SciFinance: A Program Synthesis Tool for Financial Modeling

Robert L. Akers, Ion Bica, Elaine Kant, Curt Randall, Robert L. Young

SciComp Inc.
5806 Mesa Drive, Suite 250
Austin, TX 78731
phone: 512-451-1050, fax: 512-451-1622
email: info@scicomp.com or <lastname>@scicomp.com
www.scicomp.com

Abstract

The SciFinance^{TM} software synthesis system automates the programming task for financial risk management activities ranging from algorithms research to production pricing to risk control. Introduced commercially in late 1998, the system is currently licensed to a number of major investment banks. SciFinance's high-level, extensible specification language, ASPEN, enables quantitative analysts to generate code from concise model descriptions that are written in application-specific and mathematical terminology. From these specifications, typically one page or less, the system will produce a C program thousands of lines long. The specification language's abstractions help analysts focus on their primary tasks—model description, validation, and analysis—rather than on programming details. Compared with manual programming, automating the programming process produces codes that are more sophisticated, accurate, and consistent. Analysts can develop modeling codes within a day that previously took weeks or were not even attempted. SciFinance is an extension to a system that generates scientific computing codes in a variety of target languages including Fortran and C. The implementation integrates an object-oriented knowledge base, refinement and optimization rules, computer algebra, and a planning system. The same knowledge base is used by the specification checking, synthesis, and information portal subsystems.

Problem Description

Financial risk management increasingly demands new and customized simulation codes to implement its sophisticated computational models. These codes, typically designed by the quantitative analysts at investment banks, help determine prices for investment products, make trading decisions, and assess and control financial risk. The rate of growth in this area is striking. For example, the volume of the parent industry, custom ("over-the-counter") derivative securities trading, has increased twelve-fold since 1990 to eighty trillion dollars. Spending for modeling software is close to a billion dollars per year with an expected growth rate of about 10 percent. One way quantitative analysts can keep on top of this growth is with a tool like SciFinance^{TM}, which automates code generation.

A derivative security is one whose value depends on that of some other underlying security. Derivatives allow firms to hedge risk. For example, a multi-national firm may use foreign exchange options to limit its exposure to volatile exchange rates. In 1973, Myron Scholes and Fischer Black derived a partial differential equation, the Black-Scholes equation (for which a Nobel Prize was later awarded), that estimates the fair value of a derivative security as a function of the characteristics of the underlying security and time. Since then, the mathematical theory of derivative pricing has been greatly refined, supporting the explosive growth in the volume and variety of derivatives sold in the marketplace.

Analysts need codes that accurately value and hedge derivative portfolios because as the global derivatives market grows in size, complexity, and competitiveness, clients increasingly demand products tailored to their specific investment requirements. As a bank's suite of investment products grows, corresponding simulation codes must be rapidly and accurately produced.

Large investment banks, brokerage firms, insurance companies, and hedge funds employ quantitative analysts to develop pricing models for these complex derivative structures. Analysts must create a new pricing model whenever a customer needs a price quote on a custom derivative instrument; thus new models must be produced rapidly and frequently. This demand is straining the ability of derivatives houses to model and price these instruments in a timely manner. The complexity of the deals may require a team of analysts, financial engineers, and programmers to work days or even weeks to develop the pricing model. Because a small programming or design error can cost the holding institution millions of dollars, accuracy and consistency of pricing strategies are critical. Quick turnaround is also essential, or the institution may lose the deal to a competitor.

The simulation codes involve the solution of a set of partial differential equations, each of which is an equation like the Black-Scholes equation described in Figure 1. The solution is subject to appropriate boundary conditions, initial conditions, constraints, and possible discrete events such as dividend payments. Especially important are the sensitivities of the solution to the various input parameters. Closed-form solutions are not available for any but the most trivial examples of these problems, and thus numerical approximation codes must be written.

The value V of a derivative security whose underlying stock has current price S, dividend yield D_0, volatility σ, and risk-free interest rate r satisfies the equation:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\,\sigma^2\,S^2\,\frac{\partial^2 V}{\partial S^2} + (r - D_0)\,S\,\frac{\partial V}{\partial S} - rV = 0$$

Figure 1: The Black-Scholes equation.

Analysts ensure the accuracy and efficiency of models, but because they are highly trained and compensated, they make very expensive programmers. Consequently the objective is to greatly reduce programming time while maintaining or improving the accuracy and consistency of the pricing models. In addition, new tools must be both easy to use and familiar enough to inspire confidence in their function.

The problems of financial modeling, although similar to those in other areas of engineering and scientific computing, are especially acute because the field of finance evolves much more rapidly and new models are needed much more quickly. Conventional approaches to producing modeling codes involve combinations of library packages, object libraries, and manual programming. However, these approaches are unsatisfactory for many users (one of our customers evaluated more than 10 products before choosing SciFinance). The reason for the dissatisfaction is that such approaches obscure the model and force the problem solver to think at too low a level of abstraction (reasons are described in more detail in (Akers et al. 1998)). Thinking in terms of the tools or components, rather than in terms of the problem, application, and mathematical solution techniques, can potentially lead the analyst to make compromises that cause inaccurate or incomplete solutions. Large-grained library packages, for example, do not address how to produce new codes when the specifications do not exactly match an existing library routine. Finer-grained libraries shift the emphasis to problems of matching interfaces and connecting components and fail to provide component-spanning optimizations. Manual programming is time consuming and error prone.

To address some of these shortcomings, both object-oriented libraries (including in financial applications) and expert systems (in other application areas) have been developed. Object-oriented libraries can provide more generality by abstracting data structure representations, but they are usually not independent of the specific equations being solved or of properties such as spatial dimensionality and order of accuracy of the algorithm. Even with object-oriented libraries, however, assembly and bottom-up optimization of individual modules is the analyst's focus rather than top-down decision making and global optimization.

Conventional expert systems can select and combine library modules, relieving the user of some of the programming burden. However, expert systems alone do not address issues such as an appropriate specification level, generation of arbitrary higher-order methods, global and problem-specific optimization, and platform re-targeting.

Software synthesis can solve these problems by integrating the best aspects of object libraries and expert systems and augmenting them with the power of computer algebra, program transformation, and planning. Program synthesis accepts specifications in financial and mathematical terms, provides intelligent assistance in making choices, validates specifications and generates error checking code, and optimizes globally with problem-specific knowledge.

Application Description

SciFinance transforms specifications written in a high-level language called ASPEN (Algorithm SPEcification Notation) into executable C code. The synthesis process allows mixed user/system decision making and provides feedback to users in the form of summaries of its work at a sequence of levels of problem refinement.

SciFinance is a customized version of an underlying technology called SciNapse (Kant 1993), (Akers et al. 1997) and part of a general tradition of software synthesis (e.g., (Lowry and McCartney 1991), (Johnson)). Related techniques include the use of planning, theorem proving, or expert systems to compose library modules or to construct scripts. Although these techniques have proved fruitful in other domains, for example using planning to reconfigure software libraries for image analysis (Chien et al. 1999), they typically do not generate the complex control structures and customized data representations that are required in financial applications.

The SciFinance implementation is an object-oriented knowledge base containing application, mathematical, and programming constructs. Integrated with the objects are program transformations (for program elaboration, numerical approximation, data-structure selection and program optimization) and a scheduling mechanism. SciFinance successively refines ASPEN specifications through increasingly detailed levels of representation paralleling a best-practice version of human scientific computing. The levels-of-refinement approach, consistent with state-of-the-art literature (Gallopoulos and Sameh 1997) helps give the user a sense of familiarity and confidence. It also allows mixed user/system decision making. Design choices include questions about the desired results (which only the user can answer) and selection of numerical techniques. SciFinance will make selections in the absence of user specification. SciFinance is implemented in Mathematica[R] (Wolfram 1999).

High-Level Specification Language. The ASPEN specification language represents problems in a way that is both clear to the user and suitable for manipulation by the system. Because ASPEN is concise, expressive, and flexible, users can easily write both simple and sophisticated specifications. Many numerical algorithms, equations, and other mathematical entities can be specified with keywords, and it is easy to specify equations algebraically or to define new, parameterized equation families.

Specification Checking. A front-end specification parser processes specifications and sets up the object instances representing the user's problem statement. Its extensive diagnostics trap and report specification errors early in the synthesis process. The parser is partially dynamically gener-

ated; it constructs its semantic actions based upon the current content of the knowledge base and automatically incorporates the relevant knowledge base content from the user's problem specification. Thus, the knowledge base itself defines what the front end will process and many of the actions it will take. In this sense, the front end is merely a machine presenting and processing a language defined in the knowledge base, which means that new concepts can be introduced with no impact on the front-end processor.

Knowledge Representation. SciFinance mixes rules and objects to represent knowledge about mathematics and programming and to present design choices in appropriate terms and in a logical order. Objects represent entities such as equation sets, individual equations, variables, and solvers. Attributes on object instances not only store object relationships between equations and variables, but also store the design choices that must be made for specific problems. For example, a representation attribute on an array variable can be filled by alternatives such as full, diagonal, time-independent, and stencil. Associated design choice rules encapsulate the details of the knowledge about how to make choices. Representation choice rules, for example, examine the equations in which a variable participates to determine the best data structure for a variable. Choices are not always from a fixed set but may be algebraically constructed based on equation discretizations.

We developed an object-oriented programming system, built on top of Mathematica, that supports dynamically created classes as well as instances. Tools use the dynamic classes to translate declarative, human-oriented descriptions of discrete events, algorithm templates, and discretization rules into the internal object and rule representations. The goals and agendas mechanism, which manages the synthesis and user interactions by ordering the resolution of object attribute values according to their dependencies, is tightly integrated with the object system. It allows attribute values to be computed with methods, constraints, heuristics, and both user-defined and system-defined defaults. Thus, the object system is used not only to organize the knowledge base, but also to encode the synthesis process itself.

Design Choice Rules. Associated with each choice (object attribute) are a result type, constraint and heuristic rules (which both can use previous choices), and defaults. Constraints filter the legal values of the alternatives type. Heuristics (with a simple voting scheme) and defaults are applied next if the ASPEN specification or constraints do not indicate a unique design choice. The specification may contain general default choices, such as an input file for variables not otherwise initialized.

Algorithm Templates. After most design decisions are made, SciFinance constructs a program by instantiating algorithm templates. Templates are special objects that represent mathematical algorithms such as time-evolution loops, equation-system solvers, and interpolations. Template objects are generic algorithm descriptions, free of specific equations and data-structure representations, which provide links to other synthesis entities, including other templates. SciFinance fills out the network of templates, and expands the template objects into pseudocode, inserting assignment statements based on the specific equations and representation selections.

Templates are introduced to the system declaratively. A template translator processes these declarations and incorporates them into the knowledge base in such a way that they are smoothly integrated with the synthesis process.

Elaboration Rules and Global Optimization. Unlike a library-combining process, SciFinance optimizes throughout synthesis, not just as a final code-transformation pass. For example, it eliminates unnecessary problem variables as soon as possible, maximizes parallelism based on equation dependencies, and makes space-time tradeoffs via algorithm choices and problem-specific representation selections. SciFinance also applies conventional optimizations such as the introduction of temporary variables, loop merging, and loop unrolling.

Computer Algebra. SciFinance would be much less powerful without its extensive use of computer algebra to make coordinate system transformations, numerical approximations, error estimates, and data structure and operator optimizations. Various rule-based simplifiers and transformation engines perform the algebraic manipulations of problem entities. Examples include a pseudocode optimizer integrated with a pseudocode elaboration transformer, an inequality simplifier, and translators to convert the system's low-level pseudocode to various target languages, including Fortran and dialects of C.

Platforms. The synthesis engine, SciFinance, runs on a wide variety of platforms (anything that Mathematica runs on), including UNIX, WindowsNT, and Windows9x. SciFinance-generated codes adhere to the standards of the target languages (ANSI C, Microsoft C, and Fortran-77) and may be compiled and run on any platform supporting those languages.

The Mathematica system has two separate parts. The first part is an evaluator/interpreter called the kernel, in which SciFinance objects and rules are implemented. The other part is the Mathematica Notebook, which provides window-based communication between a user and the kernel. The notebooks provide access to a built-in set of WYSIWYG document writing capabilities that can mix text and kernel instructions. We have added new menus for interacting with SciFinance to those already present on notebooks.

Information Portal.

SciFinance's information portal (Young, Kant, and Akers 2000) provides easy access to a suite of Mathematica notebooks presenting information about the system's capabilities and the synthesis in process. The notebooks all utilize a semantic network of information nodes. The semantic network uses the same knowledge representation tools as the rest of SciFinance and can refer to classes in the synthesis knowledge base corresponding to domain entities. The notebooks include reference documents, example catalogs, summaries describing the state of the problem (program) after each level of refinement, and human-authored documents automatically processed to convert selected references to hyperlinks.

```
(* Continuous knockout put, barrier X2,
   leveraged by # samples below X1 *)
Region[SMin<=S<=X2 && 0<=L<=LMax &&
       0<=t<=TMax, Cartesian[{S,L},t]];
When[Interior, BlackScholes1D[]];
When[Boundary,AutomaticBC];
When[max[S], V==0];
When[max[t], V==L*Max[0,K-S]/nsamp];
DiscreteEvents[
 Path[direction[L],
      function[L==SumOf[if[S<=X1,1,0]]],
      ReadFile[tsample, "tsamp.dat"],
      nsample==nsamp]];
Default[TaggedInputFile["DayCount.dat"]];
Output[V, "atSpot.out", spottable,
       L==LSpot, Labelled, NoInitialOutput];
ReadTable[spottable, nspot,
          "spottable.dat"];
CrankNicholson;
```

Figure 2: APSEN Specification - daycount problem

Examples of System Use. Quantitative analysts at risk-management institutions, university professors, and industry consultants use SciFinance to price custom equity-based derivative securities instruments (such as convertible bonds) and foreign exchange instruments. Some of these pricing codes are embedded in production systems for securities valuations, and some are used in research or for validating other approximations. Numerous examples have been published (Gatheral et al. 1999), (Brown and Randall 1999), (Randall, Kant, and Chhabra 1997). SciFinance is also used in university classes in computational finance.

A typical SciFinance application is modeling a derivative security. The model is specified in mathematical and numerical terms familiar to a financial analyst. Many common notions, such as equations, discretization methods, special problem conditions, and numerical algorithms are denoted by name, with variants specifiable via parameterization.

An Example Specification. The example in Figure 2 specifies a simple but non-trivial code that prices a "daycount knockin, continuous knockout put option." The generated code will solve the one-dimensional Black-Scholes equation in a two-dimensional region defined by the underlying stock price S and the knockin path variable L, which counts the number of days. `AutomaticBC` is an ASPEN specification statement that defines linearity boundary conditions. A specific boundary condition overrides this default at `max[S]` to define the continuously monitored knockout boundary condition (at `X2`). The "put" is described by the payoff condition `When[max[t], V==L*Max[0,K-S]/nsamp];` The knockin condition—the fraction of samples (`L/nsamp`) for which the spot price is below a second barrier (`X1`)—is given by the `DiscreteEvents[Path[...]];` specification. The `tsample` in the `Path` descriptor defines the set of sampling dates, and the `ReadFile` specifies their input source. By default, all other inputs are read from the file `DayCount.dat`. The output is the option value `V` interpolated to a specific series
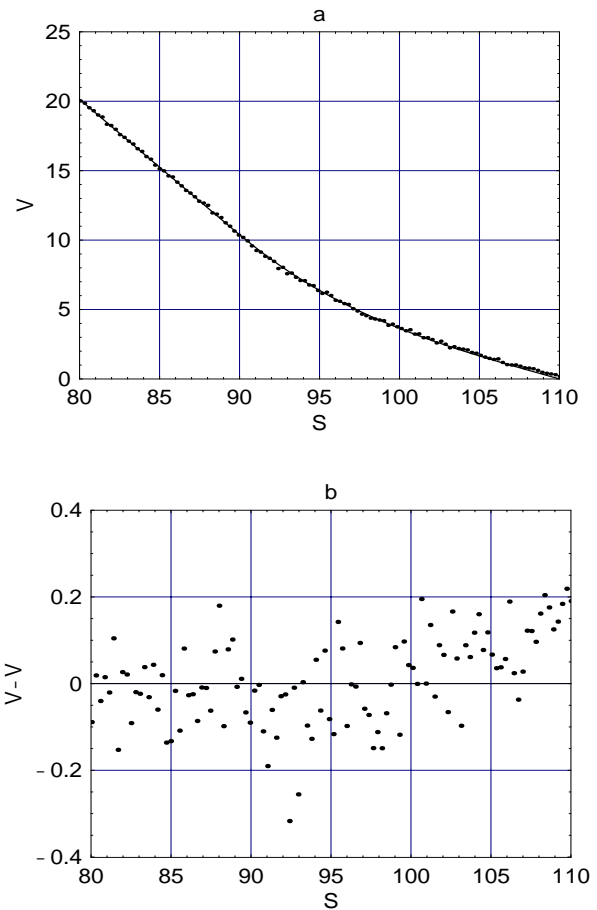

Figure 3: (a) The present value of a daycount-knockin/continuous knockout put option as a function of present stock price: the smooth curve shows results from the finite difference code generated by SciFinance and the dots show the results of the Monte Carlo (random number simulation) code. (b) The difference between the methods: the Monte Carlo code has a slight over-pricing bias near the barrier at 110, is noisy, and is about a factor of 50 slower than the finite difference code.

of spot prices (read into the array `spottable` from `spottable.dat`) and on a specific value of L, namely `L == LSpot`. The specification of numerical methods is optional. For example, The discretization scheme `CrankNicholson` is given, but since no solver is given, SciFinance will make the choice. Given this simple ASPEN specification, SciFinance generates about 1000 lines of C code. Figure 3 shows that the result produced by the SciFinance code is a significant improvement over traditional methods.

Uses of AI Technology

We attribute the success of SciFinance to the naturalness of its high-level specification language and the extensibility of the specification language and of the implementation. As noted previously, the implementation relies heavily on

the integration of object-oriented design with transformation rules, symbolic algebra, and plan-based scheduling. All of these features are extremely useful for a system that is easy to build and extend, has sufficient mathematical flexibility, and is fully automated. For example, rather than plan-based scheduling, the original prototype had a more straightforward concept expansion and rule application procedure. But after attacking some very complex examples, it became obvious the system had difficulty understanding and controlling the rule interactions and rule firings when generating varied and sophisticated codes. Other approaches to software synthesis such as automated deduction do not seem appropriate here for a number of reasons. For example, the numerical methods are only approximations, and the error is not always known, so proving the methods or deriving them automatically would be extremely difficult and time consuming. Given the high degree of accuracy needed, automated learning of methods from examples also seems impractical, although learning how to make some default settings of numerical parameters based on experimentation could be a useful new tool.

We selected Mathematica as the commercial implementation platform that provided the most of the capabilities we need—a multi-platform, unified system including a symbolic programming language with sophisticated pattern matching, a computer algebra system, and a notebook interface with a rich set of capabilities for displaying or entering traditional-looking mathematical notation. Many of our potential customers are already Mathematica users. We built the missing object representation and user interface capabilities as additional layers over Mathematica.

A declarative knowledge representation and good model of the domain were key aspects in developing SciFinance. The domain model includes problem structures, simulation code structures, and the human code-construction process. The synthesis process can thus be derived directly from the domain model. The combination of a good model with easy access to the declarative knowledge and meta-knowledge unifies and simplifies many tasks.

SciFinance classes representing mathematical and programming constructs have attributes corresponding to their properties and various design choices, along with methods for elaborating these attributes. The methods are elaboration and transformation rules that sometimes include substantial algebraic manipulations. The two technologies are appropriately mixed; knowledge representation provides meaningful locations for the methods, and robust, high-quality algebraic transformations produce the needed results.

The planner exploits the knowledge and meta-knowledge to set goals to decompose tasks or to instantiate objects and then refine them by filling in attributes. Its agenda mechanism schedules refinements and design decisions for algorithms, numerical approximations, and data structures. The planning system uses method descriptions to automatically determine refinement orderings that ensure that all data to make choices are in place before decisions are considered. Currently, SciFinance customers provide the specification in a file, but the knowledge representation and planner are for-mulated such that it would be easy to develop specifications interactively if that became desirable.

SciFinance's reflective implementation does trade ease and speed of development for performance. Fortunately, because the synthesis speed is roughly proportional to CPU speed, given reasonable amounts of memory, we can take advantage of the hardware improvement curve to increase the practical problem size that SciFinance can handle without major tuning.

Application Use and Payoff

SciFinance was first announced in October 1998, after some beta testing, and the first commercial sale came in January 1999 when Merrill Lynch licensed the product for use by quantitative analysts in its Global Equity-Linked Product and Technology Unit. Some other customers we are permitted to list are Bear Stearns, MeesPierson, and KBC Financial Products. All report that specifying problems in a high-level language and automating the code generation has many advantages, primarily the ability to quickly develop complex models, focus precious human resources on the most critical analytic tasks, and reap accurate, high-quality, and consistent code. We have no way to obtain an exact count, but we know that hundreds of codes have been generated.

Customers' descriptions of benefits

Promoting a focus on the modeling tasks. Analysts at Merrill Lynch have been using SciFinance for a year. In a detailed case study co-authored with SciComp (Gatheral et al. 1999), they write that software synthesis makes it much easier to handle complex problems and allows them to focus on the problem and modeling choices, rather than on programming and debugging. In doing research, they can now solve within a day or two problems that appeared too complex to solve in a reasonable time using conventional techniques. In addition, quick turnaround gives their busy analysts the time to experiment with alternative techniques and fine-tune production codes. The analysts have also found that automatically generating codes ensures a consistent set of assumptions about the valuation of a portfolio and a consistent style across all models, even when those models are generated by different people over an extended period of time.

Reducing labor in a risk-control environment. Dr. Raymond Hawkins, Associate Director of Risk Control at Bear, Stearns Securities Corporation, uses SciFinance in a risk-control rather than trading environment. The risk-control department performs risk analysis for clearance of client portfolios on a daily basis, re-pricing every single security within a portfolio and doing a variety of stress tests to determine the portfolio risk. For each security, the department first develops an ASPEN specification that incorporates the terms and conditions of the security, then develops a pricing tool from the code that SciFinance generates. Bear, Stearns Securities previously depended on proprietary models for the pricing tools, but moved to SciFinance because they felt it would be an extremely cost-effective approach. For Dr. Hawkins, an important feature of software synthe-

sis is its ability to reduce labor while producing consistent, highly accurate programs. With program synthesis, highly trained analysts can focus their energy on the analysis and risk control, not on programming.

Increasing code accuracy and development speed. Dr. Anastasios Politis, currently a quantitative analyst at KBC Financial Products, has been using SciFinance for a year and a half to generate codes for pricing new options (both for research and production) and to determine whether closed-form solutions are precise enough. Dr. Politis says that SciFinance makes code development faster and easier for him, and that his bank benefits from more accurate models and fewer deals lost because of slow pricing. SciFinance allows Dr. Politis to develop many models within a single day rather than over the course of a week. Using conventional manual programming methods to develop finite difference codes of the variety SciFinance produces, he says, is immensely time consuming (exactly how time-consuming depends on the resemblance to existing codes). By putting the correct numerical elements, such as solvers, at his disposal, SciFinance enables Dr. Politis to develop some new models in just a few hours. For generating certain types of models (those for American-style options and barrier options) SciFinance has become Dr. Politis' preferred method. He finds codes generated by SciFinance to be superior to the more traditional lattice-based codes. In addition, because certain features can be expressed with a single ASPEN specification statement, SciFinance greatly facilitates his pricing of the varied complex features of options such as convertible bonds.

Gaining confidence in models. MeesPierson analysts use SciFinance primarily to gain confidence in their existing models and to test new modeling approaches. They expect to generate production pricing models in the future. With Sci-Finance, analysts have been able to rapidly generate a variety of accurate, PDE-based codes to validate existing pricing products. Also, they can more quickly and confidently test new pricing models, which helps bring new exotic-option products to market faster. MeesPierson analysts also use SciFinance to research new pricing approaches and conduct experiments that give them a better feel for more sophisticated models.

Changes to business processes

For many financial institutions, using SciFinance would require changing the way they integrate codes into their production environment. To minimize these changes, ASPEN provides several integration constructs, both producing top-level codes that are callable from spreadsheets or C++ methods and providing stub functions to call customer routines. Customization of these interfaces is also relatively straightforward because ASPEN can be easily extended.

Merrill Lynch analysts, in addition to writing new codes with SciFinance, expect to rewrite much of their existing codes using the ASPEN specifications as documentation for what the codes do. They consider the business process changes to be positive, indicating an evolution from having traders responsible for everything from designing models to executing transactions into a mature industry characterized by a cooperative division of labor. This division pairs the customer's regular uncovering of new mathematical problems in their derivatives structuring activity with SciComp's experience with numerical PDE solution techniques, thus benefiting both parties.

Future benefits

As synthesis from high-level languages becomes more widely used, we can expect continued extensions in the varieties of financial and numerical methods made available and continued improvements in the efficiency of the generated codes. We also expect analysts to increasingly delegate the responsibility for design choices about numerical methods and parameters to automated systems with expertise in these areas. And as more knowledge is incorporated into the systems, specifications will be couched in even more natural "deal-sheet" terms.

The ASPEN language itself can become a useful communication tool within a large company or even industry-wide. It provides a clear conceptual framework for computational models that separates the problem from numerical algorithms and is free from unimportant implementation details. ASPEN could become a concise vehicle for auditors, risk managers, and regulators to assess portfolio risk at a high level of abstraction. Yet because it is tied to a code generating tool, the exchange and refinement of ASPEN specifications can lead directly into producing high quality executable models. We also see a possibility for ASPEN to evolve into the next-generation language for more general mathematical modeling.

Application Development and Deployment
A Brief History

SciFinance and its underlying SciNapse technology evolved over about eight years with an average of two or three computer scientists as implementors and one or two mathematicians and physicists as advisors, testers, and users. The precursor project, called Sinapse, began at Schlumberger in late 1990 with the application of modeling seismic and acoustic logging tools and with a target language of Connection Machine Fortran. Several generated codes (after some hand tuning) were used in internal logging tool design projects. In 1995, Elaine Kant, the head of the Schlumberger Sinapse project, acquired the rights to the code and founded SciComp to further develop the system. A three-year NIST Advanced Technology Program award funded additional research to advance software synthesis technology for scientific computing on multiple architectures. After about two years, the focus began to narrow to financial applications and the generation of C code. After some venture funding from the Verticality Investment Group and about a year of additional development and beta testing, full-fledged commercial sales of SciFinance began in October 1998.

The development process, though not formal, is close in spirit to a spiral model. After initial prototyping, system evolution was essentially incremental, adding new mathematical constructs and new programming or optimization

knowledge based on project plans and customer demand. Major replacement of the specification language and parser, the planning system, and the code generator occurred without interruption to system availability. After every major change, a growing set of regression tests (based on examples generated in-house and examples that customers chose to share) is run. Subsets of the tests are run after smaller changes.

Development Issues

Building a rapidly extensible underlying technology has always been a goal and has occupied much of the first years of SciNapse development. Success has been based on the object-oriented design with an emphasis on making the system interface, specification language, and documentation self-generative from the knowledge base (it took three tries to develop the most workable specification language). Also, the template representation of algorithms brings regularity and discipline to the definition of new algorithms and their availability in specifications. And Mathematica, despite its shortcomings in execution speed, programming environment, and user interface modifiability, has served as a flexible programming language that integrates programming, computer algebra and notebook interface in a single system.

Focusing on the application area and involving users in the process began early, though even earlier would have been better. After we published some textbook-level financial modeling examples in August 1996, potential customers started sending us challenge problems. We generated some increasingly sophisticated financial modeling codes, and eventually committed ourselves to finance as our initial application area. At that time, we stopped new work on parallel computing and Fortran, both useful for numerical modeling but not necessary for SciFinance. We tried, but to date have failed, to set up any formal development partnerships.

Deployment Issues

Customers typically evaluate SciFinance for several months before deciding to buy. During this period, quantitative analysts learn how to write ASPEN specifications and determine how easy it is to produce the codes they need. Usually they generate the equivalent of some of their own codes and compare them for accuracy and efficiency. When convinced that automatically generated codes are of at least comparable quality, they move on to a current problem of substantially greater difficulty to ensure that it too can be specified and correctly generated. The last step is usually to determine whether they can easily adapt the generated codes to the bank's production environment. In some cases this involves a customization of the ASPEN interface specification features to the particular information technology needs of the bank.

Convincing analysts that they can spare the time to try a new paradigm is a continuing hurdle. Enormous consequence falls on the appropriateness and accuracy of their models, and they are accustomed to working with familiar methods in a tightly controlled way. Some analysts do not regularly use PDE methods. Although time is still a major issue, the initial skepticism about whether SciFinance could generate sophisticated and accurate codes is rapidly diminishing as our customer list and technical publication list grow. We must, however, continually increase the scope of the system's applications and mathematical sophistication in order to keep existing customers and attract the new customers that already have substantial bases of existing codes.

Rapidly developing interfaces to customers' proprietary environments is an important part of making the system useful to a broad range of institutions. Unfortunately, there are no industry standards for trading system interfaces or error handling. There are too many different commercial and in-house developed back-office systems to develop in advance for all possibilities. Instead we provided some basic solutions and developed tools to simplify customization. ASPEN has many constructs for reading and otherwise initializing data, specifying external calls to user functions, and making the generated codes callable. It also has ways to specify dynamic memory management (in C) and an error-handling scheme that propagates error codes. Based on customer requests, we have created half a dozen parameterizations for the structure of the top-level generated routine, and we have modified the original error-handling scheme to make it thread-safe.

Maintenance

SciFinance serves a competitive market with rapidly evolving needs. As a commercial product, it will grow and be maintained for a long time. Evolution includes not only bug fixes, but also the addition of new algorithms, performance enhancements, better design choice heuristics, new design choice options, and interface extensions. As previously discussed, SciFinance was designed with continuous update in mind, and many system features are derived directly from the knowledge base, which also has many internal consistency checks. We update the internal development version continually, with commercial releases about once every two months. A release typically includes about a half dozen new or extended features as well as several bug fixes.

Users can make some extensions through specification macros, and eventually we will make an algorithm description language available. More extensive additions to the knowledge base must be made by the developers based on suggestions from customer or staff mathematicians and financial analysts. Typically, staff mathematicians and computer scientists must work together to devise the most appropriate generalizations of the financial-construct and mathematical-optimization suggestions before they are implemented. In addition to keeping up with new financial constructs and improvements in numerical methods, we plan the more intensive addition of a new class of methods, Monte Carlo simulations. Over the long term, we may expand into additional areas of financial modeling, produce generic PDE packages for students and professional engineers, extend the system to specific applications, and provide interfaces for less technically oriented users.

It is crucial that SciFinance generate correct code, which

is especially challenging because the system is most attractive to people who push the limits with complex, marginally tractable problems (professional practitioners already have solutions to the easy problems). We attack the correctness problem with the specification parsing previously described and with automated development-time tests and regression tests. During development, the template translator employs extensive semantic checking, providing diagnostic assistance like that of a helpful compiler. The object methods are checked for circularity and type conformance. Synthesis-time appropriateness checks encoded in our object methods help guarantee that the process is running as expected. An object examiner and various process-monitoring tools assist in unit testing and debugging. Much of the documentation is re-generated from the system's semantic network of information whenever it changes, minimizing maintenance effort and eliminating the possibility of making certain kinds of errors. A mechanical validity check of the information network verifies that every alleged node reference is to a node that actually exists and checks that every node in the network can be reached. We always subject proposed system updates to extensive regression testing, which compares the regenerated codes with previous versions, runs them through Purify^R, tests the numerical results, and monitors execution times. We test widely over the cross product of new features, including incorrect specifications, to ensure graceful error recovery and cogent diagnostics.

Summary

SciFinance brings an integrated set of AI, knowledge-based, and computer-algebra techniques to bear on the real-world problems of numerical modeling, providing a commercial software synthesis system for solving PDEs in computational finance. Customers testify that the system increases productivity, reduces development time, and yields consistently high-quality codes that can conform to institutional environments. The system's mathematical knowledge can lower the entry barrier for non-mathematicians, and the extensive data structure and programming knowledge completely relieve the user of coding burdens. The system's common knowledge base minimizes maintenance efforts. The evolution of the target application from sonic and seismic modeling to computational finance demonstrates the adaptability of the system's fundamental design. This flexibility also allows developers to respond rapidly to user needs, a necessity in the fast-moving world of securities option pricing.

Acknowledgments

References

Akers, R.; Kant, E.; Randall, C.; Steinberg, S.; and Young, R. 1997. SciNapse: A Problem-Solving Environment for Partial Differential Equations. *IEEE Computational Science and Engineering*. 4(3):32-42.

Akers, R.; Baffes, P.; Kant, E.; Randall,C.; Steinberg, S.; and Young, R. 1998. Automatic Synthesis of Numerical Codes for Solving Partial Differential Equations. Special Issue *Non-Standard Applications of Computer Algebra* of *Mathematics and Computers in Simulation* 45(1-2):3-22.

Brown, G. and Randall, C. 1999. If the Skew Fits. *Risk Magazine* 12(4):62-65.

Chien, S.; Fisher, F.; Lo, E.; Mortensen, H.; and Greeley, R. 1999. Using Artificial Intelligence Planning to Automate Science Data Analysis for Large Image Databases. *Intelligent Data Analysis* 3:159-176.

Gallopoulos, E., and Sameh, A. 1997. *CSE: Content and Product. IEEE Computational Science and Engineering* 4(2):39-43.

Gatheral, J.; Epelbaum, Y.; Han, J.; Laud, K.; Lubovitsky, O.; Kant, E.; and Randall, C. 1999. Implementing Option-Pricing Models Using Software Synthesis. *Computing in Science and Engineering* 1(6):54-64.

Johnson, W. L., and Nuseibeh, B., eds. *Automated Software Engineering: An International Journal*.

Kant, E. 1993. Synthesis of Mathematical Modeling Software. *IEEE Software* 10(3):30-41.

Lowry, M. R., and McCartney, R. D., eds. 1991. *Automating Software Design*. Menlo Park, CA:AAAI Press/The MIT Press.

Randall, C.; Kant, E.; and Chhabra, A. 1998. Using program synthesis to price derivatives. *Journal of Computational Finance* 1(2):97-129.

Wolfram, S. 1999. *The Mathematica Book*. Wolfram Media/Cambridge University Press.

Young, R. L; Kant, E.; and Akers, L. A. 2000. A Knowledge-Based Electronic Information and Documentation System. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, 280-285. New Orleans, LA:ACM Press.