SimTK 1.0 Requirements: Numerical Methods

Version 1.0

Jack Middleton

August 22, 2005

Abstract

This document discusses the requirements for incorporating numerical methods into SimTK 1.0. SimTK 1.0 will have an initial set of numerical methods which will be expanded upon in future releases of SimTK. Users will be able to download and install this initial set and access these methods through a flexible API which allows applications to switch between different methods.

1 Purpose of this document

This document describes the requirements and rationale for incorporating numerical algorithms into SimTK. Its goal is to communicate and develop a consensus regarding our plans in this area for SimTK 1.0, and to guide the development of software which implements these plans.

2 What are Numerical Methods?

In this document numerical methods refer to software implementations of a core set numerical algorithms which perform many of the common, computationally intensive tasks used in physics based simulation of biological systems. The goals of numerical methods are to provide fast, accurate, approximate solutions to problems and make effective use of the available hardware. The types of problems which are solved by these numerical methods include nonlinear optimization, linear algebra, ordinary differential equations (ODE's), partial differential equations (PDE's), multibody dynamics and finding roots of nonlinear equations.

3 How are Numerical Methods Relevant to Simbios?

Numerical methods form the foundation for many of the software tools used by the Simbios community. All four of the current Simbios Driving Biological Problems (DBP's) heavily use numerical methods as part of their research. For example, the neuromuscular DBP often use optimization, ODE's, multibody dynamics,

linear algebra and root finding numerical methods. The myosin/actin DBP uses multibody dynamics and linear algebra. The RNA DBP uses multibody dynamics, linear algebra, ODE's and optimization. The cardiovascular DBP uses PDE's, and linear algebra. Often the characteristics of the numerical method in terms of accuracy, stability or computation time are limiting factors for these DBP. It is therefore critical that SimTK supply a solid set of numerical methods that are fast, stable, accurate and easy to use.

Numerical methods are also important for researchers doing research into new numerical algorithms. SimTK needs to provide a framework for which allows these researchers to evaluate their new algorithms against the current state of the art numerical methods.

4 Which Numerical Methods will be in SimTK?

Because of the diversity of the types of problems encountered in physics based simulation of biological systems, SimTK will need to support a wide range of numerical methods. Also, because some algorithms work better on some datasets than others, SimTK will also need to offer different algorithms which solve the same type of problems. For example, SimTK needs to provide ODE solvers that are designed to solve stiff systems and solvers that are designed to solve non stiff systems. As a minimum, SimTK must solve linear algebra problems, ordinary differential equations, optimization problems, find roots, partial differential equations, and multibody dynamics. However, it not possible to support all of these directly or as SimTK core offerings in the SimTK 1.0 time frame. Therefore, SimTK 1.0 will only support linear algebra, ordinary differential equations and multibody dynamics. Root finding would be nice to have, and will be included if time permits, but will not be considered as a requirement for SimTK 1.0. Partial differential equations, and optimization will be analyzed to determine if they have any requirements which need to be addressed in SimTK 1.0 so that they can be included in a later release of SimTK.

One of the goals of SimTK is to provide easy access to proven, high quality implementations of numerical methods for the Simbios community. Initially we will focus on including existing industrial strength software packages that have a proven track record of high quality and performance. We believe that CVODES for solving ODE’s, LAPACK for solving linear algebra problems and Simbody for multibody dynamics meet these criteria. These packages will be included in SimTK 1.0 and will provide a solid platform for SimTK users as well as a benchmark for those who are developing new software algorithms to compare against. These will also serve as prototypical numerical methods to help solidify our general approach to including numerical methods in SimTK.

5 Numerical Methods Application Interface (API)

SimTK will give applications the choice of accessing numerical methods through several different levels of abstraction which are show in Figure 1 below. The lower level, more concrete interfaces are shown towards the bottom with the higher level, more abstract interfaces towards the top. At the lowest level are conventional numeric libraries such as LAPACK and CVODES which perform the actual computations. Applications can access the functionality of the numerical libraries through any of the interface levels which are explained below.

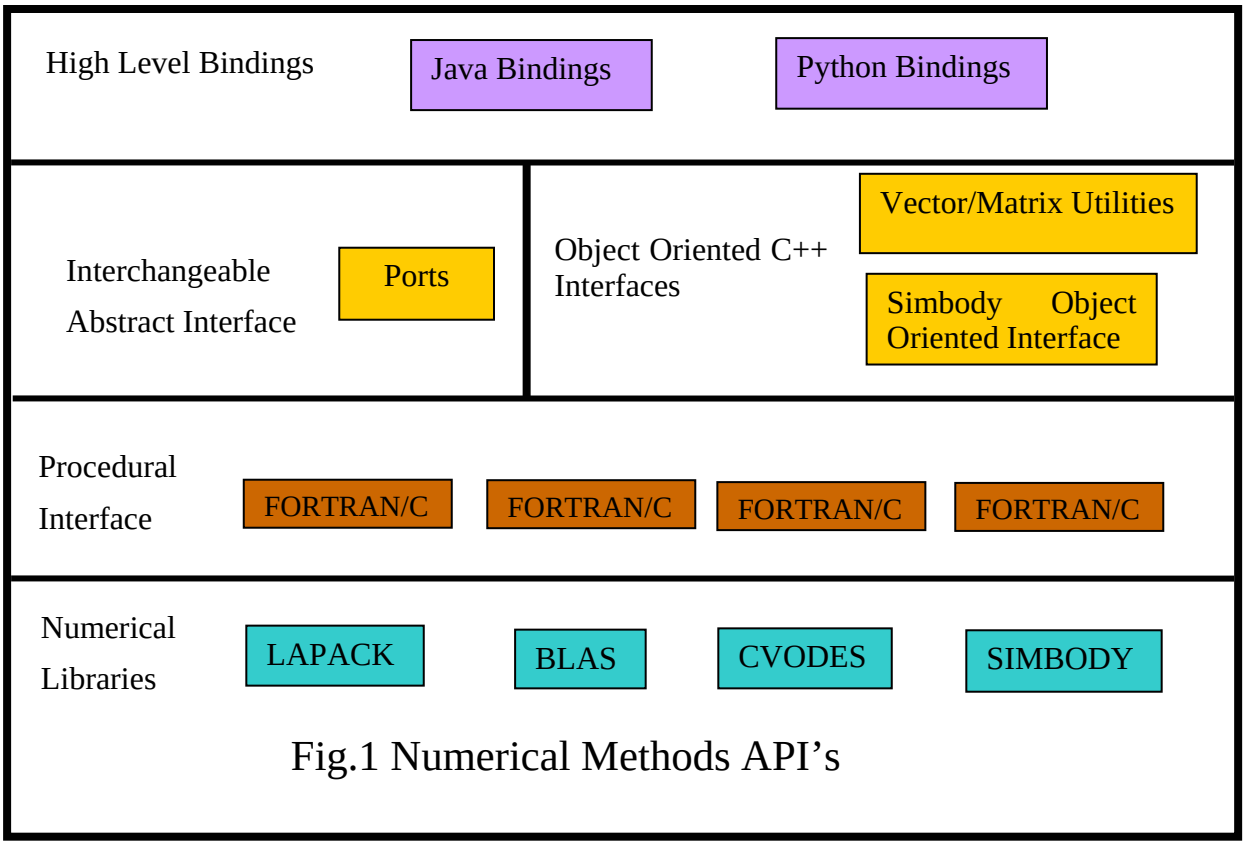

Fig.1 Numerical Methods API's

5.1 Procedural Interfaces

Procedural interfaces use conventional function/subroutine calls to access the numerical method. It is a requirement for SimTK 1.0 to provide procedural interfaces in ANSI C and FORTRAN 77 for all the numerical libraries it supports. LAPACK, BLAS, and CVODES have existing procedural FORTRAN and C interfaces however procedural interfaces will have to be created for Simbody.

5.2 Interchangeable Abstract Interface

The Interchangeable Abstract Interface provides standard interfaces for numerical methods which solve similar problems. This allows applications to easily switch between different numerical methods. For example, if an application is currently using a non-stiff ODE solver and does not know if the ODE system is stiff, it can easily switch to a stiff solver do a performance comparison.

Interchangeable interfaces are based on abstract classes called "ports". An application can easily interchange numerical methods which implement the same port. SimTK 1.0 will develop prototypes of port definitions for ODE, linear algebra and multibody dynamics solvers. Because SimTK 1.0 will only have one example for each of these types of solvers it will not be practical to finalize the port definitions without comparing them to other solvers of the same type. Therefore, this facility will still be in a prototype form in SimTK 1.0.

5.3 Object Oriented C++ Interfaces

The object oriented C++ interfaces are a collection of high level, object oriented interfaces. One example is a vector/matrix package which does basic operations on vectors and matrices similar to the functionality in MATLAB and insulates applications from the details of data formats of the particular numerical libraries. SimTK 1.0 will begin to prototype a vector/matrix package but it will not be a requirement. Another example of high level interfaces are cases where a numerical library has an existing object oriented interface such as Simbody. Simbody is based on the IVM library which has an existing object oriented interface. SimTK will make these interfaces available to the application but it is not a requirement that all numerical libraries have object oriented interfaces.

5.4 Bindings for High Level Languages

At the highest level of abstraction are bindings for high level languages such as Java, Python, Tcl, etc. These bindings will be wrappers on top of the lower level interface layers and allow applications written in high level languages to use the numerical methods. Future releases of SimTK will generate bindings using SWIG however; these are not required for SimTK 1.0.

5.5 Deployment of Numerical Methods

Numerical methods can be deployed in three ways:

1. In conventional libraries
2. Hard coded into the application
3. As CCA components

Conventional libraries can be either static or dynamic libraries and must be linked or loaded with the application. Numerical methods can also be hard coded directly into the application. This is logically the same as a statically linked conventional library. A CCA component is a dynamically linked library with additional interfaces to allow operations on components such as discovery. CCA components must implement the neo classic version of the CCA spec [1]. One of the advantages of CCA components is that components which provide the same ports can be easily interchanged. Since SimTK 1.0 will only provide prototypes of selected ports any CCA components in SimTK 1.0 will be strictly prototypes and will not be requirements. Note that we intend to use the same port definitions in our “Interchangeable Abstract Interface” as are used in CCA components.

6 Cross platform Requirements

The numerical libraries included in SimTK 1.0 and their test programs must compile and run on the following platforms: Windows 32/64bit, Red Hat Linux 32/64bit, and Mac Tiger with G5. This will require that the build systems for all numerical libraries be ported to cmake. An evaluation will be done to determine if it is practical for these libraries to compile natively or if cygwin should be used. The first choice will be to compile these libraries natively because this will allow users to use the Microsoft Studio set of development tools on these libraries. Also, the SimTK CCA component framework library will need to be ported to Windows. However, because CCA components are still in the prototype stage this will not be a requirement for SimTK 1.0.

7 Download and Installation Requirements

Some numerical libraries are layered on top of lower level computation and communication libraries which have been tuned for specific platforms. A good example is the BLAS (Basic Linear Algebra Subprograms) which is used by LAPACK [3]. Most hardware vendors have developed custom versions of these libraries which are tuned for their platform. Both Intel and AMD have developed optimized versions of BLAS for

their platforms. SimTK 1.0 will be required to download and install the correct version of these platform specific libraries with little intervention from the user.

LAPACK and BLAS routines which are optimized for AMD are distributed as part of the AMD Core Math Library (ACML) library. ACML is supported on Windows and Linux for 32 and 64 bit with both static and dynamic versions. AMD also provides separate ACML libraries for gnu, PGI [5] (www.pgroup.com) and pathscale (www.pathscale.com) compilers. For SimTK 1.0 only the gnu version will be installed. There are also different versions which are tuned for the SSE, SSE2, SSE3 instruction set extensions.

Intel sells optimized versions of LAPACK and BLAS for Linux, Windows, 32 and 64bit, and has different 64bit versions for Itanium and EMT systems. The BLAS routines appear in the libmkl_i2p.so, libmkl_lapack*.so libraries. We will need to determine if the Intel licensing agreement allows us to include these in SimTK.

The download and installation mechanism for SimTK 1.0 will determine what operating system, cpu vendor and instruction set extensions the user has and decide which software needs to be installed. For SimTK 1.0, the software will always be installed into the default directory (TBD). In future releases of SimTK the user will be given the option of changing the install directory. After SimTK 1.0 has been installed, an executable called “SimTK_install_check” will be launched which checks if SimTK 1.0 was successfully installed and prints out a descriptive error message if any problems were found. We will investigate using Java Web Start to download and install the libraries, and test programs for numerical methods in SimTK 1.0.

8 Scalability of Numerical Libraries

The numerical libraries in SimTK 1.0 will be able to take advantage of tightly coupled multiprocessors using multi-threading. This will be done in two ways. In the first case, a parallel version of an ODE solver based on CVODES [4] will be implemented. The parallel version will be thread safe so that applications can spawn multiple threads so separate computations can be run on different cpu's. Second, LAPACK will support multiple cpu's through a multi-threaded version of BLAS (Basic Linear Algebra Subprograms) on AMD and Intel systems. In both cases, by default the number of threads created will be the same as the number of processors on the system. The application user will be able to override the number of threads created, by setting the OMP_NUM_THREADS environment variable.

Scalability on distributed systems using MPI (Message Passing Interface) will be addressed in a future release of SimTK. However, the interfaces that are designed for SimTK 1.0 will need to be able to support MPI on clusters in the future.

9 Testing and validation

Tests will be developed which measure both the quality and performance of the numerical methods. These tests will be used to identify regressions, which are errors that are introduced, due to source code changes or improper installation. Each numerical library will have a set of quality and performance tests that are automatically run on a regular basis and report their results to a Dart server. These tests will be run on Windows 32/64bit systems, Red Hat Linux 32/64bit systems and Mac G5 Tiger systems. The purpose of the quality tests will be to determine if the numerical library is producing the correct results. The initial set of tests will be derived from the examples that are part of the CVODES and LAPACK distributions. New tests will need to be developed for Simbody. Also, a set of performance benchmarks will be developed which measure the performance of a library using different datasets and options. Baseline values for both quality and performance will need to be established for each library on each platform. These baseline values will be used as benchmarks for the results of the tests to determine if any regressions have been introduced. If changes are introduced which affect the quality or performance of the library the baseline values may need to be revaluated.

Tolerance values for both quality and performance will need to be established and used to determine if the results should be reported as a regression. Once the baseline values for a platform have been established, the tolerances for quality should be near the accuracy of the machine precision. Performance tolerances need to

be looser because changes in the source code can change the access patterns of instruction cache which can noticeably affect performance on some platforms. To help alleviate this problem, the performance values will be determined by running several tests and comparing the weighted combined execution time to the baseline values.

A generic numerical benchmark, similar to the SPECmark metrics, will be developed to determine the floating point performance of a system. The generic benchmark will be used to distinguish between performance improvements caused by changes in the numerical methods or improvements in the hardware and compilers. In some cases when new platforms are introduced, changes need to be made to the numerical method in order for it to compile and execute. This makes it difficult to determine if the performance changes were due to the new system or if they were due to changes in the numerical method. The ratio of a numeric benchmark and the generic benchmark can be used to compare against the same ratio on other systems. Significant differences in the ratios suggest that changes in the performance were due to changes in the numerical method's software.

10 Deliverables for SimTK 1.0

The following numerical libraries we be part SimTK 1.0.

1. CVODES for solving ODE's.
2. LAPACK and BLAS for solving linear algebra problems.
3. Simbody for solving multibody dynamics problems.

Each of these libraries will have a set of automated, quality and performance tests which will be run on a regular basis and report their results to the SimTK Dart server. These libraries will have automated download and installation utilities which can be accessed from the SimTK.org website. The installation procedure will determine the correct version of the numerical library for the user's platform, install the libraries and run the SimTK_install_check program once the libraries have been installed to ensure that the installation was successful. Each library will also have a set example programs.

10.1 Additional Documents for SimTK 1.0

The following documents will be created as part of SimTK 1.0

1. A numerical methods test plan which describes the quality and performance tests for the numerical libraries which are part of SimTK 1.0.
2. A numerical methods installation design document which describes how Java Web Start will be used to download and install the numerical methods in SimTK 1.0.
3. A numerical methods user's guide which describes how to use the numerical methods API's in SimTK 1.0.

10.2 Uncommitted Deliverables for SimTK 1.0

The following numerical methods will only be included in SimTK 1.0 if time permits but are not considered requirements.

1. Solve for roots of nonlinear equations using KINSOL from the SUNDIALS [4] package or portions of PETSc [5].
2. Vector and Matrix library. SimTK 1.0 will make a first cut at designing a set of utilities for manipulating vectors and matrices similar to the functionality in MATLAB.

11 What we will not do in SimTK 1.0

The following numerical methods features will not be implemented in SimTK 1.0 but will be implemented in a future release.

1. Distributed processing, e.g. ScaLAPACK. The numerical libraries in SimTK 1.0 will be able to take advantage of multiple processors on a single computer. However, the ability to distribute a computation across multiple computers using MPI will be addressed in a future release.
2. Runtime discovery of CCA components. Runtime discovery is the ability to search for, find, and load new components while the application is running. For example, an application that uses an existing ODE solver could swap in a new and improved ODE solver simply by installing the new solver in the correct location. The SimTK component framework could find the new solver; make it available to the application which could use the new solver without recompiling.
3. Partial Differential Equations. To adequately address the wide range of systems of partial differential equations that SimTK users need to solve will require a suite of numerical methods tools. Some of these tools such as ODE solvers and linear algebra utilities will be in SimTK 1.0. However, these will not be adequate for solving PDE systems.
4. Bindings for high level languages such as Java and Python will be addressed in future releases of SimTK using SWIG [2].

12 Acknowledgments

This work was funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers for Biomedical Computing can be obtained from http:/nihroadmap.nih.gov/bioinformatics.

13 References

[1] http://www.cca-forum.org

[2] http://www.swig.org

[3] http://www.netlib.org/lapack

[4] http://www.llnl.gov/CASC/sundials

[5] http://www-unix.mcs.anl.gov/petsc/petsc-as