1980

Is Mathematical Software a Legitimate Research Area?

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
80-362

IS MATHEMATICAL SOFTWARE A LEGITIMATE RESEARCH AREA?

John R. Rice
Mathematical Sciences
Purdue University

CSD-TR 362

ABSTRACT

This essay presents criteria for legitimacy of a research area and applies them to mathematical software. It is concluded that mathematical software is emerging as a legitimate research area. It is also concluded that any area (and mathematical software in particular) does not have to be legitimately established for one to do significant research in the area.

The essay appeared in the SIGNUM Newsletter, 16 (March, 1981) pp. 23-25.

IS MATHEMATICAL SOFTWARE A LEGITIMATE RESEARCH AREA?

John R. Rice
Purdue University

Prologue: Recently the chairman of a Computer Science department wrote to solicit my views "regarding the legitimacy of mathematical software development as a research area that can stand by itself". This question arose in that department's deliberations on promotions and tenure and they sensed "a clear split in the numerical analysis community on this question. The opposing view takes the position that mathematical software development is a sub-branch of the area of numerical analysis for which the software is being developed, and is not itself sufficient to establish research credentials for a numerical analyst." The following is my response to this solicitation.

The question posed is: Is mathematical software development a legitimate research area that can stand by itself. I approach this question from three viewpoints and consider several more specific, related questions:

1. How does any research area become legitimate?

2. Is software development a legitimate research area?

What is the nature of software research?
How does mathematical software research differ from other software research?
How does mathematical software research relate to the general requirements for legitimacy?

3. What is the relationship between numerical analysis and mathematical software?

How are these two areas related technically?
How are these two areas related sociologically?
Can one do significant mathematical software research that is not in fact, numerical analysis research?

4. How does one measure the significance of any particular research result?

After developing these questions, I answer the original question with a qualified yes; mathematical software development is emerging as a legitimate research area. Its patterns are still only partially formed and there are many false starts yet to be made. Even so, I cite first class scientific achievements in mathematical software that could never be accomplished within the framework of traditional numerical analysis. However, I also conclude that the legitimacy of mathematical software as a research area is irrelevant to the evaluation of particular research results.

I propose three criteria to test an area for legitimacy in scientific research:

Criterion 1: There are important and difficult scientific problems to be solved.

Criterion 2: There is a large and accepted body of facts and theories to apply.

Criterion 3: There are accepted successful methodologies for many tasks.

With these criteria we see that (a) Arithmetic does not qualify because it is not difficult (but number theory does qualify); (b) Politics does not qualify because there is no body of accepted facts and theories; (c) Betting on the horses does not qualify because there are no successful methodologies for winning; (d) Numerical analysis, computational complexity and programming language theory do qualify.

There are areas in Computer Science whose qualifications are widely doubted, e.g., artificial intelligence, business systems and software. I consider the last of these in more detail.

I divide software research into two categories: internal and external. Internal research does not involve what the software is supposed to do. It is the smallest of the two categories and includes specialties like program verification, software portability, software science, etc. The external research is "applications" oriented; the principal subcategories include operating systems, language processing, business systems and mathematical software. Research in these subcategories involves an interplay between internal software topics, the theory and facts from the application area, and theory and facts specific to the subcategory. For an illustration in the area of mathematical software, consider software for adaptive numerical integration. Good software must be portable, well documented, certified as correct, well structured, etc.; all these aspects are from internal software research. The underlying algorithms use certain numerical quadrature rules and error estimates of traditional numerical analysis; these are from the application area. The research also involves (a) comparative performance evaluations of adaptive numerical integration software including the definition and justification of performance criteria, (b) analysis of the trade-off between time, space and reliability among the data structures that can be selected for the algorithm (c) identification of the problem class for which the algorithm is effective; these items are specific to mathematical software.

All of the external software research areas involve a mixture of theory and practice. The research does not exclusively follow the mathematical pattern; there are engineering aspects (getting things to work well without a complete theory) and experimental science aspects (making observations, classifying phenomena and testing hypotheses). The latter aspects are analogous to much of the research in the biological sciences, astromony and geology.

In principle, mathematical software research does not differ substantially from other external subcategories of software research. In practice it differs in two ways. First, there was a large body of theory and scientists established in the application area before software became important. Thus, it is easy to distinguish between the software and the application area here while in the other subcategories of software research the software and applications theory are thoroughly mingled in one's mind and education. Second, mathematical software is older than the others, it is more mature, and some structure is emerging from the original chaos. In spite of this additional maturity (which is denied by some in other software areas), mathematical software is relatively less mature when compared to its associated application area.

I next consider how well mathematical software development meets the qualifications to be a legitimate research area. There is no doubt that there are important scientific problems to be solved. Some question whether they are difficult problems is that they believe once the numerical analysis is done, it is straight forward to transform this into high quality mathematical software and, more to the point, there are no significant difficulties beyond those of the numerical analysis. This viewpoint is especially prevalent among those who have never produced any high quality software. The COSERS panel on numerical computation research identified the top twelve accomplishments in this field since 1945. One of those is the software for ordinary differential equations developed by Gear, Shampine and others. In contrast, the fundamental work of Dahlquist on the stability of numerical methods for ordinary differential equations did not make the top twelve list. This panel of experts believed that mathematical software research can be both important and difficult and can be research of the first rank. There is not a large and widely accepted body of facts and theory in mathematical software, but this is developing. A text book with a substantial discussion of mathematical software will appear in 1981 and I expect most texts on numerical computation in the late 1980's to be a balanced blend of numerical analysis and mathematical software. I note that there are several successful methodologies for mathematical software tasks, for example, library and systematized package technology, portability of numerical software in Fortran, polyalgorithms, performance evaluation of mathematical software, preprocessors for problem areas (statistics, PDEs, linear programming, etc.) and reliability in error estimation through redundancy.

Not all mathematical software research is great or even good; its average quality is probably lower than normal for several reasons: (1) the ground rules are not yet well developed and understood, (2) the pressure of real problems forces many projects to be attempted with inadequate understanding, ability and resources and (3) many experts from other fields believe they can dash off significant software without giving it any thought or learning how it is done.

The preceding arguments are enough for me to answer with a qualified yes to the original question posed. Yet the question posed is not the real question to be answered. The real question is: Can one do significant research in mathematical software which is not in fact, numerical analysis research? To answer this question, I explore the relationship between these two specialties. First, it is clear that there is no precise statement of the content of either speciality nor is there any person or group who can say "what we do is all of numerical analysis and nothing more". Second, most workers in the general area recognize the flavor of numerical analysis. It has the taste of mathematics; definitions, theorems and proofs. Its theory is a natural decendent of what existed 35 years ago when it operated pretty much in a vacuum, very rarely was any real computation undertaken then. This flavor is still the principal one found in text books on numerical computation. On the other hand, mathematical software has the flavor of real computing; writing programs and analyzing efficiency in terms of computer or human resources used.

I see these two specialties as the theory and practice of numerical computation. There are many theoreticians who have no experience or understanding of practice and whose work has no impact on the practice. One can make important discoveries with little knowledge of the theory. Just as the American continent was discovered, so were ADI methods, adaptive quadrature, FFT (many times since the late 1800's) and the QR algorithm. All these discoveries have had important long term consequences.

However, one can do little more in mathematical software research than make chance discoveries if one does no have an understanding of and facility with the theory. The relationship here between theory and practice is similar to that in many other fields.

Theoreticians traditionally claim everything done by practitioners is just the routine elaboration of their theories. One can observe that much of the practice is the straight forward application of technical skill and not research. Theoreticians tend to forget that many things are done in practice because they are useful and not because they are novel or interesting or difficult, or research projects. However, there is also research in the practice and there is a simple test for the significance of a result; Have clever, knowledgeable and dedicated people tried to accomplish it and failed? Or, for more original work, Is it a provocative result that most such people could not achieve? Have others

recognized the work as interesting and significant? Hundreds of programs were written for solving ordinary differential equations before the current software appeared. The great superiority of the current, good software plus the multitude of previous inadequate attempts is why developing this software is regarded as one of the major research accomplishments in numerical computation.

The above test to evaluate research results completely avoids the issue raised originally. I believe there are many instances of significant research in areas that are not only not legitimate (as defined above), but areas that do not even exist (the result obtained could be the only one in the area). Thus, while I believe mathematical software is emerging as a legitimate area of research, this condition is not relevant for the evaluation of specific research results.

ON THE EFFECTIVENESS OF ITERATION FOR THE GALERKIN METHOD EQUATIONS

John R. Rice
Mathematical Sciences
Purdue University

CSD-TR 362
March 25, 1981

This paper is to be presented at the IMACS conference: Computer Methods for Partial Differential Equations, June 30 - July 2, 1981. It will appear in the proceedings: Advances in Computer Methods for Partial Differential Equations, IV, (.R. Vishnevetsky, ed.), IMACS, New Brunswick, N.J., 1981.

Post Script: It has been discovered that the Fortran compiler on the VAX was changed a few days before the data in Table 5 were measured. Considerable experimentation suggests that the execution times in Table 5 are from 75-80% of the times using the previous compiler (the one used for the other tables). To compensate one should multiply the times in Table 5 by a factor of 1.25 before comparisons are made with the data in Table 4.

ON THE EFFECTIVENESS OF ITERATION FOR THE GALERKIN METHOD EQUATIONS

John R. Rice*
Division of Mathematical Sciences
Purdue University

SUMMARY

This note reports on an experimental study of the effectiveness of matrix iteration methods when applied to the systems of linear equations obtained from the Galerkin method using bicubic Hermite polynomials for two-dimensional elliptic partial differential equations. Two iteration methods are used from ITPACK: SOR and the Jacobi Conjugate Gradient. They are compared to two direct methods: the recent LINPACK Gauss elimination routine for symmetric positive definite band matrices and a Yale Sparse Matrix Package routine. The entire study was done within the ELLPACK system for the performance evaluation of software for partial differential equations. The data shows conclusively that iteration methods are eventually (as the accuracy desired increases) more efficient than direct methods and the expected value for the cross-over point between iteration and Gauss elimination is for a 7x7 to 9x9 mesh which corresponds to an accuracy of about 0.1%. The data also shows that the LINPACK Gauss elimination routine is more efficient than the sparse matrix routine for meshes larger than 7x7. The implications for pipeline, parallel and microprocessor array computers are discussed.

I. THE EXPERIMENT

Large linear systems of equations are generated by finite element methods, such as the Galerkin method, when applied to linear elliptic partial differential equations. These systems have been solved traditionally by direct methods e.g. variants of Gauss elimination. On the other hand, iterative methods are commonly applied to the systems that arise from finite difference methods. Much of the theory for iteration methods does not apply directly to the systems generated by finite element methods and the systems are less sparse (the Galerkin system has 36 non-zero terms per equation compared to 5 for ordinary finite differences). These two facts probably explain why iteration methods have rarely been applied to linear systems arising from finite element methods. In this paper we study the effectiveness of iterative methods for such systems and conclude that the situation is similar to that for linear systems generated by finite difference methods; Gauss elimination is more efficient for smaller systems (lower accuracy), but iteration methods catch up at some cross-over point as the system size (accuracy requirement) gets higher. Furthermore, iteration methods use substantially less computer memory except for the smallest systems.

The study is experimental of the following nature: A set of 15 partial differential equations (PDEs) are chosen from the population of Rice et al[3]; their numbers are:

1-1, 4-1, 5-1, 5-4, 6-1, 7-1, 10-2, 10-3, 29-3,

41-1, 44-1, 44-2, 44-3

For the convenience of the reader these are given in the appendix of this paper. Each of these equations is self-adjoint on a rectangular domain with homogeneous boundary conditions so that the Galerkin method can be applied in a straight forward manner. The discretization is done by the program P3C1 GALERKIN written by E.N. Houstis and incorporated in the ELLPACK system[6]. The resulting systems of equations are then solved by four different methods:

SPD BAND: A LINPACK program for symmetric positive definite band matrices[1]

SOR: An ITPACK program for successive over-relaxation[5]

JACOBICG: An ITPACK program for the Jacobi method accelerated by a conjugate gradient technique[5]

YALE SPARSE: A Yale Sparse Matrix Package program which makes an LU factorization from the envelope form[1]

These programs are part of the ELLPACK system. A small study suggested that SOR and JACOBI CG are the most efficient of the five ITPACK programs in ELLPACK for these equations.

The ELLPACK system is itself part of a system for the performance evaluation of software for partial differential equations. The methodology used for this study has been presented earlier[4,7].

II. THE PERFORMANCE DATA

The basic criterion of performance is the computer time required to solve the linear system. All problems were solved on a uniform, square mesh of size NX by NX and the linear system is of order $4(NX-1)^2$. As hoped, log(time) increases linearly with log(NX) for all the problems considered so the slope of time versus NX (on a log-log scale) is taken as the primary measure of performance. In all cases, LINPACK is faster for NX=3, so there is the question of where the cross-over points lie. These are the points where iteration and Gauss elimination are equal.

Less precise but perhaps more interesting is the relationship between accuracy achieved and computer time required. We present some data for the performance measure of time needed to achieve a certain accuracy for three levels of accuracy - 5%, 0.5% and 0.05%.

The experimental data is considered in two parts. First, there is data for LINPACK SPD BAND, ITPACK SOR and ITPACK JACOBI CG from two computers: the CDC 6500 using the MNF compiler and the VAX using the UNIX (Berkeley version) compiler. The results are given in Tables 3 and 4 of the Appendix and are quite compatible. Data for the YALE SPARSE program exists only for the

*This work supported in part by the National Science Foundation, Grants MCS 76-10225, MCS 77-01408

VAX (its large memory allows a much better range of NX values) and one needs to consider both the time to solve the linear system and the time to index the equations for the sparse matrix method. This data is given in Table 5 of the Appendix.

The performances of the methods are ranked (1 is best) for each problem. These ranks are then averaged and a non-parametric statistical test[3] to see if the difference in ranks is significant.

Table 1 presents the average ranks for four measures of performances for the VAX data.

Table 1. Average performance ranks for three methods to solve the Galerkin equation on the VAX.

	Slope of time versus NX	Time to achieve accuracy of		
		5%	0.5%	0.05%
LINPACK SPD BAND	3.00	1.23	1.54	2.09
ITPACK SOR	1.77	2.46	2.38	2.46
ITPACK JACOBI CG	1.23	2.31	2.08	1.46

The differences in the average ranks of the slopes of time versus NX between LINPACK and ITPACK are significant at the 99% level. The corresponding average ranks for the CDC 6500 data are 3.00, 1.85 and 1.15 whose differences are also significant at the 99% level. In summary, the slope of time versus NX is worse for LINPACK in every case. A typical example of the plot of the data is shown in Fig. 1 at the end of this paper.

The data of Table 1 clearly show that LINPACK is better for low accuracy (the value of NX required varies from problem to problem). For the better accuracy of 0.05%, the ITPACK performance ranks improve and there is no significant difference in these ranks. The corresponding data for the CDC 6500 is similar, with LINPACK clearly better for low accuracy and ITPACK JACOBI CG pulling ahead for 0.05% accuracy.

The cross-over points occur with NX=7 to 9 for all problems except 28-3. This corresponds to computer time used on the VAX ranging from 3 to 15 seconds with 6 as the average. Problem 28-3 is extremely difficult with jump discontinuities in the coefficients of the PDE. The cross-over point in this case is at NX=17, but this corresponds to about the same level of accuracy as for the other problems. The data is again very consistant between the two computers.

The YALE SPARSE program is faster than LINPACK for small problems; the cross-over is for NX=7 or 8. For NX=9, the LINPACK program was always faster whether or not one considers the indexing step as part of the solution of the linear system. The ratio of indexing time to solution of the linear system decreases steadily as the size of the linear system grows. Samples of average values of this ratio are: NX=7 (144 equations) gives .61, N=17 (1024 equations) gives .27 and N=29 (3136 equations) gives .15. The slope of execution time versus NX (as in Figure 1) is worse for YALE SPARSE than LINPACK SPD BAND in every case.

It is well known for PDE problems that iteration methods inherently require less memory than direct methods. The advantage for finite element methods is not so dramatic as for finite difference methods (there are 36 non-zeros per equation for the particular method used rather than 5), but it still can be quite significant for large problems.

III. IMPLICATIONS FOR PARALLEL COMPUTERS.

The application of parallel computers to solving linear systems has been studied in detail[10] and it is well known that it is delicate to develop programs which achieve most of the potential speed-up of parallel computers. Without going into a detailed analysis, we note that iteration methods have an inherent speed-up advantage over direct methods for micro-processor array computers. Consider the linear system to be the simple band matrix with order N and bandwidth K. For two dimensional elliptic PDE K is about $\sqrt{N}$, for three-dimensional ones K is about $\sqrt[3]{N^2}$.

Consider first the use of vector oriented computers (e.g. CDC STAR-100, TI-ASC or Cray-1) to solve this system directly. It is difficult, perhaps infeasible in practice, to effectively use vectors of length more than K in a direct method of solution for a band matrix. Thus, it will be difficult to effectively use a vector computer with a thousand processors on most applications. Many applications do not even require K to be 100, while electronic fabrication technology is making it quite feasible to build computers with 1000 processors.

On the other hand, iteration methods allow one to effectively use vectors of length N rather than length K. This means that iteration methods potentially allow one to make full use of highly parallel machines except for relatively small problems. In the latter case the problem is quickly solved anyway.

Several groups are exploring the use of arrays of microcomputers for solving PDEs by finite element methods. The natural idea is to have one processor per element which generates the linear system; thus the microprocessor array in some sense models the physical context of the PDE. Once the linear system is generated, it is not so easy to convert the microprocessor array to an efficient machine for solving the linear system directly. However, iteration methods are naturally adaptable to such arrays, one has one processor per equation or, more likely, one processor per group of equations.

Our performance evaluation has been for sequential computations and the reasoning outlined above suggests that the relative performance of direct and iteration methods would remain unchanged for parallel computers with a low level of parallism. However, for computers which are highly parallel (involving 100s or perhaps 1000s of processors), the iteration methods have an inherent advantage which appears to make them the method of choice. This conjecture must, of course, be tested by analysis and implementation of real problems on actual machines.

IV. CONCLUSIONS.

We conclude that iteration methods for the Galerkin equations converge at rates comparable to that expected for finite difference equations. The data show that iteration methods gain in efficiency over the direct methods as the requested accuracy increases (mesh refinement becomes smaller). Thus there is a "cross-over point" where the two methods are of equal efficiency; the data suggests this occurrs for NX=7 to 9 or at accuracies of about 0.1% (we expect the cross-over to be at a lower accuracy for more difficult problems, but we have no actual evidence of this). The YALE SPARSE program is more efficient than LINPACK for small problems (NX≤7) but not for larger ones. We believe that iteration applied to the linear systems from Galerkin

methods (as well as other finite element methods) offers an inherent advantage over direct methods when one is using highly parallel computers such as arrays of micro processors.

REFERENCES

1. J.J. Dongarra, J.R. Bunch, C.B. Moler and G.W. Stewart [1979], LINPACK User's Guide, Soc. Indust. Appl. Math., Philadelphia.

2. Eisenstat, S.C., M.C. Gursky, M.H. Schultz and A.H. Sherman, Yale Sparse Matrix Package: I-The Symmetric Codes, Rpt. 112, II-The Non Symmetric Codes, Rpt. 114. Computer Science, Yale University [1977].

3. M. Hollander and D.A. Wolfe [1973], Non-parametric Statistical Methods, John Wiley, New York, Chapter 7.

4. E.N. Houstis and J.R. Rice [1980], An experimental design for the computational evaluation of elliptic partial differential equation solvers, in Production and Assessment of Numerical Software, (M.A. Hennel, ed.), pp. 57-66.

5. D. Kincaid, R.G. Grimes and D.M. Young [1979], The use of iterative methods for solving large sparse PDE-related linear systems, in Advances in Computer Methods for Partial Differential Equations, III, (R. Vishnevetsky and R. Stepleman, eds.), IMACS, New Brunswick, N.J. pp. 29-32.

6. J.R. Rice [1977], ELLPACK: A Research tool for elliptic partial differential equations software, in Mathematic Software III (J.R. Rice, ed), Academic Press, New York, pp. 319-341.

7. J.R. Rice [1979], Methodology for the algorithm selection problem, in Performance and Evaluation of Numerical Software (L.D. Fosdick, ed.), North-Holland, Amsterdam, pp. 301-307.

8. J.R. Rice, E.N. Houstis and W.R. Dyksen [1980], A population of linear, second order, elliptic partial differential equations in rectangular domains, Part I and II. Mathematics Research Center Reports 2079 and 2080, University of Wisconsin.

9. J.R. Rice, E.N. Houstis and W.R. Dyksen [1981], A population of linear, second order, elliptic partial differential equations on rectangular domains Mathematics of Computation, 35 (1981) pp.

10. R.G. Voight [1977], The influence of vector computer architecture on numerical algorithms, in High Speed Computer and Algorithm Organization (D. Kuck, ed.), Academic Press, New York, pp. 229-344. See also: Solution of partial differential equations on vector computers, Proc. 1977 Army Numer. Anal. Corp. Conference, ARO Report 77-3, pp. 475-525.

APPENDIX: THE PDEs AND THE DATA

The partial differential equation problems used for this study are listed below. The domain for each problem is the unit square $0 \le x,y \le 1$ and the boundary conditions are all homogeneous i.e. $u(x,y) = 0$ on the boundary. In each case the forcing term $f(x,y)$ is determined to produce a particular true situation.

1 -1 $(e^{xy}u_x)_x + (e^{-xy}u_y)_y - u/(1+x+y) = f$

4 -1 $u_{xx} + u_{yy} = f$

5 -1 $4u_{xx} + u_{yy} = f$

5 -4 $4u_{xx} + u_{yy} + 10u = f$

6 -1 $u_{xx} + u_{yy} - (100+\cos(3\pi x) + \sin(2\pi y))u = f$

7 -1 $u_{xx} + u_{yy} = 1$

10-2 $u_{xx} + u_{yy} = f$

10-3 $u_{xx} + u_{yy} = f$

28-3 $(wu_x)_x + (wu_y)_y = 1$ where $w = 100$ for $0 \le x,y \le .5$, $= 1$ otherwise

41-1 $u_{xx} + u_{yy} + 10u = f$

44-1 $u_{xx}+u_{yy}+wu = w$, $w = -2.030625\, e^{(r/(1+r/2))}$, $r(x,y)$ tabulated

44-2 $u_{xx}+u_{yy}+wu = w$, $w = -100\, e^{(r/(1+r/2))}$, $r(x,y)$ tabulated

44-3 $u_{xx}+u_{yy}+wu = w$, $w = -2.030625(1-r)e^{(r/(1+r/25))}$, $r(x,y)$ tabulated

The performance data collected is listed in Tables 3,4 and 5 for each problem. MESH refers to the number of mesh lines including the boundary, ACCURACY is the maximum error measured on a 20x20 grid of points. The execution times in seconds for a CDC 6500 or DEC VAX are given for solving the system of linear equations with each of three methods.

Table 3. The performance data for the CDC 6500. The SPD BAND solution times are independent of the PDE and are 0.06, 0.65, 2.18 and 3.52 seconds, respectively for 3x3, 5x5, 7x7 and 8x8 meshes.

	PDE 1-1			PDE 4-1			PDE 5-1		
MESH	ACCURACY	SOR	JACOBI	ACCURACY	SOR	JACOBI	ACCURACY	SOR	JACOBI
3x3	9.5E-3	0.16	0.39	3.0E-3	0.15	0.19	2.5E-2	0.18	0.31
5x5	7.7E-4	1.06	1.45	2.8E-4	1.02	1.10	1.4E-3	1.06	0.85
7x7	1.6E-4	2.79	3.32	5.6S-5	2.56	2.74	4.8E-4	2.91	2.61
8x8	1.0E-4			3.3E-5			2.5E-4		
	PDE S-4			PDE 6-1			PDE 7-1		
3x3	2.7E-2	0.18	0.31	3.3E-0	0.26	0.30	7.1E-4	0.18	0.17
5x5	1.4E-3	1.05	0.88	2.8E-2	1.09	0.76	1.2E-4	0.99	0.66
7x7	4.8E-4	3.01	2.52	5.5E-3	2.55	2.20	3.4E-5	2.58	2.05
8x8	2.5E-4						3.2E-5		
	PDE 10-2			PDE 10-3			PDE 28-3		
3x3	5.2E-2	0.13	0.17	6.3E-2	0.13	0.17	4.1E-2	0.40	0.19
5x5	6.9E-3	0.86	0.65	8.7E-3	0.83	0.70	1.2E-2	1.99	1.57
7x7	9.1E-4	2.30	2.18	5.0E-3	2.21	2.27	1.3E-2	5.03	4.09
8x8	1.4E-4			1.2E-3					
	PDE 41-1			PDE 44-1			PDE 44-2		
3x3	6.4E-2	0.20	0.17	1.0E-3	0.17	0.17	4.3E-1	0.47	0.21
5x5	1.4E-3	1.51	0.68	2.4E-4	1.03	0.68	1.5E-2	1.29	0.78
7x7	1.4E-3	2.81	2.03	6.9E-5	2.61	2.20	4.5E-3	2.98	2.23
8x8				5.8E-5			2.8E-3		
	PDE 44-3								
3x3	1.6E-3	0.17	0.17						
5x5	2.4E-4	1.05	0.68						
7x7	6.9E-5	2.61	2.22						
8x8	5.7E-5								

Table 4. The performance data for the DEC VAX. The SPD BAND solution times (in seconds) are independent of the PDE and are listed first in a short table.

NX	3	5	7	8	9	11	13	15	17	19	21	23	25	27	29
Time	0.1	0.9	3.2	5.3	8.3	17.5	33.2	57.7	93.2	143.0	211.5	300.5	414.8	561.1	745.2

	PDE 1-1			PDE 4-1			PDE 5-1		
MESH	ACCURACY	SOR	JACOBI	ACCURACY	SOR	JACOBI	ACCURACY	SOR	JACOBI
3x3	9.5E-3	0.3	0.4	3.0E-3	0.3	0.2	2.5E-2	0.3	0.4
5x5	7.7E-4	1.4	1.8	2.8E-4	1.3	1.5	1.4E-3	1.4	1.3
7x7	1.6E-4	3.6	4.6	5.5E-5	3.4	3.8	4.8E-4	3.7	3.8
9x9	6.3E-5	8.0	8.2	2.1E-5	7.8	8.1	1.1E-4	8.3	8.1
13x13	1.1E-5	29.3	25.9	4.1E-6	29.0	23.4	4.2E-5	25.0	25.0
17x17	3.6E-6	75.8	61.1	1.4E-6	51.8	51.0	1.9E-5	71.3	47.9
21x21	2.3E-6	111.1	119.6	1.1E-6	113.1	96.8	7.3E-6	121.5	86.6
25x25	4.0E-6	181.1	207.7	2.0E-6	213.1	173.8	5.2E-6	181.9	149.1
29x29	6.4E-6	317.1	353.4	1.1E-6	349.2	262.2	3.8E-6	287.6	231.0
	PDE 5-4			PDE 6-1			PDE 7-1		
3x3	2.7E-2	0.3	0.4	3.3E-1	0.4	0.4	7.1E-4	0.2	0.2
5x5	1.4E-3	1.3	1.3	2.8E-2	1.7	1.1	1.2E-4	1.2	1.0
7x7	4.8E-4	3.9	3.9	5.5E-3	3.5	3.4	3.4E-5	3.3	3.0
9x9	1.1E-4	8.3	8.3	1.8E-3	6.7	7.1	2.9E-5	7.8	6.6
13x13	4.0E-5	24.8	22.7	3.9E-4	17.6	18.2	6.1E-6	30.5	19.1
17x17	1.4E-5	64.7	47.2	1.8E-4	35.0	32.3	9.3E-7	74.2	39.6
21x21	8.6E-6	117.4	84.9	6.7E-5	67.0	56.6	1.3E-6	110.8	81.5
25x25	1.0E-5	187.4	141.7	3.5E-5	116.4	96.7	4.5E-7	202.7	140.7
29x29	4.0E-6	282.3	234.9	2.4E-5	195.7	149.3	5.3E-7	305.0	202.6

MESH	PDE 10-2 ACCURACY	SOR	JACOBI	PDE 10-3 ACCURACY	SOR	JACOBI	PDE 28-3 ACCURACY	SOR	JACOBI
3x3	5.2E-2	0.3	0.3	6.3E-2	0.3	0.2	4.1E-2	0.9	0.3
5x5	6.9E-3	1.2	1.0	9.7E-3	1.2	1.0	1.2E-2	2.9	2.6
7x7	9.1E-4	3.2	3.2	5.0E-3	3.1	3.6	1.3E-2	6.5	6.8
9x9	1.7E-4	7.9	6.9	1.2E-3	7.3	6.8	2.0E-2	11.0	12.3
13x13	7.2E-5	27.2	20.7	2.0E-4	26.5	20.5	3.0E-2	30.9	35.3
17x17	4.1E-5	60.8	41.2	1.0E-4	62.0	42.8	2.9E-2	55.3	65.3
21x21	1.9E-5	118.1	70.6	4.7E-5	120.9	76.3	3.3E-2	119.2	119.0
25x25	6.3E-6	218.9	118.9	1.4E-5	197.3	127.5	3.5E-2	214.5	209.5
29x29	1.2E-6	271.7	170.7	2.8E-6	302.6	184.8	3.7E-2	294.2	327.7

MESH	PDE 41-1 ACCURACY	SOR	JACOBI	PDE 44-1 ACCURACY	SOR	JACOBI	PDE 44-2 ACCURACY	SOR	JACOBI
3x3	6.4E-2	0.4	0.3	1.6E-3	0.3	0.3	4.3E-1	0.8	0.3
5x5	1.4E-3	1.4	1.0	2.4E-4	1.3	1.0	1.5E-2	1.6	1.2
7x7	1.4E-3	3.8	3.0	6.9E-5	3.6	3.4	4.5E-3	4.1	3.7
9x9	1.4E-3	6.6	6.2	5.1E-5	7.6	7.4	2.7E-3	7.1	7.7
13x13	1.4E-3	16.2	16.2	3.9E-6	28.7	20.1	5.7E-4	16.4	19.2
17x17	1.4E-3	33.5	32.0	1.0E-5	77.9	41.0	6.3E-4	35.4	34.0
21x21	1.4E-3	61.2	51.5	1.1E-5	111.9	72.5	7.0E-4	56.2	52.7
25x25	1.4E-3	102.1	87.6	8.1E-6	204.5	123.4	7.2E-4	95.5	89.0
29x29	1.4E-3	165.0	144.5	7.6E-6	324.0	216.0	4.7E-4	135.6	135.5

MESH	PDE 44-3 ACCURACY	SOR	JACOBI
3x3	1.6E-3	0.3	0.3
5x5	2.4E-4	1.3	1.0
7x7	6.9E-5	3.4	3.4
9x9	5.1E-5	8.0	5.7
13x13	5.8E-6	28.7	19.5
17x17	1.1E-5	78.4	40.7
21x21	1.1E-5	109.6	71.1
25x25	8.4E-6	202.1	137.5
29x29	8.0E-6	320.3	187.1

Table 5. The performance data of YALE SPARSE for the DEC VAX. The execution time (in seconds) for indexing the linear equations is Time 2; the time to solve the linear system is Time 3.

MESH	PDE 1-1 Time 2	Time 3	PDE 4-1 Time 2	Time 3	PDE 5-1 Time 2	Time 3	PDE 5-4 Time 2	Time 3
3x3	.1	.1	.1	.1	.1	.1	.1	.1
5x5	.6	.8	.7	.9	.6	.8	.5	.8
7x7	1.9	3.1	1.8	2.9	2.1	4.4	2.1	3.1
9x9	4.9	9.8	5.9	12.9	4.8	9.3	4.5	10.1
13x13	12.5	39.5	14.3	46.0	14.3	45.9	13.2	42.0
17x17	31.5	122.6	38.2	152.3	32.8	130.9	33.8	129.4
21x21	51.4	254.7	54.3	238.2	53.6	242.1	68.0	332.1
25x25	87.4	527.9	96.4	496.3	98.2	439.2	90.8	430.5
29x29	115.0	789.0	137.2	959.6	117.7	742.8	160.0	>999

MESH	PDE 6-1 Time 2	Time 3	PDE 7-1 Time 2	Time 3	PDE 10-2 Time 2	Time 3	PDE 10-3 Time 2	Time 3
3x3	.1	.1	.1	.1	.1	.1	.1	.1
5x5	.7	.9	.7	.9	.7	.8	.6	.8
7x7	2.0	3.2	2.0	3.2	1.9	3.1	1.8	2.8
9x9	5.0	9.8	5.9	12.2	5.8	12.2	5.7	11.8
13x13	13.3	38.6	14.2	46.0	14.4	46.0	15.2	48.4
17x17	30.9	109.6	38.0	142.2	39.2	152.6	40.6	154.9
21x21	50.0	232.3	55.6	242.0	53.9	237.2	54.2	242.6
25x25	89.3	453.3	92.3	492.2	100.4	493.0	96.0	491.6
29x29	122.4	801.8	138.3	959.2	137.1	939.0	138.3	959.3

MESH	PDE 28-5 Time 2	PDE 28-5 Time 3	PDE 41-1 Time 2	PDE 41-1 Time 3	PDE 44-1 Time 2	PDE 44-1 Time 3	PDE 44-2 Time 2	PDE 44-2 Time 3
5x3	.1	.1	.1	.1	.1	.1	.1	.1
5x5	.6	.8	.5	.8	.6	.8	.6	.7
7x7	1.9	3.4	1.7	3.1	1.8	3.1	1.9	3.1
9x9	4.9	8.6	5.0	8.8	4.8	11.0	4.2	9.0
13x13	12.9	35.9	15.0	49.0	14.0	49.2	13.2	40.9
17x17	35.1	115.2	35.5	139.1	29.4	106.0	29.4	105.6
21x21	55.6	239.1	58.1	252.4	48.7	221.1	50.8	274.2
25x25	104.6	523.3	102.3	485.5	84.0	482.1	79.5	400.3
29x29	135.9	878.0	116.5	688.6	113.3	693.3	113.8	736.4

MESH	PDE 44-3 Time 2	PDE 44-3 Time 3
5x3	.1	.1
5x5	.6	.8
7x7	2.0	3.1
9x9	5.0	11.1
13x13	13.1	36.0
17x17	55.5	125.2
21x21	54.9	250.8
25x25	95.5	511.8
29x29	139.5	756.0

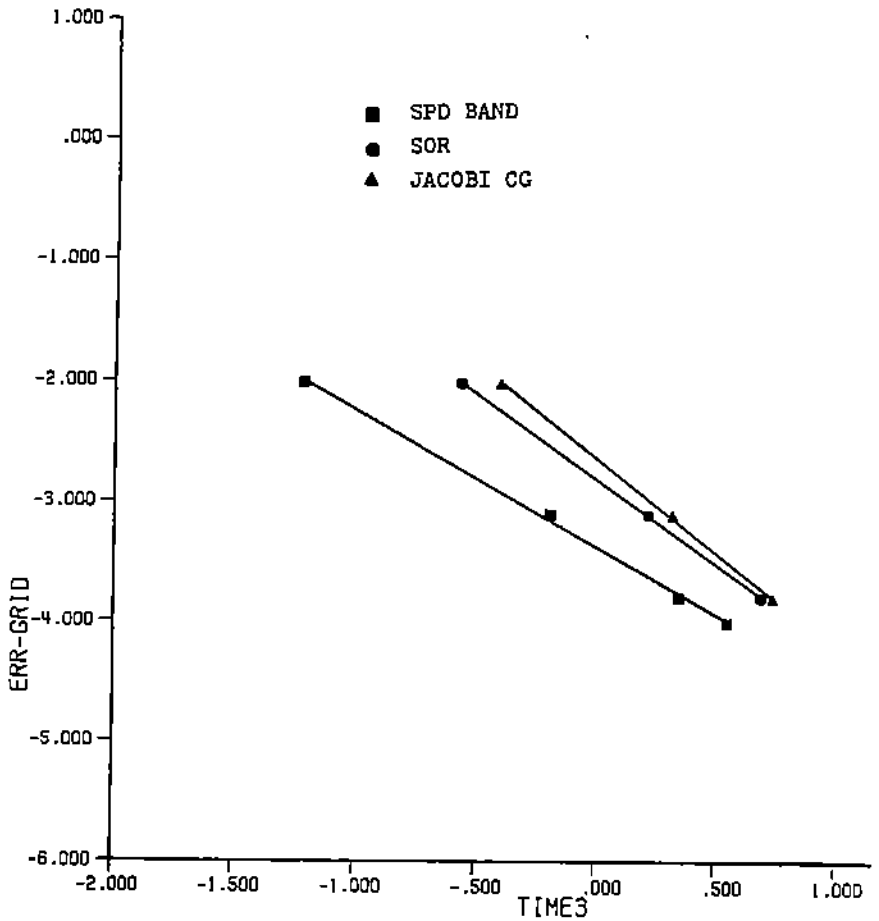

Figure 1. A typical set of data comparing the linear equations solution time for iterative and direct methods. The plot is log of error on a 20x20 grid versus log of time in seconds.