Using the IMSL MATH/LIBRARY in Numerical Methods Courses

by

K. R. Jackson
T. E. Hull
University of Toronto

IMSL Technical Report Series
No. 9005

IMSL, Inc.
2500 Permina Tower
2500 City West Boulevard
Houston, Tesas 77042-3020, USA
Telephone (713) 782-6060 Telex 791923 IMSL INC HOU

Using the IMSL MATH/LIBRARY in Numerical Methods Courses

K. R. Jackson, T. E. Hull
Computer Science Department
University of Toronto
Toronto, Ontario, Canada M5S 1A4

April 18, 1990

Abstract

This article describes our experience using the IMSL MATH/LIBRARY in five numerical methods classes taught to engineering students at the University of Toronto during the 1987–88 and 1988–89 academic years.

1 A Profile of the Students, Prerequisites, and Course Objectives

A one-term course in numerical methods, comprising thirty-nine 50-minute lectures, was taught twice by Hull to a small class (12 students one year, 6 the next) of fourth-year electrical engineering (EE) students. Jackson taught similar one-term courses, comprising twenty-six 50-minute lectures, twice to a large class of about 80 third-year engineering science (ES) students and once to a combined class of about 100 third-year industrial, geotechnical, and metallurgical engineering (IGME) students.

The students in these various engineering streams have different mathematical and programming training, aptitude, and ability. Moreover, their desire to learn about numerical methods and scientific computing varies greatly.

All students had previously taken courses in calculus (in both one and several real variables), complex analysis, linear algebra, and computer programming. Only the IGME class had been taught FORTRAN. In addition, the EE and ES students had previously taken a differential equations course; the IGME class studied differential equations and numerical methods concurrently. All students had been introduced to elementary numerical methods, such as the Newton-Raphson iteration. The IGME class had a more extensive exposure to the numerical solution of linear systems and of nonlinear equations in a single variable, as well as interpolation and approximation in their prerequisite computing course. Thus, the IGME course contained roughly half as much new material as the other two, partly compensating for its brevity and the students' lower level of mathematical ability and motivation.

The radically different class sizes, course lecture hours, and student backgrounds, abilities, and motivations affected our approach to teaching these courses. In each course, it was our goal to cover numerical software approaches to linear algebra, nonlinear equations, interpolation and approximation, quadrature, and ordinary differential equations.

Although we have only taught numerical methods courses based on the IMSL MATH/LIBRARY to engineers, we feel that this approach is equally appropriate for science students.

2 Choosing the IMSL MATH/LIBRARY and Textbook

It was our objective to teach the engineering students how to use numerical software to solve scientific and engineering problems both reliably and efficiently; our goal was to produce sophisticated users, not to clone more numerical analysts or mathematical software developers. Some knowledge of the relevant problem characteristics and underlying numerical algorithms is essential to select and use numerical software intelligently. Most of our lectures were devoted to instilling this knowledge. Secondary goals for these courses were to sharpen the students' programming skills, and to train them in good computing practices.

In previous courses, we used software packages distributed with the popular numerical methods textbooks, as well as mathematical software collected from colleagues and public domain sources such as *netlib* (1987) and ACM TOMS. We feel that exposing students to a comprehensive and sophisticated library, such as the IMSL MATH/LIBRARY, better meets our course objectives. Students should learn to use such a library if they are to engage in any significant scientific computing in their later courses or future professional careers. We have often been dismayed to find that students who learn "toy" mathematical software packages in introductory numerical methods courses persist in using these often inappropriate packages.

Having chosen the IMSL MATH/LIBRARY, our next task was to select a textbook and appropriate documentation. We first considered *Numerical Methods, Software, and Analysis* by Rice (1983), but reluctantly decided against using it because that edition is based on the "old" IMSL Library (before the major revision in 1987). Since we were using Version 1.0, the names of most routines and many of their calling sequences would not agree with those in Rice's book. (Note that a new edition of Rice's book, based on IMSL MATH/LIBRARY Version 1, is scheduled for publication by Academic Press in the third quarter of 1990.) We found no other introductory numerical methods book based on MATH/LIBRARY.

We considered some good textbooks with associated small numerical software packages, but most of them couple the discussion of numerical methods with their software packages. We felt that this coupling would be confusing for our students; therefore, we focused on "software independent" books. Of the many good texts in this category, we chose *An Introduction to Numerical Methods for Differential Equations* by Ortega and Poole (1981), partly because of its novel approach: using the numerical solution of initial value problems (IVPs) and boundary value problems (BVPs) for ordinary differential equations (ODEs) to introduce most of the usual topics covered in an introductory numerical methods course. We felt that this would be most suitable for engineers who need to solve differential equations in many of their other courses, and in their future professional careers. (Regrettably, it seems that this excellent text is being withdrawn from the market.)

All that remained was to choose appropriate documentation for those IMSL routines to be used. The first year, we photocopied the relevant sections from the IMSL user's manual; the second year, students obtained complete copies of the *Softcover Edition* of the *IMSL MATH/LIBRARY User's Manual*. However, IMSL has now released the Interactive Documentation Facility, which provides online documentation and help for all IMSL routines. This, together with the new edition of Rice's book, may be sufficient documentation for most purposes.

3 Sample Exercises

Many course assignments required the students to write programs calling IMSL MATH/LIBRARY subroutines. We used these practical exercises to motivate more theoretical discussions in class, as well as to illustrate and further explore some points developed in lectures. A sample of the exercises taken from two areas, IVPs for ODEs and linear algebraic equations, follows.

Whenever we refer to a specific IMSL routine, we use its single precision name, although all the routines actually called were the corresponding double precision routines.

3.1 A Preliminary Exercise on IVPs for ODEs

After a brief discussion of "The World of Scientific Computing" in Chapter 1, Ortega and Poole introduce the solution of IVPs for ODEs in Chapter 2. We began our courses with these two topics also. As Ortega and Poole concentrate on Runge-Kutta and multistep methods, with only a brief mention of extrapolation methods, we limited our discussion of IVP methods to the IMSL routines `IVPRK` and `IVPAG`. `IVPRK` is a variable stepsize method based on an explicit Runge-Kutta formula-pair of orders 5 and 6. `IVPAG` is a variable order, variable stepsize hybrid method, incorporating both Adams and Gear formulas. (The latter are often called backward differentiation formulas, or BDFs for short.)

To introduce the students to the IVP chapter of IMSL, we asked them to read the sections on `IVPRK` and `IVPAG` and study the example on pages 636–639 of the *IMSL MATH/LIBRARY User's Manual*, Version 1.0. The sample program listed there uses `IVPRK` to solve the often quoted predator-prey problem

$$\begin{aligned} f' &= f(r-1), & f(0) &= 3, \\ r' &= 2r(1-f), & r(0) &= 1, \end{aligned}$$

which models the population dynamics of a two-species system, consisting of a predator, f, and its prey, r. In the IMSL example, the predator-prey problem is solved with the tolerance TOL $= .0005$ on the interval $[0, 10]$, and with the solution printed at $1, 2, 3, \ldots, 10$.

The students modified this example to solve the problem with the `IVPAG` Adams predictor-corrector method instead of `IVPRK`. Specifically, they first computed an accurate reference solution, by solving the problem with the stringent tolerance of TOL $= 10^{-12}$, then solved the problem again with the cruder tolerances of TOL $= 10^{-1}, 10^{-2}, 10^{-3} \ldots, 10^{-9}$. Finally, the students printed the numerical solution associated with each tolerance at the endpoint, 10, with an approximation to the global error in the cruder solution, calculated by subtracting the accurate reference solution from the cruder one.

In addition to introducing the students to the IMSL MATH/LIBRARY, this exercise demonstrates that the user-supplied tolerance parameter TOL that appears in the IMSL IVP routines specifies a *local error request*, not a *global error bound*. Inexperienced users are often confused about this distinction. Few currently available IVP routines control the global error directly. (The numerical results from this exercise, however, showed that the global error at the endpoint for this problem is roughly proportional to TOL, a property often referred to as *tolerance proportionality*). Although not illustrated in this example, the constant of proportionality is both method and problem dependent: some codes, such as `IVPRK`, exhibit a higher degree of tolerance proportionality than others, such as `IVPAG`, and the constant of proportionality can vary dramatically from problem to problem. After the students completed this exercise, it was easy to explain to them how extrapolation based on solutions computed at two different tolerances by a code that exhibits good tolerance proportionality can be used to compute a crude global error estimate, a useful "trick" that can be used in many other similar situations.

3.2 Error Control for IVPs for ODEs

To emphasize the importance of choosing an appropriate error control, we asked students to use `IVPRK` to solve the simple IVP $y' = y$, $y(0) = 1$, twice on the interval $[0, 50]$ with tolerance TOL $= 10^{-6}$ each time, but with relative error control in the first run and absolute error control in the second. In both cases, they printed the solution and the associated global error at $5, 10, 15, \ldots, 50$. With relative error control, an accurate numerical solution is computed efficiently. But with absolute error control, the program aborts part way through the integration with the message: "Maximum number of steps allowed, 500, used. The problem may be stiff." Students were asked to explain the cause of the difficulty, and why it occurred in the second run, but not the first.

This simple IVP is not stiff. The source of the difficulty here is that the numerical solution y_n at t_n approximates e^{t_n}. Thus, to keep an estimate of the absolute local error in y_n less than 10^{-6} in magnitude would require the computer to carry the equivalent of about 25 decimal digits and use an excessively large number of steps. On the other hand, to keep an estimate of the relative error in y_n less than 10^{-6} in magnitude requires only that roughly the equivalent of the six most significant decimal digits in the local solution be correct, independent of how large t_n and y_n may be — provided, of course, that y_n does not "overflow." As this example illustrates, it is generally inappropriate to request absolute error control in problems for which the computed solution is large. Moreover, it emphasizes that the user should give careful thought to the appropriate error control for a problem, a commonly overlooked source of difficulty.

3.3 A Comparison of IVP Methods for ODEs

The students were asked to use a set of driver routines written by Steve Young, a teaching assistant, to solve each of the following four problems taken from DETEST (1981). Each problem is solved on the interval of integration $[0, 10]$.

1. Bessel's equation of order 1/2 with the origin shifted one unit to the left, written as a first-order system:

$$\begin{aligned} y_1' &= y_2 & \qquad y_1(0) &= 0.6713967071418030 \\ y_2' &= \left(\frac{1}{4(x+1)^2} - 1\right) y_1 - \frac{y_2}{x+1} & \qquad y_2(0) &= 0.09540051444747446 \end{aligned}$$

2. Euler's equations of motion for a rigid body without external forces:

$$\begin{aligned} y_1' &= y_2 y_3 & y_1(0) &= 0 \\ y_2' &= -y_1 y_3 & y_2(0) &= 1 \\ y_3' &= -.51 y_1 y_2 & y_3(0) &= 1 \end{aligned}$$

3. A radioactive decay chain problem:

$$\begin{aligned} y_1' &= -y_1 & y_1(0) &= 1 & & \\ y_i' &= (i-1)y_{i-1} - i y_i & y_i(0) &= 0 & & i = 2,3,4,\ldots,9 \\ y_{10}' &= 9 y_9 & y_{10}(0) &= 0 & & \end{aligned}$$

4. The Robertson chemical kinetics problem:

$$\begin{aligned} y_1' &= -.04 y_1 + .01 y_2 y_3 & y_1(0) &= 1 \\ y_2' &= 400 y_1 - 100 y_2 y_3 - 3000 y_2^2 & y_2(0) &= 0 \\ y_3' &= 30 y_2^2 & y_3(0) &= 0 \end{aligned}$$

The first two problems are nonstiff; the third is mildly stiff; while the fourth is quite stiff.

Each of Young's eight driver routines solves one of these four problems with either `IVPRK` or `IVPAG`. Each driver asks the user for a tolerance. The `IVPAG` drivers also ask whether the user wishes to use the Adams or Gear formulas, and whether they want to use one of the following to solve the equations associated with the implicit Adams or Gear formulas: functional iteration, the chord method with a user-supplied Jacobian, or the chord method with an approximate Jacobian computed with divided differences by `IVPAG` itself. The driver then computes the solution with the method specified, and prints both the solution values and approximate global errors at the output points $1, 2, 3, \ldots, 10$. Additionally, it reports the tolerance selected, the number of steps and the CPU-seconds taken, and the number of function and Jacobian evaluations used throughout the integration.

The students used the principle of *tolerance proportionality* discussed in "A preliminary exercise on IVPs for ODEs" to select a tolerance that would produce a maximum global error at the output points of between .0005 and .001 for each of the possible 28 combinations of methods and problems. The students also collected and analyzed the data produced. Because of the bulk of programming required to write the routines to collect these statistics, we felt it best to supply the students with Young's drivers.

The main lesson clearly illustrated here is the important distinction between stiff and nonstiff problems. `IVPAG` with Gear formulas and the chord method, a combination appropriate for stiff problems, solves the stiff Robertson chemical kinetics problem far more efficiently than does either (a) `IVPRK`, or (b) `IVPAG` with either Adams or Gear formulas and simple iteration. The results also show that there is little difference between using the user-supplied Jacobian and one approximated with divided differences by the method itself, a rule-of-thumb that holds true except in extreme cases.

In addition, the results demonstrate that while all methods can effectively solve the first two nonstiff problems and the third mildly stiff one, the nonstiff methods `IVPRK`, and `IVPAG` with the Adams formulas and simple iteration, are a little better. One can extrapolate from the numerical results that the superiority of the nonstiff methods for nonstiff problems will usually increase with the number of components in the ODE system. Nonstiff methods, unlike the stiff ones, do not require the computation of a Jacobian or the associated solution of linear systems, the costs of which grow rapidly with the number of components.

One surprising result emerged: at the tolerance requested, `IVPAG` with the Adams formulas and the chord method solves the stiff Robertson chemical kinetics problem almost as efficiently as `IVPAG` with the Gear formulas and the chord method. The explanation for this is that, although the Adams-Moulton formulas of orders greater than two are not suitable for stiff problems, the second-order Adams-Moulton formula is the trapezoidal rule, which, coupled with the chord method, is suitable for stiff problems. In this example, the order selection strategy for `IVPAG` with the Adams formulas reduces the order to two when stiffness is encountered, and successfully integrates the problem with the trapezoidal rule. In addition, because of the lax tolerance requirement, the second-order formula is a relatively effective choice in this case. However, one cannot expect to be so lucky in general.

3.4 A Comparison of Methods for Solving Systems of Linear Algebraic Equations

One advantage of using a comprehensive library in a numerical methods course is that it provides a wide variety of routines for solving standard problems. Our courses emphasize that users should try to choose software appropriate for the problem at hand. This can be illustrated effectively for linear systems of algebraic equations. Consider the banded symmetric positive-definite $N \times N$ matrix

$$A_N = \begin{bmatrix} 5 & -4 & 1 & 0 & 0 & 0 & \cdots & 0 \\ -4 & 6 & -4 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & 1 & -4 & 6 & -4 & 1 & 0 \\ 0 & \cdots & 0 & 1 & -4 & 6 & -4 & 1 \\ 0 & \cdots & 0 & 0 & 1 & -4 & 6 & -4 \\ 0 & \cdots & 0 & 0 & 0 & 1 & -4 & 5 \end{bmatrix}$$

from the book of test matrices by Gregory and Karney (1969, p. 75). It is easy to verify that, for $h = 1/(N+1)$ and each $k = 1, 2, 3, \ldots, N$, A_N has an eigenvector $\{\sin(k\pi ih)\}_{i=1}^{N}$ and corresponding eigenvalue $4\{1 - \cos(k\pi h)\}^2$. Consequently, the condition number of A_N is asymptotically equal to $4N^4/\pi^4$. For moderately-sized problems ($N = 10$ to 300), the effect of conditioning on the numerical solution can be observed, but the problem is not so ill-conditioned to preclude solution with the floating-point hardware available on most computers.

For the values of N listed in Table 1, we asked the students to solve the system $A_N x_N = b_N$ for $b_N = A_N e_N$, where e_N is the vector of N ones, using the following IMSL subroutines:

- `LSLRG`, for full, general matrices
- `LSLSF`, for full, symmetric matrices
- `LSLDS`, for full, symmetric, positive-definite matrices
- `LSLRB`, for banded, general matrices
- `LSLDS`, for banded, symmetric, positive-definite matrices.

In each case, students recorded the CPU-time used and the error in the computed solution x_N. The CPU-times in seconds for a MicroVax II are listed in Table 1.

N	`LSLRG`	`LSLSF`	`LSLDS`	`LSLRB`	`LSLDS`
50	.333	.250	.433	.117	.100
75	.700	.533	1.12	.183	.183
100	1.22	.933	2.28	.283	.283
150	2.73	2.03	6.48	.483	.533
200	4.80	3.55	14.1	.717	.900
250	7.45	5.51	25.9	1.05	1.30
300	10.7	7.88	43.0	1.42	1.82

Table 1: CPU-times in seconds for the solution of $A_N x_N = b_N$.

Based on the operation counts for the standard algorithms, we expected to see that the CPU-times for the full matrix routines `LSLRG`, `LSLSF`, and `LSLDS` would be proportional to N^3, with the constant of proportionality for the symmetric positive-definite routine `LSLDS` being about half that of the general routine `LSLRG`, and the constant of proportionality for the symmetric routine `LSLRG` lying between these two extremes. Moreover, we anticipated that the times for the band matrix routines `LSLRB` and `LSLDS` would be proportional to N, and hence significantly less for large N than the times for the full matrix routines. We also expected the constant of proportionality for the symmetric positive-definite routine `LSLDS` to be a little less than that for the general routine `LSLRB`.

The results for the band matrix routines `LSLRB` and `LSLDS` are roughly as we expected, except that `LSLRB` is a little more efficient than `LSLDS` for large N. However, the results for the full matrix routines `LSLRG`, `LSLSF`, and `LSLDS` were not as we anticipated. To our surprise, the CPU-times for `LSLRG` and `LSLSF` are roughly proportional to N^2, and the times for both are significantly less than those for the symmetric positive-definite routine `LSLDS`, the times for the latter being roughly proportional to N^3 as expected.

We sent the data for this apparent anomaly to IMSL. Jing Li, an IMSL mathematical software designer, gave us an explanation. When the routines `LSLRG` and `LSLSF` find a zero multiplier during the factorization stage, the associated row multiplication and addition are skipped. For this example, only two row multiplications and additions are required at each step. Hence, the number of floating-point operations is proportional to N^2, as the numerical results suggest, rather than N^3, as our simple model forecast. Jing Li also sent us some numerical results that show that, if 10^{-6} is added to all elements of the matrix A_N, making it full rather than banded, then the CPU-times for `LSLRG`, `LSLSF`, and `LSLDS` behave exactly as we originally expected.

4 The Students' Evaluation of the Course

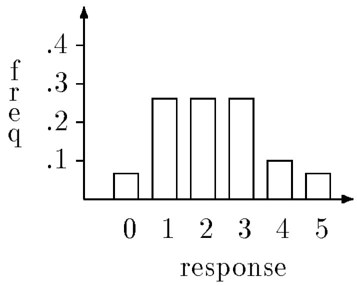
The use of the IMSL MATH/LIBRARY in our numerical methods courses was generally well-received by the students. They found the IMSL documentation and routines harder to understand than the commonly used "toy" software, but by the end of the course, they appreciated their exposure to "real world" professional mathematical software.

Jackson conducted a short written survey in his 1989 ES course. For each of the four questions listed below, the 31 students who completed the survey chose the number that most closely reflected their answer to the question. In each case, we list the mean and standard deviation as well as a histogram of the frequency with which each number was chosen.

Informal polling conducted in the other classes agrees with the generally positive reaction of the students reflected in this survey.

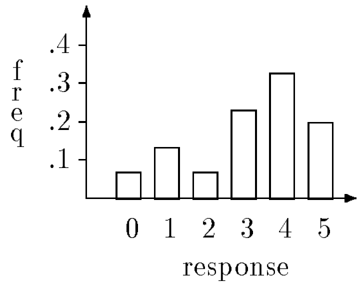
1. Did you find the IMSL MATH/LIBRARY easy or hard to learn to use?

   easy 0 1 2 3 4 5 hard
   mean: 2.26 s.d.: 1.32



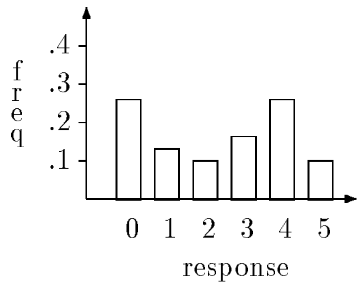
2. Now that you have learned to use the IMSL MATH/LIBRARY, do you think that the time taken was worthwhile?

   no 0 1 2 3 4 5 yes
   mean: 3.19 s.d.: 1.51



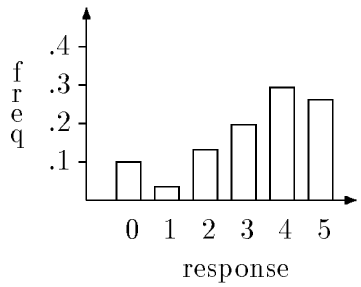
3. Do you think that you will use the IMSL MATH/LIBRARY in other courses or your senior thesis?

   no 0 1 2 3 4 5 yes
   mean: 2.32 s.d.: 1.80



4. Do you think that learning to use the IMSL MATH/LIBRARY will be a long-term benefit to you as a professional engineer?

   no 0 1 2 3 4 5 yes
   mean: 3.32 s.d.: 1.56



5 Conclusions

There are three major advantages in using a comprehensive mathematical software library in a numerical methods course:

1. Provides a broad range of reliable, robust, and efficient numerical routines.

2. Exposes the students to "real world" sophisticated mathematical software, preparing them to use state-of-the-art routines in their later courses, thesis research, and professional careers.

3. Motivates discussion of the algorithms and leads the students to a deeper understanding of the underlying methods and how to use sophisticated mathematical software intelligently. Often, unexpected results, such as those discussed in "A comparison of IVP methods for ODEs" and "A comparison of methods for solving systems of linear algebraic equations," are particularly useful for this purpose.

On the other hand, the following considerations should be addressed before using this approach:

- It is easier in many respects for the students, teaching assistants (TAs), and instructors to use a numerical methods text that is integrated with the mathematical software package used in the course.

- Even with a suitable text and documentation, using a sophisticated mathematical software library demands more of the students, TAs, and instructors than using a "toy" numerical methods package. The routines, underlying algorithms, and documentation are more complicated and require more lecture-time to explain and more effort from the students to master. Therefore, either the students must work harder, or fewer topics can be covered than in an equivalent traditional course.

- Almost all sophisticated numerical software is written in FORTRAN, while the recent trend has been for undergraduate students to learn structured programming languages, such as Pascal. Consequently, in addition to learning numerical methods and software, students may be faced with assimilating FORTRAN as well.

- As our examples above demonstrate, the likelihood of obtaining unexpected results with sophisticated mathematical software is greater than with simple numerical packages. The instructor is required to prepare exercises more carefully.

- Sophisticated packages often require substantial computer resources for storage, compilation, and execution of routines.

- The instructor may be faced with a charge for using commercial software if it is not already available at his site. If source code is not available to the students, the instructor cannot use the source to illustrate points about writing good mathematical software and the students cannot run the routines on other systems, such as their home computers.

On the whole, we felt that the advantages of using the IMSL MATH/LIBRARY justified the extra work, and we would recommend that other numerical methods instructors consider this approach.

Acknowledgment. We thank Steve Young for his many insightful comments and helpful suggestions about the preparation of the assignments for these courses while he was working for us as a teaching assistant during the summer of 1988 and the 1988–89 academic year.

6 References

1. Dongarra, Jack J., and Eric Grosse (1987), *Distribution of mathematical software via electronic mail*, CACM 30, pp. 403–407.

2. Enright, Wayne H., and John D. Pryce (1987), *Two FORTRAN packages for assessing initial value methods*, ACM TOMS 13, pp. 1–27.

3. Gregory, Robert T., and David L. Karney (1969), *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, New York.

4. Ortega, James M., and William G. Poole, Jr. (1981), *An Introduction to Numerical Methods for Differential Equations*, Pitman, Marshfield, MA.

5. Rice, John R. (1983), *Numerical Methods, Software, and Analysis: IMSL Reference Edition*, McGraw-Hill, New York.