# An Overview Of The Advanced CompuTational Software (ACTS) Collection

L. A. Drummond and O. A. Marques
Lawrence Berkeley National Laboratory

The ACTS Collection brings together a number of general-purpose computational tools that were developed by independent research projects mostly funded and supported by the U.S. Department of Energy. These tools tackle a number of common computational issues found in many applications, mainly implementation of numerical algorithms, and support for code development, execution and optimization. In this article, we introduce the numerical tools in the collection and their functionalities, present a model for developing more complex computational applications on top of ACTS tools, and summarize applications that use these tools. Lastly, we present a vision of the ACTS project for deployment of the ACTS Collection by the computational sciences community.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed Programming*; *Parallel Programming*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Software Libraries*; *User Interfaces*; D.2.13 [**Software Engineering**]: Reusable Software; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.1.8 [**Numerical Analysis**]: Partial Differential Equations; G.4 [**Mathematical Software**]: —*Efficiency*; *Parallel and vector implementations*; *Reliability and robustness*; *User interfaces*; J.2 [**Physical Sciences And Engineering**]: —*Astronomy*; *Chemistry*; *Earth and atmospheric sciences*; *Physics*; *Electronics*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms: Algorithms, Design, Performance, Reliability

Additional Key Words and Phrases: Computational Sciences, High Performance Computing

## 1. MOTIVATION AND INTRODUCTION

As illustrated in Figure 1, large complex application codes have several *software constituents*. We use this term to identify a collection of subroutines, programs, sections of programs or combinations of these that implement a computational service. We classify these constituents according to their functionalities into three classes: algorithmic, data manipulation, and I/O implementations. In the first class, we find implementations of numerical algorithms ranging from basic linear algebra to complex iterative schemes in which one solves several systems of equations by

Lawrence Berkeley National Laboratory, One Cyclotron Road, MS 50F 1650, Berkeley, California 94720, USA. E-mail addresses: LADrummond@lbl.gov and OAMarques@lbl.gov

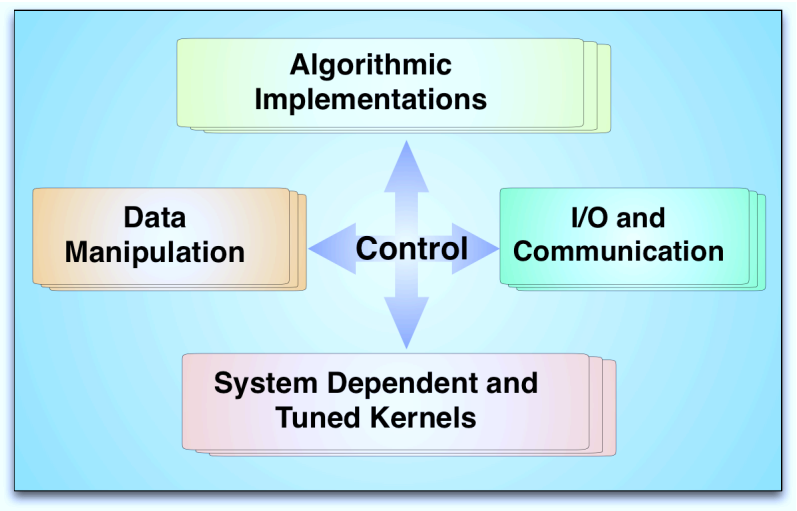


Fig. 1. **Main components of a complex computer application**

combining different methodologies. Examples of software constituents in this class are the widely used implementations of the Fast Fourier Transform (FFT), the Krylov subspace based and multigrid based algorithms, and finite element and volume discretization techniques. Data manipulation software constituents are those that implement a strategy for managing data in memory for processing or storage. Examples of software constituents in this class are the implementations of matrix storage format, parallel data distribution, and block partitioning techniques. Lastly, the I/O class includes implementations of software strategies that involve a transfer of data from a process' or task's memory to either disk or the memory of one or more other processes or tasks. Examples of these constituents are the sequential or parallel operations to write or read from disk, one-sided communication operations, and send and receive operations using a message passing system.

Nowadays, there are a good number of computational implementations of scientific and engineering applications that have followed the software development model in Figure 1. In this model, a developer or a group of developers realize all the different software constituents and create control procedures in a main program to link them. Frequently, the next step for application developers is to devise mechanisms to use more efficiently the often limited computational resources. This task reoccurs many times in the life of an application code and it is mostly driven by the changes in computational resources.

Application developers use combinations of compiler directives, language extensions, specialized library calls and even code rewrites to optimize the performance of their applications. Code optimization of this kind is referred as *system dependent and tuned kernels* in Figure 1. Overall, this part of the application development is operationally very expensive and with very short term impact on the application. Furthermore, the cost of optimizing a code render this kind of development almost

impractical for meeting today's and future's computational challenges in emerging collaborative sciences, in particular when this cost is summed to the costs of the initial code development and consecutive version upgrades. All these costs increase as the complexity of the application is also increased and the nature of high-end computing continues to evolve.

We advocate the use of existing robust, reliable and extendible software libraries for the development of scientific and engineering codes. Such libraries should provide high performance and scalability in a variety of computational systems. A basic model of this development approach is illustrated in Figure 2. In this approach the application developers try to reuse as much as possible existing software libraries or routines. In return, the application developers benefit from a faster code implementation plus portability and system tuned kernels for free. If the libraries are also scalable, reliable and robust, the application developers will also inherit these features into their application codes.

To realize code development following the model depicted in Figure 2, we need to create and maintain a collection of reliable, robust and high quality software libraries with an infrastructure that guarantees long-term support of the tools, provides independent tool evaluations to assess their quality, collects and disseminates relevant feedback on the tools among tool development projects, computer vendors, compiler developers and research groups in computational sciences.

The Advanced CompuTational Software (ACTS) Collection [Drummond and Marques 2002a; Marques and Drummond 2001] is a set of computational software tools that aim at simplifying the solution of common and important computational problems. The ACTS project is our effort to support users of ACTS tools beyond the individual tool development programs. To that end, the ACTS project aims to establish the infrastructure to facilitate the development of high-end computing applications and software libraries. The ACTS project differs from available software repositories by providing a high-level support to tool users during the tool selection and application development phases. Additionally, tools in the ACTS Collection are subject to independent tool evaluations to guarantee their functionality and maintain the high quality software standards of tools and different versions, while delivering fast and robust high performance solutions. Furthermore, the ACTS Collection is one of the results from software implementations of peer-reviewed research and successful scientific and engineering breakthroughs using these technologies. ACTS tools are available at no cost to the computational science community.

## 2. AN OVERVIEW OF THE ACTS COLLECTION

The Collection evolved from the former US Department of Energy (DOE) 2000 Project, which had two main components, the Advanced Computational Testing and Simulation Toolkit and the National Collaboratory Project. One of the goals of the project was to change the way scientists work together and address major challenges of scientific computation by developing and exploring new computational tools and libraries. Thus, most of the tools currently in the ACTS Collection were primarily developed at DOE laboratories, and in some cases in collaboration with

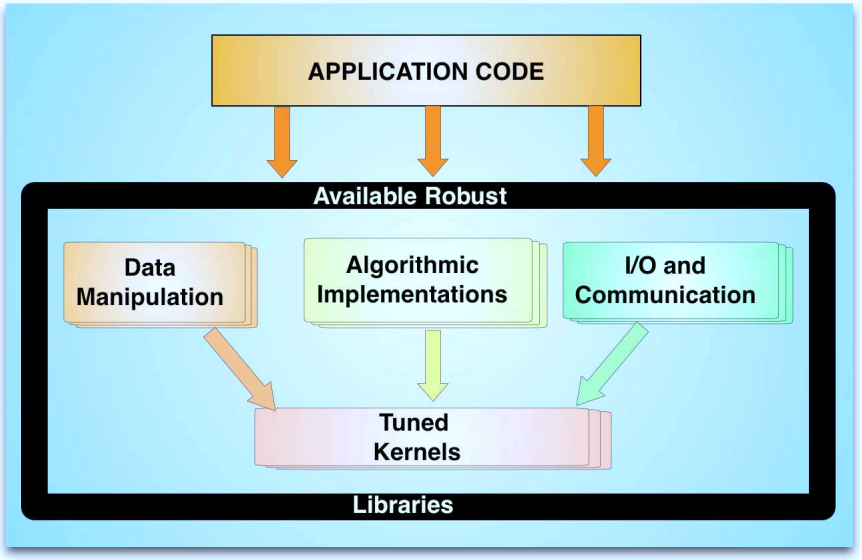


Fig. 2. **Development of complex computer application on top of optimized, scalable, reliable and robust software libraries**

universities. In addition, some tools have been co-funded by DOE and other agencies like the US National Science Foundation (NSF) and the US Defense Advanced Research Projects Agency (DARPA).

The ACTS project gathers years of uncoordinated software development and makes it available, at no direct costs, to a wide international community of scientists and engineers through the ACTS Collection. By no direct costs, we mean that the users do not pay for the use of the tools. However, users may have to devote resources when prototyping their application using the tools. ACTS tools are mostly open source and in some cases potential users are required to sign an agreement on the library use before downloading. Pointers and more information on how to download the tools can be found at the ACTS Information Center [Marques and Drummond 2001]. The ACTS project complements the library research and development efforts by adding technical support, quality assurance and outreach. The high-level technical support provided by the ACTS project ensures that development efforts are better employed. As a result of this effort, ACTS libraries have reached higher acceptance levels among computational scientists and institutions. In turn, this outreach provides the necessary means for individual tool projects to interact with more users, so the tools can gradually mature, becoming both more robust and portable to state-of-the-art high performance computing environments.

The ACTS project is also actively engaged in education activities, by organizing and participating in workshops, tutorials and other events related to computational sciences. Important lessons have emerged from these activities. These lessons are valuable to the computational science and engineering communities, and in particular to the software development and support projects. Here we summarize the

most important ones.
First, there is still a gap between tool developers and application developers which leads to duplication of efforts. Without projects like ACTS, application developers will continue to design and implement codes using techniques that are already available from other sources. In the worst case scenario, duplication of efforts cannot prevent software developers from falling into pitfalls or shortcomings that have resulted in software that is unresponsive to present and future computer simulation needs. Secondly, users demand long-term support of the tools, and have expressed concern on the longevity of support from tool developers and required evolution of the software as the hardware technology continues to evolve and the complexity of the scientific application continues to grow. The ACTS Project has begun to serve as a mechanism for the development, support, and promotion of quality high performance software tools. The ACTS project seeks to support users beyond the software development phase of the ACTS tools. This long-term involve maintaining a solid based of software tools and functionalities, helping users migrate to the different software versions and software supersedes.
The key component of this long-term user support is the coordination of efforts between software and hardware vendors, tool developers, users and ACTS. We have realized that the main parameters for software maturity are portability, robustness, acceptance, and long-term support. And it is in turn the interactions between application and tool developers that have made the software tools more mature, portable, robust and better documented. Therefore, we are working with commercial software developers and computer vendors to guarantee the long-term support for the tools and to effectively reach out more user communities. Our work with commercial software developers will result in software interfaces for their commercial packages to be able to call ACTS tools. For instance, some of the ACTS numerical tools now provide interfaces that allow them to be called from Matlab [Choy and Edelman 2002]. Lastly, our work with computer vendors will facilitate the porting of ACTS tools to emerging computer architectures.
As shown in the Appendix, the numerical tools in the ACTS Collection offer overlapping and complementary functionalities. We have learned that application developers demand a unified interface that will facilitate the porting of applications from one tool to another, and the interaction between the tools. Several of the ACTS development teams have been working to deliver fully interoperable interfaces. In these lines, the Common Component Architecture (CCA) [Armstrong et al. 1999] developments are leading towards more flexible interoperable environments. From our experience with the ACTS project, we have learned that education of graduate students and postdoctoral fellows provides a viable resource to build a bridge between computer scientists and domain scientists because there is an increasing demand in computational sciences and engineering for people with dual backgrounds in computation and a specific applied field of study. Within ACTS, we actively pursue the education of scientists and engineers in the use of the tools. As the students have become familiar with the technology, it has also become easier for their communities to accept, rely and use the technology at a faster rate than in the past. Thus, they are able to develop codes using state-of-the-art tools, and minimize the tool selection process, application development time and the time to

first solution.
Software libraries in the ACTS Collection fall in one of four categories: *Numerical Tools*, *Tools for Code Development*, *Tools for Code Execution* and *Tools for Library Development*. The tools are selected, considered for inclusion and evaluated using the tool's efficiency, scalability, reliability, portability, flexibility and ease-of-use as metrics to define the quality of the software ([Drummond and Marques 2002b]). Efficiency refers to the optimal use of computational resources in the system. Scalability is the ability to increase the number of processes and processors as the size and complexity of the problem being solved is also increased without compromising efficiency. Reliability refers to the failure free features of the library and proper handling of error bounds. Portability refers to the almost adaptability of the libraries to a wide variety of computational environments. Flexibility is the feature that allows users to construct new routines, libraries and codes from well defined tool modules. Therefore, the use of flexible software automatically breeds extensible software. Ease-of-use delivers interfaces that users outside the community of developers can adopt and become familiar with. All these goals make ACTS tools valuable assets for a wide spectrum of applications.
ACTS tools are targeted for distributed computing and use different mechanisms for interprocess communication.[1] One of the advantages of this model is that it facilitates the implementation of applications and solutions of problems on small clusters, and then porting to bigger systems when necessary, providing an attractive and cost-effective solution to many research groups.
In this article, we focus primarily on the numerical tools in the ACTS Collection and their functionalities. In this category we find software libraries that implement linear and nonlinear solvers for various types of systems of equations using direct and iterative techniques, eigenvalue solvers, and numerical optimization solvers. In addition to the numerical tools, we present a short description of ATLAS [Whaley et al. 2001], which is a library that belongs to the Library Development category but is included here because of its relevance to the numerical tools. ATLAS is widely used for automatic tuning of basic linear algebra kernels on different computational platforms.
Tables III-V (see Appendix) summarizes the functionalities provided by the numerical tools available in the ACTS Collection. In the following nine subsections we introduce the eight numerical tools currently in the Collection as well as ATLAS.

### 2.1 Hypre

This is a library for solving large, sparse linear systems of equations on massively parallel computers [Chow et al. 1998]. It has been developed by the Scalable Algorithms group of the Center of Applied Scientific Computing (CASC) at the Lawrence Livermore National Laboratory (LLNL). The main features of Hypre are scalable preconditioners, a suit of common iterative methods including conjugate gradient and GMRES for symmetric and unsymmetric matrices, respectively, intuitive grid-centric interfaces, and dynamic configuration of parameters. Hypre provides user-defined interfaces for multiple languages, and targets users with different level of expertise. As it will be discussed in Section 3, Hypre provides four

[1]Some of the tools are also available for sequential architectures.

different types of conceptual interfaces, which free up users from having to express their system of equations in a matrix, row-column, form. Thus, these interfaces provide a more natural way for expressing the system of equations being solved, while internally Hypre benefits from more efficient storage representations and computational kernels to deliver scalable performance.

### 2.2 OPT++

This library is intended for the efficient solution of nonlinear optimization problems [Meza 1994]. Its development started in 1992 at the Sandia National Laboratories (SNL) and its philosophy has been to provide a nonlinear optimization toolkit in which the problem is expressed in terms that the application user understands rather than forcing the application users to understand the mathematical concepts behind the algorithms. Additionally, OPT++ interfaces can easily accommodate changes to the algorithms and the components that are used in the solution of a problem previously set up with OPT++. OPT++ offers three types of nonlinear optimization solvers: direct search, conjugate gradient and newton-type methods. It targets four major classes of problems: problems with a basic non-linear function and without derivative information available, nonlinear function with first derivative information available, nonlinear function with approximated first derivative, and nonlinear function with first and second derivative information available. OPT++ handles boundary constraints, linear and nonlinear inequality constraints, and linear and nonlinear equations constraints.

### 2.3 PETSc

PETSc stands for Portable, Extensible Toolkit for Scientific computation and has been developed mainly by members of the Mathematics and Computer Science Division (MCS), at the at the Argonne National Laboratory (ANL) ([Balay et al. 1997; Balay et al. 2002; 2001]). This toolkit provides primarily a comprehensive set of tools to support the parallel numerical solution of PDEs that require solving large-scale, sparse linear and nonlinear systems of equations. Although its parallel implementation is based on MPI, PETSc users rarely need to invoke MPI because the communication is efficiently and transparently handled inside the PETSc routines. Therefore, PETSc's programming environment allows its users to focus more on the implementation details of their applications.

The basic elements of PETSc are indices, vectors and matrices. Indices are objects containing lists of natural numbers that are used for indexing matrices, vectors and arrays. Vectors are objects for storing data from right-hand side vectors, field variables, etc. Matrices are the objects for storing linear operators and PETSc supports a wide variety of matrix formats. For distributed vectors and matrices, PETSc distributes the data among the participating processes. These tasks are handled internally by the library but users can still perform data assignments and computations on these objects from their applications, independently of the internal and actual PETSc distribution.

PETSc's linear solvers interface is called KSP, which implements a variety of Krylov subspace methods and a family of preconditioning techniques to accelerate the convergence of Krylov based solvers. PETSc's non-linear solver interface is called SNES (for Scalable Nonlinear Equation Solver). In addition to KSP and SNES,

PETSc provides support for the iterative solution of time dependent partial differential equations of the form $U_t = F(U, U_x, U_{xx}, t)$ through the interface TS (for Time-Stepping solvers). In this case, the user provides a routine to evaluate $F$ and a routine to compute its Jacobian. As in the SNES interface, the user can alternatively apply a finite difference approximation to compute the Jacobian, or use PETSc's interface to ADIC ([Bischof et al. 1997] and ADIFOR [Bischof et al. 1992], which implement automatic differentiation strategies. Currently, PETSc supports Euler, backward Euler and pseudo-transient continuation time-stepping solvers, and it can interface to PVODE [Byrne and Hindmarsh 1998]. Lastly, PETSc provides a very elaborated set of tools for profiling, tracing and viewing information relevant to any of its objects, from a vector to a non-linear solver. This is a feature that greatly facilitates the fast prototyping of applications while providing great insight on the behavior of a PETSc object inside an application program.
PETSc has been built around the concept of extensibility and it has a growing number of application users and development projects that re-use some its functionalities. Examples of these are the software tools TAO [Benson et al. 2003] and SLEPc ([Hernández et al. 2003]) for the solution of optimization and eigenvalue problems, respectively.

### 2.4 ScaLAPACK

ScaLAPACK [Blackford et al. 1997] is a library of high-performance linear algebra routines for distributed-memory message-passing *Multiple Instruction Multiple Data* (MIMD) computers and networks of workstations. It is complementary to the LAPACK library [Anderson et al. 1999], which provides linear algebra routines for workstations, vector supercomputers, and shared-memory parallel computers. The ScaLAPACK library contains routines for solving systems of linear equations, least squares, eigenvalue problems and singular value problems. It also contains routines that handle computations related to those, such as matrix factorizations or estimation of condition numbers. We refer the reader to [Blackford et al. 1997] for a list of the functionalities currently provided by the library, implementation details, working notes and references.
The ScaLAPACK routines are based on block-partitioned algorithms in order to minimize the frequency of data movement. The fundamental building blocks of the library are the Basic Linear Algebra Subroutines (BLAS), distributed versions of those (PBLAS), and a set of Basic Linear Algebra Communication Subprograms (BLACS). The BLACS handles communication tasks that arise frequently in parallel linear algebra computations and supports MPI and PVM. In the ScaLAPACK routines, the majority of interprocessor communication occurs within the PBLAS, so the source code of the top software layer of ScaLAPACK looks similar to that of LAPACK.
The software design practices adopted by the ScaLAPACK team leads to routines that are efficient, reliable, scalable, portable, flexible and easy to use. Efficiency is achieved by means of optimize computation and communication engines, as well as block-partitioned algorithms (level 3 BLAS operations) for good node performance. Reliability is achieved by using well proved LAPACK algorithms and error bounds whenever possible. Scalability is achieved as the problem size and number of processors grow (LAPACK algorithms that did not scale well were replaced).

Portability is achieved by isolating machine dependencies to the BLAS and the BLACS. Flexibility is achieved by building upon a rich set of linear algebra tools (i.e. BLAS, BLACS and PBLAS). Finally, ease-of-use is achieved by defining subroutines calling interfaces similar to the ones available in LAPACK.
While an infrastructure for high level specification of parallel dense linear algebra algorithms has been proposed by the PLAPACK Project [Van de Geijn 1997], to date, ScaLAPACK provides the most comprehensive set of routines intended for parallel dense linear algebra calculations. The library has been ported to a great number of computer architectures, more recently to Linux clusters, has provided a solid infrastructure for the development of interactive supercomputer environments [Choy and Edelman 2002], as well as facilitated research on software development intended for computational grids environments [UTK 2000].

### 2.5 SUNDIALS

The SUite of Nonlinear and DIfferential/Algebraic equation Solvers (SUNDIALS), developed at LLNL, is a family of closely related numerical solvers for systems of ordinary differential equations (CVODE [Cohen and Hindmarsh 1994] and CVODES [Hindmarsh and Serban 2002]), nonlinear equations (KINSOL [Taylor and Hindmarsh 1998]), and differential-algebraic equations (IDA [Hindmarsh and Taylor 1999]). The different solvers in SUNDIALS share some common modules that contain the vector kernel, generic linear solvers, and are suitable for parallel and sequential computing environments.
CVODE contains methods for the solution of both stiff and nonstiff initial value problems. Integration methods include variable-coefficient forms of the Adams-Moulton and backward differentiation formula methods for serial computers and Krylov-based methods, GMRES in particular, for parallel environments. CVODES is a variant of CVODE with added features for sensitivity analysis, in case the user is concerned with computing the derivative of the solution or related quantities with respect to parameters appearing in the equations. Kinsol has been developed for systems of nonlinear equations and uses inexact Newton methods with linear-search strategies to accelerate the convergence. Lastly, IDA targets the solution of systems of differential-algebraic equations and uses backward differentiation formula methods.

### 2.6 SuperLU

SuperLU [Demmel et al. 2003] is a library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines. It was originally developed at the UC Berkeley Computer Science Division and at the National Energy Research Scientific Computing (NERSC) Center.
The library performs an LU decomposition with numerical pivoting and triangular system solves through forward and back substitution. The LU factorization routines can handle non-square matrices but the triangular solves are performed only for square matrices. The user may preorder the matrix before performing the factorization. The matrix columns may be preordered either through library or user supplied routines. SuperLU provides iterative refinement subroutines, in the working precision, to improve backward stability. SuperLU also provides routines

to equilibrate the system, estimate the condition number, calculate the relative backward error, and estimate error bounds for the refined solutions.
The factorization algorithm uses a graph reduction technique to reduce graph traversal time in the symbolic analysis, and data movement between levels of the memory hierarchy is reduced through loop ordering and the use of dense matrix operations in the numerical kernel. For the distributed memory implementation, a two-dimensional block cyclic matrix distribution is used to enhance scalability. SuperLU contains a collection of three related subroutine libraries: sequential SuperLU for uniprocessors, the multithreaded version SuperLU_MT for medium-size SMPs, and the MPI version SuperLU_DIST for large distributed memory machines. All these implementations are portable across many different platforms. SuperLU has been successfully used in the solution of large number of scientific and engineering problems.

### 2.7 TAO

The Toolkit for Advanced Optimization (TAO) has been developed at ANL/MCS. It supports the solution of large-scale optimization problems, including nonlinear least squares, unconstrained minimization, bound constrained optimization, and general nonlinear optimization. The implementation of the algorithms in TAO places a strong emphasis on software reusability, and it uses external robust services provided by other tools, like PETSc, where appropriate. TAO's design has been strongly motivated by the needs of many scientists who develop codes for high performance computing environments and in many cases work with legacy codes. TAO works in parallel and single processor environments. Its high-level programming interfaces allows the expression of complex problems with very few statements (TAO calls) and flexibility to test different parameters in the code or at execution time.

### 2.8 Trilinos

Trilinos is a project that targets the development of parallel solver algorithms and libraries within an object-oriented software framework. The target applications are from science and engineering that involve the solution of large-scale, complex multi-physics. Trilino's design respects the autonomy of its component software packages.[2] Thus, each of its packages is a self-contained, independent piece of software with its own set of requirements, development team and base of users. Additionally, Trilinos offers a set of tools to other tool development teams for integrating their packages to the software framework. These tools provide support for building installations in multiple computer platforms, generation of documentation and regression testing across a set of platforms. Trilinos is being primarily developed at Sandia National Laboratories. AztecOO is one of the package currently present in Trilinos. It superseded Aztec, which was originally included as a stand-alone package in the ACTS Collection.

[2]The Research and Development magazine awarded Trilinos one of the *2004 R&D Awards* for being one of the 100 most technological significant products of 2003.

### 2.9 Atlas

ATLAS (Automatically Tuned Linear Algebra Software) [Whaley et al. 2001], developed at the University of Tennessee, Knoxville, is a toolkit for the automatic generation of optimized numerical kernels for a variety of computer architectures and compilers. Atlas focuses on level three BLAS operations (matrix-matrix multiplications), and also a few routines from LAPACK that have high potential for optimization. As discussed in Section 1, the hand-optimization and hand-tuning of these kernels is generally very tedious and requires rigorous testing and results in many of the aforementioned drawbacks. The optimized libraries generated by ATLAS have been able to meet and even exceed the performance of the vendor supplied, hand-optimized BLAS, on a range of platforms.
ATLAS superseded PHiPAC, which was a related project developed primarily at the University of California, Berkeley. Both ATLAS and PHiPAC achieve performance through loop unrolling, explicit removal of unnecessary dependencies in code blocks, and use of machine sympathetic C constructs. Code generators are parameterized and scripts are used to find the optimal choice of parameters for a given architecture and compiler.

## 3. USING ACTS TOOLS

Before using any of the tools in the ACTS Collection, the user needs to identify the computational problem. For the identification of common problems and the available numerical tools in ACTS we have produced Tables III-V (see Appendix). Secondly, the user must determine whether the tool has been previously installed in his or her computational environment. If the tool has not been installed the user can download it from the developers site. Most of the numerical tools provide automatic configuration scripts that easily guide the users through the installation.
ACTS tools provide different types of interfaces. We start reviewing the most basic one and walk our way to more sophisticated interfaces. The basic interfaces form a set of routines that can be called by either C or Fortran programs. Some libraries like Hypre, PETSc, TAO, and OPT++ specifically provide support for C++. Other libraries can be called from C++ programs but they do not provide any specific object-oriented support. Some of the the tools that are not written in C++ use context variables to extend the semantics of the basic variable types in Fortran, C, and C++. The context contains relevant information attached to the behavior of a particular variable. For instance information about the solution of a linear system, matrix or iterative scheme. This information is particularly useful for the beginner to understand the behavior of the numerical solver or schemes being used.
A higher level of support is provided by Hypre, for example, through its four different interfaces. This approach aims at bringing a more natural view of the problem being prototyped and user friendlier interface. The four interfaces in Hypre are: structured-grid Interface, for applications with rectangular grids; semi-structure grid interface, for applications that involve block-structure, structure-AMR, and in general semi-structure grids; finite element interface, for finite element applications in which the grids are unstructured; and linear-algebraic interface, for applications with linear systems expressed in a sparse matrix format. Hypre users have to first

determine the conceptual interface that they are interested in developing, then choose among Hypre's preconditioners and iterative solvers, and lastly chose a matrix format that is compatible with their first two choices. As for other tools, users have to call a sequence of Hypre routines to properly setup their problem solutions. In many instances, application developers have to experiment with their choices of preconditioners, matrix representations, iteration schemes, stopping criteria, etc. This itself turns into an iterative process that requires code rewriting, running, verification and analysis of results. This suggests the need for a higher level of user interface that renders the initial prototyping more agile. PETSc offers a feature that allows the user to write a program with a generic or default object, like a generic choice of solver, matrix representation, etc., that will be specified at run time. This feature greatly speeds-up the process of selecting the right set of parameter to solve a particular problem. Since TAO and SLEPc have been built on top of PETSc, they also inherited this feature. Therefore, this is a good example of how to build robust, extensible, and reliable software on top of existing software with similar qualities.

We are currently working on the development of a Python [Lutz 2001] based user interface called PyACTS [Kang and Drummond 2003] to bring equally high-end interfaces to other tools in the ACTS Collection. PyACTS will not target high performance but rather intends to help the users familiarize themselves with the multiple functionalities available in the ACTS Collection.

## 4. HOW CAN ACTS WORK FOR YOU?

In this section we summarize some exemplar utilization of the tools in important scientific and engineering applications from an international pool of users. Tables I and II list these applications and showcase not only the benefits from the use of the numerical tools in the ACTS Collection, but also provide guidance on how the tools work in different application areas. We compile and maintain a more comprehensive list of these applications in *The ACTS Application Matrix and Performance* [Drummond et al. 2005] that is available on line [Drummond and Marques 2003]. The matrix is resourceful for all levels of users of the tools and even more for prospective users. User can find in this matrix a gallery of examples of the proper use of the functionality available in the collection and choices made by other application developers. In every case, we highlight the impact of the tool in the application's computational performance as well as in the algorithmic formulation and field contributions.

## 5. THE ACTS COLLECTION: ROAD MAP AND CONCLUSIONS

This introductory paper on the ACTS Collection discussed some of the main issues related to the deployment of a software infrastructure for the development of engineering and scientific simulation codes. Tools in the ACTS Collection are the products of peer-reviewed research published in computational science and engineering communities. These tools have been used inside many scientific and engineering applications addressing current grand challenges. ACTS tools brings a plethora of high quality and robust services to the hands of high-end computing users at no di-

| Application | Computational Problem | ACTS Functionality | Some Higlights |
|---|---|---|---|
| MADCAP [Borrill 1999] A Microwave Anisotropy Dataset Computational Analysis Package. From: NERSC. | Area: Cosmology. Determine power spectra of the Cosmic Microwave Background. | Uses ScaLAPACK to factorize and perform several subsequent triangular solves of a dense angular correlation matrix. | —Has been easily ported to many platforms.<br>—Near perfect scalability (IBM SP)<br>—Scientific Breakthrough, cover of Nature in 2000 |
| Simulation of Collisional Breakup in a Quantum System of Three Charged Particles. [Rescigno et al. 1999] From LLNL, LBNL, and Univ. of CA Davis. | Area: Quantum Chemistry. The complete solution of electron-impact ionization of hydrogen. Finite difference approximations, solving large, complex, nonsymmetric linear systems. | Uses SUPERLU to build preconditioners for a conjugate gradient squared algorithm. | —Solves linear systems of order 8.4 millions.<br>—Realized computations that were not possible before.<br>—Scientific Breakthrough made the Cover of Science in 1999. |
| FLAPW [Canning et al. 2000] Parallel Full-potential Linearized Augmented Plane-Wave. From: LBNL/NERSC. | Area: Material Sciences. Electronic structure calculations based on solving Schrdinger's equation. | Uses ScaLAPACK to diagonalize a Hamiltonian matrix, finding the lowest 5-10% eigenvalues and eigenfunctions that correspond to wave functions and energies of the electrons. | —Code has been ported to many platforms.<br>—Most accurate and heavily used in materials science.<br>—Gordon Bell Prize 1998.<br>—Allowed simulation containing 700 atoms. |
| FUN3D [Anderson et al. 2003] Computing turbulent flows on unstructured grids. From: NASA-Langley. | Area: CFD. Solution of compressible and incompressible Euler and Navier-Stokes equations. | Uses PETSc to parallelize solvers [Keyes et al. 1997] | —Won the Gordon Bell Prize. 1999<br>—Example of a legacy code promptly parallelized. |
| NIMROD [Sovinec et al. 2004] Nonlinear phenomena in fusion reactor plasmas. From OFES, LANL, U. of WI, Utah State, CO Univ. Bolder, SAIC, UCLA, SNL, GA, and NASA-JSC | Area: High Energy Physics. High-order finite element representation of the poloidal plane and a finite Fourier series representation of the toroidal direction. | Uses SUPERLU to replace legacy code based on preconditioned conjugate gradient (PCG) solver. | —The single processor code that uses SUPERLU is a 100 times faster than the PCG one.<br>—The improved code performance is of order 5-fold, which equates to 3-5 years progress hardware. |

Table I. Example of Scientific and Engineering Applications that use ACTS. The new institutional acronyms used in this table are LBNL for Lawrence Berkeley National Laboratory, OFES for the Office of Fusion Energy Sciences, SAIC for Science Applications International Corporation, GA for General Atomics, and NASA-JSC for NASA Johnson Space Center.

| Application | Computational Problem | ACTS Functionality | Some Higlights |
|---|---|---|---|
| M3D [Park et al. 1999] Multilevel, 3D, parallel, plasma simulation code. From Princeton Plasma Physics Laboratory | Area High Energy Physics. Nonlinear calculations of plasmas in toroidal topologies including Tokamaks and Stellarators | Uses PETSc for the manipulation of unstructured meshed problems and solution of linear systems of equations. | —Reduced development time.<br>—From PETSc, easily test different preconditioners available in PETSc and Hypre. |
| CAVIRES [Roman 2002] Electromagnetic analysis of cavities. From University Polytechnic of Valencia (UPV). | Area: High Energy Physics. Generalized algebraic eigenvalue problem. Solving Maxwell eqs. in a bounded volume. | Uses SLEPc to compute the eigenvectors corresponding to a few of the smallest positive nonzero eigenvalues, | —Scalable and Portable code.<br>—Use in addressing problems in the order of millions. |
| Time-dependent Neutron Diffusion Equation [García et al. 2004] Fast transient analysis code. From UPV. | Area: Nuclear Physics. Large system of linear stiff Ordinary Differential Equations (ODE). | Uses SUNDIALS to solve the ODEs. | —Fast and efficient code development.<br>—Superlinear speedups. |
| Model Reduction [Guerrero et al. 2002]. From UPV. | Area: Control Problems. Reduction of a linear control system model. | Uses ScaLAPACK to implement parallel model reduction methods based on balancing techniques. | —Code included in the parallel SLICOT [Benner et al. 1999].<br>—Scales to a moderate number of processes. |
| Omega3P [Sun 2003] modeling and analysis of accelerator cavities. From Stanford Linear Accelerator. | Area: Nuclear Physics. Calculates cavity mode frequencies and field vectors by solving a generalized eigenvalue problem from finite element discretization of Maxwell's equations. | Uses SuperLU and PARPACK [Lehoucq et al. 1998] to implement a parallel exact shift-invert eigensolver. | —Previous code had poor convergence and problem size could not be scaled.<br>—Has been easily ported to many platforms.<br>—Solves problem of order 7.5 million. |
| Analysis of Lambda Modes for safety of nuclear reactors [Hernandez et al. 2003] From UPV. | Area: Nuclear Physics. Differential eigenvalue problem derived from the Neutron Diffusion equation. The problem is discretized by means of a collocation method. | Uses SLEPc for solving the eigenvalue problem combined with PETSc for solving the linear system. | —Several benchmark reactors have been used for validation, typical problem sizes ranging from 20,000 to 500,000.<br>—Scalable code |

Table II. Example of Scientific and Engineering Applications that use ACTS (cont.)

rect cost. The ACTS project implements a viable solution to bring all these efforts to the computational science community while exploring mechanisms for extending the longevity of the software beyond their development phase. Thus, the ACTS project is cardinal for maintaining a high quality software collection by not only attesting the quality of tools but also ensuring that users select the more suitable tools and tool functionalities, and make proper use of these. Clearly, this higher level support differentiates the ACTS project from other main software repositories. In Figure 3, we show how we envision the multiple interactions and roles played by the ACTS project within the computational science community. the ACTS project has been working with a basic set of reliable tools and is paying close attention to tools being developed by other initiatives in order to incorporate them into the ACTS Collection. In the figure, the tools to be tested and eventually accepted into ACTS are produced by developers working with and in the *User Community* and in the ACTS project. We plan to continue expanding the current infrastructure and bringing new software development into ACTS. The expanded infrastructure for the ACTS project is able to accumulate expertise from tool developers, users and application scientists and produce relevant feedback and knowledge to be disseminated inside the computational sciences community.

As depicted in Figure 3, the strong components of the ACTS Collection and its infrastructure are the unique coordination of high-level support to the *User Community*, the independent *Testing and Acceptance* of software tools and the the constant interactions with *Scientific Computer Centers* and *Computer Vendors*. The high level support is geared towards minimizing the application development time, from the first prototype code to the production code. The testing and acceptance of new tools into the ACTS Collection ensures the tool quality and expansion of available functionality. In the inclusion of tools to the collection, *Interoperability* is playing a major role to guarantee software reusability, incremental development, and a ready-to-use wide variety of services to the end users. Further, the combination of high-level tool support and the collection of high quality software tools in all optimizes the computational production time of the application. In addition, securing the long-term support for the tools and its portability to today's and tomorrow's computational platforms also secures the life of the computational applications that use these tools. ACTS also plays an active role at keeping application developers abreast with the tool developments, evolutions and innovations.

The organization of workshops, the design and maintenance of educational tools, such as the ACTS Information Center and PyACTS have ensured the visibility of the tools in the computational science community. This effort continues to promote software reusability, over duplication of efforts, while set higher standards for high performance computing software. Within ACTS we have learned that the exposure of the tools to a wide variety of applications and contexts has not only made the tools more mature and widely accepted but also has brought up new challenges and research directions in computational sciences.

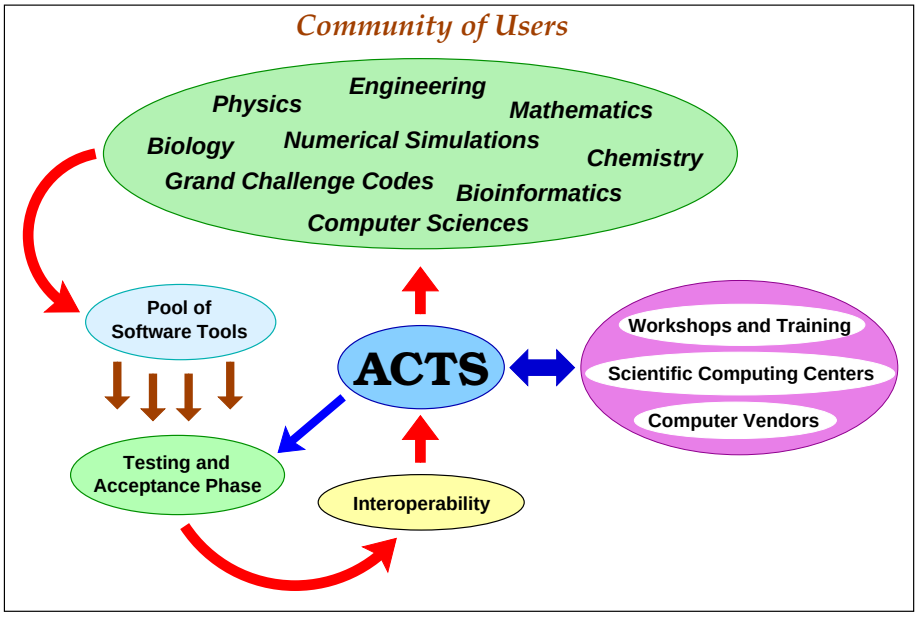


Fig. 3. **ACTS interactions within the computational science community**

ACKNOWLEDGMENT

This work was supported by the Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract No. DE-AC03-76SF00098.

REFERENCES

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J. W., Dongarra, J. J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. C. 1999. *LAPACK User's Guide*, third ed. SIAM, Philadelphia, Pennsylvania.

Anderson, W. L., Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F. 1999. Achieving high sustained performance in an unstructure mesh CFD application. In *Proceedings from SC'99*. also available as Argonne preprint ANL/MCS-P776-0899.

Anderson, W. L., Nielsen, E., Park, M., and Lee-Rausch, B. 2003. FUN2D/3D Webpage. http://fun3d.larc.nasa.gov/index.html.

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., and Smolinski, B. 1999. Toward a common component architecture for high-performance scientific computing. In *Proceedings of the Eighth International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press, Los Alamitos, CA, 115–124.

Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M., McInnes, L. C., Smith, B. F., and Zhang, H. 2001. PETSc home page. http://www.mcs.anl.gov/petsc.

Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M., McInnes, L. C., Smith, B. F., and Zhang, H. 2002. PETSc users manual. Tech. Rep. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory.

Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F. 1997. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhauser Press, 163–202.

Benner, P., Mehrmann, V., Sima, V., Huffel, S. V., and Varga, A. 1999. SLICOT: A subroutine library in systems and control theory. In *Applied and Computational Control, Signal and Circuits*, B. N. Datta, Ed. Vol. 1. Birkhauser, 499–536.

Benson, S., McInnes, L. C., Moré, J. J., and Sarich, J. 2003. TAO User Manual. Tech. Rep. ANL/MCS-TM-242-Revision 15, Argonne National Laboratory. Jan.

Bischof, C., Carle, A., Corliss, G., Griewank, A., and Hovland, P. 1992. Adifor - generating derivative codes from fortran programs. *Scientific Programming 1*, 1–29.

Bischof, C., Roh, L., and Mauer-Oats, A. J. 1997. Adic: an extensible automatic differentiation tool for ansi-c. *Software: Practice and Experience 27*, 1427–1456.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J. W., Dhillon, I., Dongarra, J. J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1997. *ScaLAPACK User's Guide*. SIAM, Philadelphia, Pennsylvania.

Borrill, J. 1999. MADCAP: the microwave anisotropy dataset computational analysis package. Available at the Los Alamos e-Print ArXiv: http://xxx.lanl.gov/ps/astro-ph/9911389.

Byrne, G. D. and Hindmarsh, A. C. 1998. User documentation for PVODE, an ODE solver for parallel computers. Tech. Rep. UCRL-ID-130884, Lawrence Livermore National Laboratory. May.

Canning, A., Mannstadt, W., and Freeman, A. J. 2000. Parallelization of the FLAPW method. *Computer Physics Communications 130*, 233–243.

Chow, E., Cleary, A. J., and Falgout., R. D. 1998. Design of the Hypre preconditioner library. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, M. Henderson, C. Anderson, and S. Lyons, Eds.

Choy, R. and Edelman, A. 2002. MATLAB*P 2.0: Interactive supercomputing made practical. Ph.D. thesis, MIT.

Cohen, S. D. and Hindmarsh, A. C. 1994. CVODE User Guide. Tech. Rep. UCRL-MA-118618, Lawrence Livermore National Laboratory. Oct.

Demmel, J. W., Gilbert, J. R., and Li, X. 2003. *SuperLU User's Guide*. University of California, Berkeley.

Drummond, L., Hernandez, V., Marques, O., Roman, J., and Vidal, V. 2005. A Survey of High-Quality Computational Libraries and Their Impact in Science and Engineering Applications. In *Lecture Notes in Computer Science*. Vol. 3403. Springer-Verlag, Valencia, Spain, 37–50.

Drummond, L. and Marques, O. 2003. The ACTS Application and Performance Matrix. http://acts.nersc.gov/MatApps.

Drummond, L. A. and Marques, O. 2002a. ACTS: A collection of high performance software tools for scientific computing. In *Proceedings from the Tenth ECMWF Workshop on the Use of HPC in Meteorology*, W. Zwieflhofer and N. Kreitz, Eds. World Scientific Publishing, Reading, UK.

Drummond, L. A. and Marques, O. 2002b. The ACTS Collection robust and High-Performance Tools for Scientific Computing: Guidelines for tool inclusion and retirement. Tech. Rep. LBNL/PUB-3175, Lawrence Berkeley National Laboratory. Nov.

García, V. M., Vidal, V., Verdu, G., Garayoa, J., and Miro, R. 2004. Parallel resolution of the two-group time-dependent neutron difussion equation with public domain ODE codes. Submitted to 6th International Meeting on High Performance Computing for Computational Science.

Guerrero, D., Hernandez, V., and Roman, J. E. 2002. Parallel SLICOT model reduction routines: The Cholesky factor of Grammians. In *15th Triennal IFAC World Congress*. Barcelona, Spain.

Hernández, V., Romá, J. E., and Vidal, V. 2003. SLEPc User's Manual: Scalable Library for Eigenvalue Problem Computations. Tech. Rep. DSIC-II/24/02, Universidad Politecnica de Valencia. May.

Hernandez, V., Roman, J. E., Vidal, V., Verdu, G., and Ginestar, D. 2003. Resolution of the neutron diffusion equation with SLEPc, the Scalable Library for Eigenvalue Problem

Computations. In *Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew*. American Nuclear Society, Gatlinburg, Tennessee.

Hindmarsh, A. C. and Serban, R. 2002. User Documentation for CVODES, An ODE Solver with Sensitivity Analysis Capabilities. Tech. Rep. UCRL-MA-148813, Lawrence Livermore National Laboratory. July.

Hindmarsh, A. C. and Taylor, A. G. 1999. User Documentation for IDA, a Differential-Algebraic Equation Solver for Sequential and Parallel Computers. Tech. Rep. UCRL-MA-136910, Lawrence Livermore National Laboratory. Dec.

Kang, N. and Drummond, L. A. 2003. A first prototype of PyACTS. Tech. Rep. LBNL-53849, Lawrence Berkeley National Laboratory.

Keyes, D. E., Kaushik, D. K., Smith, B. F., and Anderson, K. 1997. Porting FUN3D to distributed memory parallelism. *Parallel Computing Research 5,* 4. CRPC.

Lehoucq, R. B., Sorensen, D. C., and Yang, C. 1998. *ARPACK User's Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. SIAM, Philadelphia, PA.

Lutz, M. 2001. *Programming Python*, second ed. O'Reilly & Associates, Inc., Sebastopol, CA.

Marques, O. A. and Drummond, L. A. 2001. The ACTS Information Center. http://acts.nersc.gov.

Meza, J. C. 1994. OPT++: An Object Oriented Class Library for Non-Linear Optimization. Tech. Rep. 94-8225, Sandia National Laboratories.

Park, W., Belova, E. V., Fu, G. Y., Tang, X., Strauss, H. R., and Sugiyama, L. E. 1999. Plasma simulation studies using multilevel physics models. *Physics of Plasmas 6,* 5, 1796–1803.

Rescigno, T., Baertschy, M., Isaacs, W., and McCurdy, W. 1999. Collisional breakup in a quantum system of three charged particles. *Science 286*, 2474–2479.

Roman, J. E. 2002. Software portable, escalable y extensible para la resolución de problemas de valores propios dispersos de gran dimensión. Ph.D. thesis, Universidad Politécnica de Valencia.

Sovinec, C. R., Barnes, D. C., Gianakon, T. A., Glasser, A. H., Nebel, R. A., Kruger, S. E., Schnack, D. D., Plimpton, S. J., Tarditi, A., Chu, M. S., and the NIMROD Team. 2004. Nonlinear Magnetohydrodynamics Simulations with High-order Finite Elements. *Journal of Computational Physics 195*, 355–386.

Sun, Y. 2003. The filter algorithm for solving large-scale eigenproblems from accelerator simulations. Ph.D. thesis, Stanford University.

Taylor, A. G. and Hindmarsh, A. C. 1998. User Documentation for KINSOL, A nonlinear solver for sequential and parallel computers. Tech. Rep. UCRL-ID-131185, Lawrence Livermore National Laboratory. July.

UTK. 2000. The Grid Application Development Software (GrADS) project. http://icl.cs.utk.edu/grads.

Van de Geijn, R. A. 1997. *Using PLAPACK*. The MIT Press.

Whaley, R. C., Petitet, A., and Dongarra, J. 2001. Automated empirical optimizations of software and the atlas project. *Parallel Computing 27,* 1-2, 3–25.

Appendix - ACTS Collection: Numerical Tools and their Functionalities

| **Computational Problem** | **Methodology** | **Algorithms** | **Library** |
|---|---|---|---|
| **Linear Equations** | Direct Methods | $LU$ Factorization | ScaLAPACK (dense)<br>SuperLU (sparse) |
| | | Cholesky fact. | ScaLAPACK |
| | | $LDL^T$ (Tridiag A) | ScaLAPACK |
| | | QR Factorization | ScaLAPACK |
| | | $QR$ factorization | ScaLAPACK |
| | | $QR$ + Col Pivoting | ScaLAPACK |
| | | $LQ$ factorization | ScaLAPACK |
| | | Full Orthogonal factorization | ScaLAPACK |
| | | Generalized $QR$ factorization | ScaLAPACK |
| | Iterative Methods | Conjugate Gradient (CG) | AztecOO (Trilinos)<br>PETSc |
| | | GMRES | AztecOO<br>Hypre<br>PETSc |
| | | CG Squared | AztecOO<br>PETSc |
| | | Bi-CG-Stab | AztecOO<br>PETSc |
| | | QMR | AztecOO |
| | | Transpose Free QMR | AztecOO<br>PETSc |
| | | SYMMLQ | PETSc |
| | | Preconditioned CG | AztecOO<br>Hypre<br>PETSc |
| | | Richardson | PETSc |
| | | Block Jacobi preconditioner | AztecOO<br>Hypre<br>PETSc |
| | | Point Jacobi preconditioner | AztecOO |
| | | Least-squares polynomials | AztecOO |
| | | SOR precond. | PETSc |
| | | Overlapping ASM precond. | PETSc |
| | | Approx. Inverse | Hypre |
| | | Sparse $LU$ preconditioner | AztecOO<br>Hypre<br>PETSc |
| | | Incomplete LU preconditioner | AztecOO<br>Hypre<br>PETSc |

Table III. Summary of the numerical functionalities currently available in the ACTS Collection

| Computational Problem | Methodology | Algorithms | Library |
|---|---|---|---|
| **Linear Equations** | Multigrid (MG) | MG preconditioner | Hypre<br>PETSc |
| | | Algebraic Multigrid | ML (Trilinos)<br>Hypre |
| | | Semicoarsening | Hypre |
| **Linear Least Squares** | Least Squares | $min_x \|b - Ax\|_2$ | ScaLAPACK |
| | Minimum norm | $min_x \|x\|_2$ | ScaLAPACK |
| | Minimum norm least squares | $min_x \|x\|_2$ and $min_x \|b - Ax\|_2$ | ScaLAPACK |
| **Standard Eigenvalue** | Symmetric Eigenvalue | $Az = \lambda z$ for $A = A^T$ or $A = A^H$ | ScaLAPACK (dense)<br>SLEPc(sparse) |
| **Singular Value** | Singular Value Decomposition | $A = U\Sigma V^T$<br>$A = U\Sigma V^H$ | ScaLAPACK (dense)<br>SLEPc (sparse) |
| **Generalized Symmetric Definite Eigenproblem** | Eigenproblem | $Az = \lambda Bz$,<br>$ABz = \lambda z$,<br>$BAz = \lambda z$ | ScaLAPACK (dense)<br>SLEPc (sparse) |
| **Non-linear Equations Problems** | Newton-based | Line Search | PETSc<br>KINSOL (SUNDIALS) |
| | | Trust Regions | PETSc |
| | | Pseudo-transient continuation | PETSc |
| | | Matrix free | PETSc |
| **Non-linear Optimization** | Newton-based | Newton | OPT++<br>TAO |
| | | Finite Differences | OPT++ |
| | | Quasi Newton | OPT++<br>TAO (LMVM) |
| | | Nonlinear Interior Point | OPT++<br>TAO |
| | CG | Standard nonlinear CG | OPT++<br>TAO |
| | | Limited memory BFGS | OPT++ |
| | | Gradient Projection | TAO |
| | Direct Search | Without derivative information | OPT++ |
| | Semismooth | Infeasible semismooth | TAO |
| | | Feasible semismooth | TAO |
| **Ordinary differential equations (ODEs)** | Integration | Variable coefficient Adams-Moulton | CVODE (SUNDIALS) |
| | Backward Differential | Direct Solvers | CVODE |
| | | Iterative Solvers | CVODE |
| **ODEs with Sensitivity Analysis** | Integration | Variable coefficient Adams-Moulton | CVODES (SUNDIALS) |
| | Backward Differential | Direct Solvers | CVODES |
| | | Iterative Solvers | CVODES |

Table IV. Summary of the numerical functionalities currently available in the ACTS Collection (continued)

| Computational Problem | Methodology | Algorithms | Library |
|---|---|---|---|
| **Differential Algebraic Equations** | Backward Differential Formula | Direct solvers | IDA (SUNDIALS) |
| | | Iterative solvers | IDA |
| **Nonlinear Equations + Sensitivity Analysis** | Inexact Newton | Line search | SensKINSOL (SUNDIALS) |
| **Tuning and Optimization** | Automatic code generators and compilations | BLAS and some LAPACK routines | ATLAS |

Table V. Summary of the numerical functionalities currently available in the ACTS Collection (continued)