a variety of solvers. The overhead incurred by the use of XML may be of concern, however, and must be measured relative to the total interaction time.

User-level security has not been incorporated into the current implementation of iNEOS. Anyone has access to iNEOS and can view the data submitted to it. This is not a major concern at present, because as stated in the introduction, all the confidential information about the simulation is kept on the client, and the data submitted to the solver does not reveal the model. A future implementation of iNEOS, must however contain security mechanisms to control the usage of iNEOS. This can be done by using the CORBA security service or the Framework provided by Application Server Environments.

Acknowledgment. We would like to thank Todd Plantenga for his encouragement and support in the early stages of this work.

References

[1] The Apache Software Foundation. The Apache Server Project. Available from http://www.apache.org/httpd.html

[2] R.H. Byrd, P. Lu, J. Nocedal and C. Zhu (1995). "A limited memory algorithm for bound constrained optimization", SIAM Journal on Scientific Computing, 16, 5, pp. 1190-1208.

[3] L. Carcione, J. Mould, V. Pereyra, D. Powell and G. Wojcik. "Nonlinear inversion of piezoelectric transducer impedance data", 1999, Technical Report, Weidlinger Associates, Los Altos, California.

[4] J. Czyzyk, M. Mesnier, and J. Moré. The NEOS server. IEEE Journal on Computational Science and Engineering, 5:68-75, 1998. See http://www-neos.mcs.anl.gov/.

[5] R. Fourer, D. Gay, and B. Kernighan. "AMPL: A Modeling Language for Mathematical Programming". Duxbury Press, Belmont, CA, 1993.

[6] M. Good. "Applications of CORBA in metacomputing environments", M.Sc. Dissertation, Electrical and Computer Engineering, Northwestern University.

[7] C.J. Lin and J.J. Moré "Newton's method for large bound-constrained optimization problems", SIAM J. Optim. 9 (1999), pp. 1100–1127.

[8] Message Passing Interface Forum. MPI. Available from http://www.mpi-forum.org.

[9] Microsoft. Microsoft COM, 1999. Available from http://www.microsoft.com/com

[10] J. Moré and S.J. Wright, "Optimization Software Guide", SIAM Publications, 1993.

[11] J.J. Moré. "A collection of nonlinear model problems", in Computational Solution of of Nonlinear Systems of Equations, vol. 26 of Lectures in Applied Mathematics, American Mathematical Society, (1990), pp.723–762.

[12] The Nexus Multithreaded Communication Library. Available from http://www.globus.org/nexus.

[13] J. Nocedal and S.J. Wright, "Numerical Optimization", Springer Verlag, New York, 1999.

[14] Object Management Group. CORBA. Available from http://www.omg.org.

[15] Sun Microsystems, "The Java Servlet Technology". Available from http://www.java.sun.com

[16] William J. Pardi. XML in Action, Web Technology. Microsoft Press 1999. ISBN 0-0756-0562-9

[17] C. Zhu, R.H. Byrd, P. Lu and J. Nocedal. "Algorithm 778:L-BFGS-B, Fortran subroutines for large scale bound constrained optimization", ACM Transactions on Mathematical Software, 23 (1997), pp. 550–560.