Scalable Libraries for Solving Systems of Nonlinear Equations and Unconstrained Minimization Problems

William D. Gropp*  Lois Curfman McInnes†  Barry F. Smith*

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439

Abstract

Developing portable and scalable software for the solution of large-scale optimization problems presents many challenges that traditional libraries do not adequately meet. Using object-oriented design in conjunction with other innovative techniques, we address these issues within the SNES (Scalable Nonlinear Equation Solvers) and SUMS (Scalable Unconstrained Minimization Solvers) packages, which are part of the multilevel PETSc (Portable, Extensible Tools for Scientific computation) library. This paper focuses on our design philosophy and its benefits in providing a uniform and versatile framework for developing optimization software and solving large-scale nonlinear problems. We also consider a three-dimensional anisotropic Ginzburg-Landau model as a representative application that exploits the packages' flexible interface with user-specified data structures and customized routines for function evaluation and preconditioning.

1 Challenges in Optimization Software Design

The solution of large-scale systems of nonlinear equations and unconstrained minimization problems pervades many areas of computational science and demands robust and flexible solution strategies. As surveyed by Moré and Wright [13], various software packages are currently available for solving such problems; however, their portability, versatility, and scalability are restricted within parallel environments.

The current generation of optimization software generally has a rigid form that imposes many limitations, even when considering uniprocessor architectures. In traditional software design, the expressions of algorithms make assumptions about the way mathematical objects, such as vectors and matrices, are represented by the computer. Thus, users are forced to convert from the natural representation of data for a particular application to one imposed by the software developer, often at the expense of considerable overhead. In addition, library routines are typically characterized by long and complicated calling sequences, with no consistent interface among different algorithms that solve a particular class of problems.

Library developers have attempted to overcome some of these difficulties by using reverse communication, in which the optimization algorithm returns to the calling routine for such tasks as function evaluations and the solution of linear systems. While reverse communication is quite flexible, this unnatural programming style makes the basic optimization algorithms difficult to understand, maintain, and extend. Also, serious inefficiency can result due to the continual transition through layers of routines.

These issues are magnified by the very nature of multiprocessor architectures, since robust and efficient implementation of mathematical abstractions involves the added considerations of parallel data structures and communication. An effective software package should exploit different parallel programming techniques for various phases of the solution process. For example, evaluating discretized functions on a mesh often requires nearest-neighbor communication, while linear solution methods typically require global communication. We must design the software accordingly.

*This work was supported in part by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.
†This work was supported by an appointment to the Distinguished Postdoctoral Research Program sponsored by the U.S. Department of Energy, Office of University and Science Education, and administered by the Oak Ridge Institute for Science and Education.
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
Since many application problems require the computational power of high-performance computers, a need clearly exists for a uniform and flexible framework for developing optimization software and solving application problems. Our goal is to use object-oriented and other software engineering techniques to create such an environment that serves both application programmers and optimization software developers. Recently Hohmann [7] and Meza [10] have employed object-oriented design for nonlinear optimization, although their work addresses only some of the issues we consider and is restricted to uniprocessor environments.

This paper focuses on our design philosophy and its benefits in developing robust and scalable solution strategies within the Scalable Nonlinear Equation Solvers (SNES) and Scalable Unconstrained Minimization Solvers (SUMS) packages. While our primary interest within this work is the solution of large-scale optimization problems, many of the issues that we address extend throughout the realm of mathematical software. In Section 2 we describe various aspects of the design philosophy, and in Section 3 we present examples of both basic and customized usage. In Section 4 we consider a three-dimensional anisotropic Ginzburg-Landau model as a representative application that exploits the software’s flexible interface with user-specified data structures and routines for scalable function, gradient, and Hessian evaluation. Finally, in Section 5 we discuss future issues in the development of optimization software.

2 Our Design Philosophy

We address key issues that promote the effectiveness and extensibility of numerical software in the SNES and SUMS packages, which are designed for both uniprocessor and multiprocessor architectures. This software, which is written in C and is fully usable from both C and Fortran 77, is part of the multi-level PETSc (Portable, Extensible Tools for Scientific computation) library [6], [1]. PETSc provides a hierarchy of modules that support the optimization packages, including flexible data structures for vectors and sparse matrices, direct and iterative linear solution techniques, preconditioning strategies, domain decomposition tools, communication routines, and performance monitors.

While many similarities exist in the approaches for solving systems of nonlinear equations and unconstrained minimization problems, the construction of two libraries, SNES and SUMS, enables us to exploit individual features of the solution strategies, and to communicate more effectively with the different user communities. Nevertheless, these libraries share a great deal of infrastructure, and from the high-level user’s perspective they differ only slightly. Newton-like methods, which are known for their efficiency and robustness, make up the core of the packages. In particular, several variants of line search strategies and trust region techniques are currently included. We note that our intent is not to duplicate the code in a package such as MINPACK-1 [12], but rather to create an interface that is more flexible and extensible. In fact, additional solution techniques from such packages can be easily added to the suites of solvers.

Unlike traditional optimization software, the SNES and SUMS packages are designed so that they are easily adaptable to the continually changing multiprocessor environments and the steadily advancing state of the art in computational linear algebra. We employ a hierarchical design of modular components, thereby enabling both software developers and application programmers to isolate the issues unique to the various levels. This design style facilitates experimentation with newly developed solution strategies and customization according to the user’s needs. This attribute is especially beneficial in the development of optimization software, since the effectiveness of the global solution process depends upon that of the lower-level components. For example, this approach enables the incorporation of new iterative techniques and parallel preconditioners, so that the methods for solving linearized systems can easily be updated as this field evolves.

2.1 Object-Oriented Programming

We use certain aspects of the object-oriented programming paradigm in combination with other innovative techniques to create the unique programming environment within SNES, SUMS, and the PETSc library as a whole. The componentware [16] concept, which involves data encapsulation to obtain maximum code reuse, encompasses the most useful features of object-oriented design for our needs. The PETSc design philosophy uses data and state encapsulation, abstract classes, and very limited inheritance to create a powerful foundation for software development. Different pieces of the PETSc modules can be mixed and matched, thus facilitating the development of componentware. This feature enables maximum code reuse, so that much less new code must be developed and debugged. This coding style enables us to increase code reuse, and hence productivity, within our small group
of software developers.

While neither SNES nor SUMS is restricted for use in conjunction with the other PETSc modules, the PETSc foundation allows the user to augment and customize the suites of nonlinear solution strategies, without being familiar with the internal structure of vectors, matrices, iterative routines, etc. This infrastructure enables the base algorithmic level of SNES and SUMS to expose mathematical details in a uniform fashion, without making unnecessary assumptions about the representation of mathematical objects. This data-structure-neutral approach [3] allows the natural storage formats for vectors and matrices to be dictated by the user’s application. Since issues regarding the selection of storage formats for parallel, sparse linear algebra are usually quite complicated, this feature is critical to the software’s performance.

2.2 Context Variables

Data and state encapsulation in the form of context variables is the key to the libraries’ organization. The nonlinear context, `NLCtx`, which is a data type defined by a C structure, contains virtually all information regarding the solution procedure, including data about the local state and option values. This information includes pointers to data structures and intermediate routines, as well as miscellaneous data and performance monitors. The `NLCtx` context is subdivided into a public part, which is identical for all solution techniques, and a private section that contains options, workspace, etc., which are unique for each particular method. The distinction between the two portions is invisible to the application programmer.

An optional user-defined context, containing information deemed necessary by the application programmer, can be embedded within the nonlinear context. For example, the user's context might contain information that enables parallel evaluation of the function, gradient, and Hessian matrix. Since the nonlinear context is incorporated throughout the SNES and SUMS packages, this user-defined data is automatically available throughout the entire suite of nonlinear solution routines.

The `NLCtx` context provides a simple yet powerful interface to SNES/SUMS. This context enables the user to access and manipulate its contents with basic routine calls, so that the user need not be familiar with the context's detailed structure. These routine calls allow the user to perform tasks such as selecting the desired solution strategy from the suite of nonlinear solvers, modifying parameters that control the solver's performance, replacing various phases of the solution process with custom routines, and monitoring the solver's progress. In addition, the `NLCtx` context provides a simplified interface to a variety of other PETSc modules for support, including sparse matrix data structures, preconditioners, and performance monitors.

2.3 Executable Templates

The flexible environment of SNES and SUMS fosters the development of customized, scalable solution methods. The libraries not only provide guidance for the user's selection of effective strategies, but also furnish underlying support for their implementation in the form of executable templates. Executable templates are source code written using high level abstractions so that the code can be easily understood and modified. Default routines can be employed for the various phases of the solvers, so that users can garner the full power of the libraries without concern for the underlying details of the techniques. Additional routines, which can serve as templates in the design of user-defined alternatives, are also provided.

This design enables the libraries to satisfy application programmers who desire complete support, ranging from algorithms through low level data structures. We also serve high-end programmers who desire to customize their own data structures and precisely tailor their solution strategies. In addition, the library structure is critical for software developers, who must be able to isolate a variety of software aspects, such as mathematical algorithms, data structures, numerical linear algebra, and interprocessor communication.

3 Using SNES and SUMS

The SNES and SUMS packages are intended for a wide range of users with varying needs and different levels of experience. Default solution procedures and data structures are provided, so that to solve a system of nonlinear equations, the user must simply supply routines to evaluate the residual and its corresponding Jacobian matrix. Likewise, to solve an unconstrained minimization problem, the user need only supply routines to evaluate the objective function, gradient, and Hessian matrix. Due to the modular design, the user can customize virtually any facet of the solution process according to specific needs and interests. We first illustrate basic usage of the packages, followed by an example of customization with executable templates. The users' manual for SNES and SUMS [2] provides additional details.
3.1 Basic Usage

Usage of the libraries involves the following stages for all solution methods:

- Create the nonlinear context.
- Establish the routines to evaluate the residual and Jacobian (SNES), or the objective function, gradient and Hessian (SUMS).
- Specify the step procedure for the global iterations.
- Set optional parameters, routines, and performance monitors.
- Select the desired vector operations and data structures.
- Set up the nonlinear solver.
- Execute the nonlinear solver.
- Destroy the nonlinear context.

The code fragment of Figure 1 illustrates these stages within the context of solving an unconstrained minimization problem with the multiprocessor SUMS package. Note that only trivial modifications are needed to convert this example to a uniprocessor version. In particular, routines within this example that have the form NXXX are identical within the uniprocessor and multiprocessor libraries. Within this example routines having the PNL prefix are unique to the parallel version and have uniprocessor analogues of the form NXXX.

The first command within the code fragment creates an optional user-defined context, which can contain any information deemed necessary by the application programmer. The next command, PNLCreate(), configures the chosen parallel nonlinear solver, which in this case is a trust region Newton method, and embeds the user context within the nonlinear context. Note that all of the different methods within SNES and SUMS employ an identical calling sequence. Optional arguments are passed by calling additional routines, which are ignored if the option is not appropriate. In this way any of the methods in the library may be used without altering the application code. Also, additional nonlinear solution methods may be added to the library without a need for any changes to the application codes.

We next establish the routines for evaluating the initial guess, function, and gradient, and Hessian matrix, and we specify the step procedure for the global

main(argc, argv)
int argc;
char **argv;
{
NLctx *nlP; /* context */
UserCtx *usrP; /* user context */
double *x; /* solution */
NLMethod m = NLN_MTR1; /* method type */
int gdim; /* global dimen. */
int ldim; /* local dimen. */
void Initial(); /* initial guess */
void Function(); /* function */
void Gradient(); /* gradient */
void Hessian(); /* Hessian */
/* Create optional user context */
usrP = NEW( UserCtx );
/* Create nonlinear context */
nlP = PNLCreate( m, usrP );
/* Set various routines */
NLSetInitialGuessRoutine( nlP, Initial );
NLSetFunctionRoutine( nlP, Function );
NLSetGradientRoutine( nlP, Gradient );
/* Set matrix and preconditioning info */
NLSetSlesSetRoutines( nlP, Hessian );
NLSetSpMatSetLocalRows( nlP, ldim );
NLSetSpMatSetCols( nlP, gdim );
/* Set optional parameters here */
NLSetXXX();
/* Set internal data structures */
PNLCreateDVectors( nlP, ldim );
NLSetUp( nlP );
/* Execute solution method */
its = NLSolve( nlP );
x = NLGetSolution( nlP );
/* Destroy nonlinear context */
NLDestroy( nlP );
}

Figure 1: Sample Multiprocessor Code Using the SUMS Package to Solve a Minimization Problem
iterations. As explained in the following section, the step procedure enables customization of the methods within the basic suite of nonlinear solvers. In this simple example, `PWLslesSetRoutines()` automatically sets the SUMS defaults for interfacing with the parallel preconditioning routines of the Simplified Linear Equation Solvers (SLES) [4] package. The SLES routines work with a variety of matrix storage formats provided within PETSc, thus freeing the user from the concern of manipulating matrix data. However, if users wish to provide their own matrix formats and operations, they are free to do so.

Next, we can determine various optional parameters, routines, and performance monitors, enabling the user to impose a great deal of customization to the solution process, even at this high level. We select parallel vector data structures with the command `PWLCreateDVectors()`, and we set the internal data structures for the nonlinear solver with `NLSetUp()`. Finally, we execute the solver with `NLSSolve()`, and free the data structures with `NLDestroy`.

This simple high-level interface provides application programmers with a wide range of options in terms of data structures, performance monitors, preconditioners, and iterative solvers. In fact, this high-level approach, with possibly a few customizations of the step procedure, should fully satisfy the needs of most application programmers.

### 3.2 Customized Usage

All of the nonlinear solvers are designed so that the user can tailor the solution process according to the application under consideration and the desired tools for parallel linear algebra. The Newton-like methods are subdivided into three basic phases of computation within each global step of the solution process:

- **Step-set-up:** (optional)
  This phase initializes the step computation, which typically includes forming the Jacobian (or Hessian) matrix and setting up the linear solver.

- **Step-compute:** (required)
  This phase computes the new step to try, and thus usually requires solving a linear system.

- **Step-destroy:** (optional)
  Typically, this phase involves freeing memory associated with the step computation. The user can also incorporate any problem-specific calculations required after a successful step computation.

In addition to providing default routines for these phases, we present several alternatives that can be used as templates in the design of custom routines.

The linear systems that arise throughout the global solution process can be solved by the SLES [4] package or by alternative routines chosen by the user. The SLES library provides easy access to a variety of direct and iterative linear solvers as well as preconditioners, so that the SNES-SLES and SUMS-SLES combinations facilitate experimentation with different linear solution techniques. As discussed in Section 2.3, defaults for each of these phases are provided in the form of executable templates, and even the most inexperienced users can adapt these options according to their needs.

Figure 2 demonstrates typical use of SLES within the step-set-up phase and can serve as a template for customized user routines. Within this phase, the user can easily select the precise means of calculating a step within the global solution process. For example, the user can specify different methods for solving the linearized Newton systems at each iteration. Also, the user can work directly with the Krylov Space (KSP) [5] iterative methods of the PETSc library, or can choose to provide alternative routines for such operations as the application of the inverse of the Jacobian matrix. In particular, as soon as a newly developed parallel linear solution technique is available, it can be easily incorporated into the package due to the modular design.

### 4 An Application Case Study: Superconductivity

The need for a robust and efficient solution strategy as well as the computational resources of high-performance computers is particularly evident for the problem of determining the minimum energy vortex solutions in high-temperature superconductors. The computational challenges presented by this application, which employs a full three-dimensional anisotropic Ginzburg-Landau model, include solving a very large, sparse system (with $O(10^6)$ degrees of freedom and $O(10^7)$ nonzeros in the Hessian matrix), which has local minima and saddle points. The SUMS package provides the appropriate software and a flexible interface with user-specified data structures and customized routines for function evaluation and preconditioning.

McInnes and Palumbo used the SUMS software to study the effect of anisotropy on the lower critical field
of a uniaxial superconductor [9]. We focus here on the characteristics of the problem, the solution strategy, the features of SUMS that proved to be especially useful, and the performance results. We refer the reader to [9] for details regarding the model and the computational results.

4.1 Computational Model

To study the magnetic properties of anisotropic materials, we seek the global minimizer of the anisotropic Ginzburg-Landau free energy functional. For the case of uniaxial superconductors, the total free energy over some volume \( \Omega \) is given by

\[
\mathcal{F}[\psi, A] = \int_{\Omega} (\mathcal{F}_{\text{cond}}[\psi] + \mathcal{F}_{\text{grad}}[\psi,A] + \mathcal{F}_{\text{field}}[A])d\Omega,
\]

where \( \psi \) is a complex-valued order parameter, and \( A \) is the magnetic vector potential. The three terms in the free energy are typically referred to as the condensation, gradient, and field energy terms, respectively. To simplify our work, we introduce an approximation to the free energy which makes the resulting system quasi-two-dimensional [14]. A noteworthy feature of the free energy functional is its invariance under a gauge transformation. This symmetry property implies that there is no unique solution to the anisotropic Ginzburg-Landau model, since any solution is known only up to a gauge transformation. Numerically this indicates the presence of a large null space, which presents a nontrivial obstacle in the minimization process.

Discretization of the domain \( \Omega \) with \( n_D \) grid points produces the unconstrained optimization problem

\[
\min \{ \mathcal{F}(x) : x \in \mathbb{R}^n \},
\]

where \( \mathcal{F} \) is the free energy functional, \( x \) consists of the magnetic vector potential and order parameter evaluated at the grid points, and \( n = 5n_D \) for the three-dimensional case. We use a regular domain with a uniform discretization. The parallel implementation employs a two-dimensional grid decomposition and a distributed sparse row scheme for matrix data. Thus, each processor stores the unknowns as well as the corresponding rows of the Hessian matrix that are associated with its grid points. The data-structure neutrality of the SUMS library is particularly useful for this case, since the matrix storage format can be specified by the user.

4.2 Solution Strategy

Newton-like methods have recently proven to be robust and efficient for solving such minimization prob-
lems arising within the context of superconductivity. We employ a sparse, parallel version of a trust region Newton method that uses the preconditioned conjugate gradient method to determine an approximate minimizer of the resulting quadratic function. This approach was first described by Steihaug [15], and a uniprocessor version is included within MINPACK-2 [11]. We use a new parallel, sparse implementation within the SUMS library that enables us to achieve scalable performance.

The trust region approach is based on the fact that Newton-like methods employ a quadratic model of the objective function around the current iterate, \( x_k \). Thus, the reduction \( F(x_k + s) - F(x_k) \) is modeled by

\[
q_k(s) = g(x_k)^T s + \frac{1}{2} s^T H_k s,
\]

where \( g(x_k) = \nabla F(x_k)^T \), and \( H_k = \nabla^2 F(x_k) \) is the symmetric Hessian matrix. Since the model is accurate only in a neighborhood of the current iterate (called the trust region), we determine the succeeding trial step, \( s_k \), to be the approximate minimizer of the quadratic constrained within the trust region. Thus, the resulting subproblem is to solve

\[
\min \{ q_k(s) : \| s \| \leq \Delta_k \},
\]

where \( \Delta_k \) is a positive parameter, and \( \| \cdot \| \) is the Euclidean norm in \( \mathbb{R}^n \). If \( F(x_k + s) \) is sufficiently reduced, then the step is accepted and \( \Delta_k \) can be increased; otherwise, the step is rejected and \( \Delta_k \) should be reduced.

We solve the minimization problem (1) inexactly using the preconditioned conjugate gradient method. In particular, we employ incomplete Cholesky factorization of the augmented Hessian matrix, as given by

\[
H_k + \gamma_k I \approx LL^T,
\]

where \( I \) denotes the identity matrix, and \( \gamma_k \) is a positive scalar that is computed at each iteration. This modification overcomes difficulties inherent in the superconductivity problem, since the Hessian, \( \nabla^2 F(x^*) \), is positive semidefinite at the solution.

The modular design of SUMS enables experimentation with newly developed solution strategies and customization according to the user's needs. For example, in solving the trust region subproblem, we easily incorporated the parallel incomplete Cholesky factorization approach of the BlockSolve [8] as a preconditioner. To achieve scalable solution, this approach involves reordering the preconditioning matrix based on a coloring of the graph associated with the matrix.

The use of such techniques are critical to the overall performance of the optimization software, since the efficiency of the global solution process is largely dominated by that of evaluating the function, gradient, and Hessian, and preconditioning the problem.

### 4.3 Performance Results

Table 1 presents performance results obtained on the Intel DELTA for the superconductivity problem. We demonstrate the scalability of the implementation by fixing the local domain size at 64x64 and varying the number of processors. Times are presented for the total solution process, where in each case the final gradient norm was less than 1.0e-8. We observe that the average performance per processor depends only slightly on the total number of processors. Key components in obtaining these results were the use of user-specified data structures and customized routines for function evaluation and preconditioning.

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Number of Newton Iterations</th>
<th>Solution Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>23</td>
<td>27.1</td>
</tr>
<tr>
<td>32</td>
<td>24</td>
<td>29.6</td>
</tr>
<tr>
<td>64</td>
<td>25</td>
<td>31.8</td>
</tr>
<tr>
<td>128</td>
<td>24</td>
<td>33.2</td>
</tr>
</tbody>
</table>

Table 1: CPU Solution times for a constant local domain size of 64x64.

### 5 Conclusion

As illustrated by the software design within SNES and SUMS, and more generally within the PETSc library as a whole, object-oriented programming techniques have proven to be extremely powerful in providing a framework for obtaining code reuse within single projects. However, code reuse between disparate groups with different coding philosophies is still quite limited and difficult to achieve. Development of the next generation of optimization software will require the combined efforts of many groups with different areas of expertise: optimization algorithms, parallel computing, software design, and application-specific understanding. Since the knowledge and software in each area are evolving rapidly, effectively using these tools is very different from using traditional static libraries like Linpack. The challenges presented by such coordination emphasize the need for using componentware technology.
References


