

# Aspects of Scientific Computing

Gene H. Golub

*Department of Computer Science*

*Stanford University*

*Stanford, CA 94305-9025 USA*

*email: golub@sccm.stanford.edu*

Text of the 1995-1996 Johann Bernoulli Lecture given at the University of Groningen on April 8, 1996. The Johann Bernoulli Foundation for Mathematics, founded in Groningen in 1988 to promote mathematics, organizes each year a Johann Bernoulli Lecture for which it invites prominent scientists. Johann Bernoulli was Professor of Mathematics at the University of Groningen from 1695-1705.

## INTRODUCTION

It is a great honor to be here and to speak in this forum. Today has been a wonderful day here in Groningen and I have just been enormously stimulated. Not only is it a great honor to speak in a series named after Johann Bernoulli, but also because some of my predecessors. I want to show the first thing however how I obtained some information about Bernoulli: that was over the Internet. It is always possible now to get as much information as you like. In fact the amount of information that one can obtain is overwhelming, and since I thought to know more about Bernoulli I was able to get this directly from some location by just using a search algorithm. I ended up on the following website, which gives an informative description of Johann Bernoulli: [http://www.vma.bme.hu/mathhist/Mathematicians/Bernoulli\\_Johann.html](http://www.vma.bme.hu/mathhist/Mathematicians/Bernoulli_Johann.html).

Former Bernoulli lectures were given by Kalman (1992), Zeeman (1993), Halmos (1994) and Lenstra (1995). I am very much delighted to speak in a forum where they have spoken and I'd like to say something about these people.

First of all let's note that Kalman and Halmos and Lenstra have all taught and lived in the United States. They were not born there, but they came to the United States for extended periods. Kalman was born in Hungary, Halmos was also born in Hungary and Lenstra is of course a native of Holland. That seems very much in the spirit of Johann Bernoulli, because he spent ten years here before he returned to Switzerland. Mathematicians have a long tradition in being para-patetic. For instance, Euler lived in Russia for many years, Descartes has lived here in Holland as well as in France, Goldbach has lived everywhere.

The US has been very fortunate in this century by the number of immigrants that have come to the US. I myself was born in Chicago, but my parents were immigrants from eastern Europe. Halmos and Kalman, as I say, were both born in Hungary, but they received much of their education in the US. Kalman got his PhD from Columbia and spent years at several universities including Stanford (he was at Stanford for several years). Currently he has a position in Florida and at ETH in Zürich. Halmos grew up in Chicago too; he received his PhD at the University of Illinois and he now lives in California. So my life scenario is very similar to his.

## SCIENTIFIC COMPUTING: HISTORY AND APPLICATIONS

Today what I am going to talk about is scientific computing, and this is a subject where many Hungarian mathematicians have made a contribution. Three names which really played a very strong role in scientific computing are Cornelius Lanczos, John von Neumann and Peter Lax. Cornelius Lanczos was born in the nineteenth century in Hungary. In the 30-ies he spent sort

of an oscillatory time between Hungary and the US. During the war years he lived in the US and afterwards he went to Ireland since he was concerned about the political situation in the US. John von Neumann spent most of his adult life in the US. He was an enormously influential mathematician; I'll say more about him in just a moment. Of the three Peter Lax would be the youngest. He is fortunately still alive and his contributions to the numerical solution of differential equations cannot be underestimated<sup>1</sup>.

### *Profile of scientific computing*

So what do I mean by scientific computing. As I regard it, scientific computing is a synthesis of applied mathematics, numerical analysis and computer science. Each one of these areas could be defined more precisely. I am not going to say what they are, but we take the broadest view of each of these topics. For instance, applied mathematics not only covers the traditional area of applied mathematics, but also more contemporary topics such as signal processing and control theory. I think the SIAM journals have displayed a broad interest in applied mathematics in the US and elsewhere.

The other aspect of scientific computing that makes it somewhat different from other topics that are often studied is that it is driven by technology and it is driven in two ways. First of all many of the new scientific problems that arise come about because of technology, that is there are new problems that are constantly arising leading to different kinds of classes of mathematical problems. And secondly there is the computer revolution, which has now continued through fifty years. It is not abided that I am constantly amazed by what is coming next and it seems to be impossible to foretell. Each new development has allowed us to develop new techniques and get more insightful information about computation. Numerical experimentation has increasingly become more sophisticated and enables us to understand our computational experience to a higher degree.

### *Von Neumann*

Let me return to one of my heroes, I should say, and that is John von Neumann. Von Neumann (1903-1957) occupied a position at the Institute for Advanced Study and in a sense he is the great scientific computer. Here are just a few of the contributions that he made in the 54 years that he lived.

First of all he planned: he wrote a series of papers about planning and coding of problems. He actually described how one would go about some problems. He designed computers. Some of the earliest computers that were designed were designed by Von Neumann; they were called Von Neumann machines frequently. With Goldstine he analyzed the solution of linear equations. One of the earliest analyses of how good is Gaussian elimination was done by Von Neumann and Goldstine. Not only that, he developed iterative methods for solving equations and I'll say more about that in just a little while. He developed methods for hydrodynamical calculations, and he actually identified numerical fluid dynamics as a major area of interest. Because he was such a well known person, it was important that he promoted and explained the relevance of computers and computation. So he played truly a terrific role.

George Dantzig, who was one of my colleagues at Stanford, told me that it was Von Neumann who explained to him the importance of duality theory in linear programming; he was really responsible for that. By the way, he was not perhaps the originator of Monte Carlo methods, but in the development of Monte Carlo methods he played an important role, and he used to go around thinking of new ways of generating random numbers. Unfortunately not all of them worked very well, but some of the early methods of generating random numbers were due to Von Neumann. And there are many, many other contributions.

---

<sup>1</sup>Note added Spring 2006: In 2005, Lax was awarded the Abel Prize.

Unfortunately, he died at an early age, he died of cancer, while a commissioner for the Atomic Energy Commission, and I think we all suffer for that.

#### *Matrix computations in pre-electronic days*

My interest is in matrix computations. It is not a very sophisticated subject necessarily, but it is a topic that arises in almost any scientific computation. And what I want to explain to you today is just some of the things that we do in matrix computations. Some of the problems, some of the techniques that have evolved. There is a long history of matrix computations and it comes off in so many different ways.

A lot of things were done using hand calculators. For instance there was Southwell. He was a British engineer, developed relaxation methods, and when told that maybe relaxation methods could be used on computers he dismissed it. He did not think that these techniques could be used in an automatic way, but they required rooms full of young women working at desk calculators. Statisticians and also psychometricians developed some of the early techniques: e.g. Burt in Britain and Thurstone in the US. Interestingly enough, these psychometricians developed methods that today have become quite important in signal processing. Some of these methods already were developed by Jacobi. One of the methods we use even today for getting eigenvalue estimates. Gauss played a leading role and I refer to that in just a moment. Hotelling was a famous statistician who made some estimates about computations that were unfortunately overly negative and pessimistic.

Already we've talked about some of the applications that arise in scientific computing, like structural engineering, statistical data analysis (especially econometrics and psychometrics), plasma physics, signal processing, solution of elliptic equations and geodesy. Of course the list goes on endlessly and one is not suspecting of how they come about.

#### *Matrix structure*

What I am going to talk about in part is solution of linear equations. So this is an old classical problem. I'll talk more about it when we pass along. But let me tell you about why it becomes a complicated subject. Of course everybody learns at school if you want to solve  $Ax = b$  then invert  $A$ , multiply it by  $b$  and you are finished. The problem is more complicated than that.

So the first consideration of solving a linear system of equations is 'what is the source of the problem, where did it arise'. Frequently it comes about from a discretization of a partial differential equation or system of equations. I have had a long discussion with people today just about such topics. Another source is data-fitting problems and I'll mention one in just a moment that Gauss looked at in the 18th century. And then of course there's a considerable interest in Markov chains. So there are many different sources of problems where linear equations arise.

An important role in any numerical method for solving a system of equations is played by the structure of  $A$ . We can ask all kinds of questions, e.g. is it a sparse matrix. The very large problems are usually matrices with almost all zeroes. Nevertheless they are very difficult to solve. When they are sparse, does the sparsity come about in a structure as you often have in a partial differential equation, or is it a random sparsity? We have lots of names associated with different kinds of matrices: Toeplitz matrices, Hankel matrices, p-cyclic matrices, . . . . And each one of these types of matrices have special algorithms that one can use. So constantly in scientific computing one is developing new algorithms that we hope will be satisfactory for certain classes of problems. Other properties we look at: is the matrix symmetric, is it positive definite, is it indefinite, is it complex symmetric? Those matrices are continually arising.

#### *Other considerations*

What are the computing resources? Are we going to use a supercomputer where we are able to do many operations in parallel, or are we going to use a computer where vector operations

are very simply done? What kind of computing resources? Many of us use workstations today. Should we use workstations in a distributed manner? These are all important issues.

There are other considerations too. In many problems it is not one right-hand side that you are solving, but you have several right-hand sides, and even there we have different situations. Sometimes these right-hand sides occur sequentially, sometimes they are given all at one time.

The matrix itself may change. You may have low-rank corrections, or there may be small perturbations. Is the entire solution required? You see classically what we do in our discussion of solving linear equations is we just talk about finding the entire solution. But in many situations it is not necessary to find the entire solution but only some linear combinations of the components of the solution are of interest. And of course, we may not need fifteen significant digits to every solution, we may only need a few digits. I am not going to talk about all of these issues. I am just alerting you to the fact that these are issues of importance; they come up very often.

One of the most important areas of research has been the problems where we have small changes in our matrices, and of updating and downdating. Algorithms which can make use of these ideas of making small changes in the matrix and then quickly finding a new solution, those are of some importance in many applications.

### *Least squares*

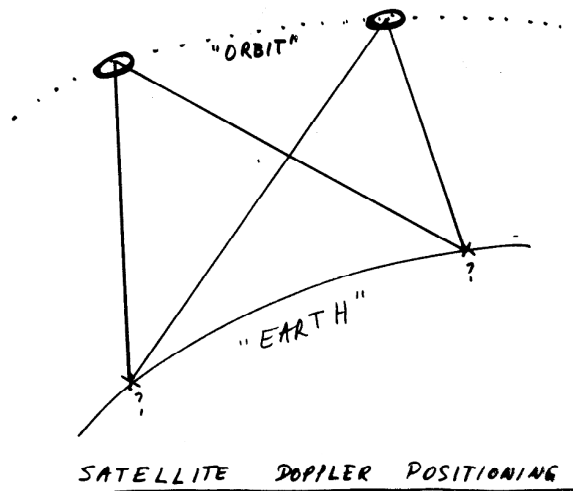
Now I want to refer to an example given by Gauss (1777-1855), just to show you that some of the problems that we look at today have been studied previously. Gauss wrote his 'Theoriae Combinationis Observationum Erroribus Minimus Obnoxiae' originally in Latin, and recently this monograph of Gauss has been translated by G.W. Stewart [1]. Those of you who want to look at the work of a great master can find this in an easier published form. So let's mention what Gauss was interested in looking at. He took – he was describing solving nonlinear least-squares problems – some data of De Krayenhof, who in 'Précis historique des opérations trigonométrique faites en Hollande' had described data linking various cities in the Netherlands. With reference to the map shown, these cities were Harlingen, Sneek, Oldeholtspade, Ballum (Ameland), Leeuwarden, Dokkum, Drachten, Oosterwolde and Groningen. The first idea was that you could triangulate these cities: here is for instance triangle 131, giving the triangulation of three cities including Groningen:

Dokkum	57° 1' 55.292"
Drachten	83° 33' 14.525"
Groningen	39° 24' 52.397"



The next idea was to take these observations and then – the data was not exact – to recompute the data and get a more exact location of each one of these cities. You can see the typical kind of information that he had. So one knew the angles, and you can see the kind of measurements that were made. The idea was to adjust these triangles to get a more perfect fit.

This kind of problem still comes up, and in fact the application that I will show next comes about from satellites. We get enormous amounts of data, each station there are three spatial coordinates that are known only approximately and we have parameters describing the orbit, and we have three velocity components. What we are trying to locate is the exact position of each one of these stations on the earth.



What this leads to, I want just to illustrate the point, this leads to a big matrix problem.

$$A = \begin{pmatrix} B_1 & & & C_1 \\ & B_2 & & C_2 \\ & & \ddots & \vdots \\ & & & B_m & C_m \end{pmatrix} \equiv (B, C),$$

$$\text{where } B_i = \begin{pmatrix} B_{i1} \\ B_{i2} \\ \vdots \\ B_{im} \end{pmatrix}, \text{ and } C_i = \begin{pmatrix} C_{i1} & & & \\ & C_{i2} & & \\ & & \ddots & \\ & & & C_{im} \end{pmatrix}.$$

Here  $m$  is the number of satellite passes,  $k$  counts the orbital parameters,  $n$  is the number of ground stations, and  $p$  counts the unknown coordinates of a ground station. The matrices  $B_{ij}$  and  $C_{ij}$  are of size  $(k + p) \times k$  and  $(k + p) \times p$  respectively. The number of unknowns equals  $mk + np$ , whereas the number of equations is  $mn(k + p)$ . The space in the matrix  $A$  contains just zeroes, and then the question is how to solve these systems of equations so that you get a very accurate reading. The problem and even much of the technique goes back to Gauss. But yet we want to incorporate his ideas and more modern ideas in a computing environment: what is the way for solving these problems. We'll come back to this later.

### Acceptance

The unfortunate fact that I have to tell you about scientific computing is that sometimes those of us who are in academia sit back. We can actually devise very elegant algorithms, but then having those algorithms used by others is sometimes a very difficult matter. There is a gap, as always in the academic world, between the generation of ideas and the acceptance of those ideas by others.

### SOLVING LINEAR EQUATIONS IN MODERN COMPUTING

What I want to talk about now is solving linear equations. One of the earliest papers on this topic was done by Von Neumann, as I mentioned earlier. He and Goldstine published a long monograph analyzing Gaussian elimination (in fixed point arithmetic) as well as using the Jacobi

method for solving systems of linear equations. Jacobi actually developed his algorithm for computing eigenvalues. It's a very simple algorithm. It was used in the early days of computing and then after this method was developed, improved upon, analyzed, and it fell by the way side. Why? Because other people had come about and developed more interesting algorithms or algorithms that were somewhat faster. These algorithms are generally associated with the names of Givens and Householder. Then this method of Jacobi went out of use, and for many years it was not seriously considered until parallel computers came along, and then it became again the method of choice. So this is one of the things that comes about very often in scientific computing: methods come into favor, they fall out of favor and then they return to favor.

Therefore, in any educational process it is necessary to give our students a broad background. Some folks say that you only need to learn how to use a black box, and I don't think that's really enough. I think we really have to make our students understand what scientific computing is about from elementary principles and let them gain a broader understanding of scientific computing.

### *Error analysis*

The man of the period we did talk about so much is J.H. Wilkinson. He was British and member of the Royal Society; he died approximately 10 years ago. He is really a very fine fellow with a great sense of humor. He developed tools for analyzing floating point computations: he developed backward error analysis for Gaussian elimination and other matrix computations. So he played a leading role, and people as myself have used these tools that he developed for analyzing other algorithms.

Let me just state very simply from a mathematical point of view what he did and the kinds of things he was able to do. He developed a nice notation

$$fl(a \text{ op } b) = (a \text{ op } b)(1 + \varepsilon), \text{ where 'op' indicates a numerical operation.}$$

What does that mean? It means that if you perform a computation on a computer, if you add  $a$  and  $b$  together, then you really are adding two numbers together, but they are not the numbers you began with very often. In other words, the effect of doing inexact arithmetic on a computer is to really add two numbers together, but they are different from the numbers that you started with in many cases. However you can bound how large that relative error will be. So what Wilkinson then showed, and this is the key point in fact with error analysis, is that instead of solving  $Ax = b$ , you are solving  $(A + E)y = b$ . In other words, the original matrix is perturbed by some small amount. That small amount depends upon the arithmetic, the type of interchanges one uses and a few other things, but basically, and this is the important point of Wilkinson, you can bound the norm of the matrix  $E$ . He gave some bound of that nature.

That bound in itself is not important, but knowing that it exists is quite important. So this work of Wilkinson was fundamental to our understanding how to use Gaussian elimination and solving linear equations. I was reading some biographical notes the other day, and a Czech mathematician said: 'Well after reading Goldstine and Von Neumann's paper they thought that for no matrix of order over a hundred would it ever be possible to solve. It turns out because of Wilkinson's analysis one can solve much larger systems. Well, independent of his analysis, the pessimistic view was that you cannot use these direct methods of solving linear equations because of the growth of the error. Nevertheless very large systems, systems of order 10-20 thousand which are filled-in, are solved on a regular basis now.'

### *Golden fifties*

Let me go back a bit more, and let me say something about iterative methods, which is a particular interest of my own. The great period in numerical linear algebra was the 'golden fifties'. Numerical analysis was able to draw upon some of the best minds for developing new

algorithms. I feel that at any given time there is much restless energy in the scientific community, and as a field opens up these fields draw upon people with creativity and ideas, and they are drawn to a field which is rapidly developing. Some names: Hestenes, Stiefel, David Young. Von Neumann is instrumental in this list too. These were people who were very creative, having great knowledge and then were able to develop methods.

### *Conjugate gradients*

So, for instance, the method that is so frequently used today is called the conjugate gradient method. It was developed independently by Hestenes, who was an American mathematician of Norwegian descent, and Edward Stiefel. Stiefel was from ETH in Zürich. He was actually working in topology early on, but after the war, I think, he felt that developments had passed him by, and so he turned his head to numerical computing. Hestenes and Stiefel met at a meeting. They both had discovered they had certain common ideas, and together they wrote a classic paper on the conjugate gradient method.

Here too is a very interesting story, because the method was a great excitement to the community at first, but then the numerical properties and the mathematical properties were not the same. The mathematical theory said that in a finite number of iterations this algorithm would converge, and unfortunately it did not converge in the way it was anticipated. The method lost credibility, so it was dismissed. It could not solve 50 or 100 equations in 50 or 100 iterations. After it was dismissed it fell into disuse until it was revived by engineers who needed to solve large systems of equations. Now people began to understand it better, they used it on larger problems where it became the method of choice. Again you see this idea: a method is first one is excited about, falls into disuse, it returns (maybe with a different understanding from what originally was anticipated) and it comes into play again.

### *Preconditioning*

Next let us discuss another idea that was quite important, and this is a part where my colleague Richard Varga played an important role. It is often said an applied mathematician is someone who solves the problem that you are almost interested in, and this is exactly the idea of splitting and preconditioning. Somehow you have a matrix. This matrix can then be broken up into the sum of two matrices, one of which you can easily solve. A very key point now is how to determine the splitting, or the preconditioner. This enables you to sort of accelerate the convergence greatly. Today I had a number of discussions on how to construct preconditioners. Sometimes the problem itself tells you how to construct the preconditioner, other times using algebraic techniques one wants to construct the preconditioner. So this is the basic idea. You break up your problem in something you can easily solve and then iterate.

Today there is a whole alphabet of methods that are used: BiCG, BiCGStab( $\ell$ ), CG, CGS, GMRES( $m$ ), GMRESR, ICCG, ILU, ILUM, MICCG( $\ell$ ), MILU( $\ell$ ), MINRES, MRILU, SYMMLQ, QMR, . . . . The reason I want to show you these: many of these ideas have developed here in the Netherlands. You are very fortunate having professor Van der Vorst at Utrecht University who has created a whole school of people about him who are constantly developing new methods with additional letters, even with subscripts and so forth. Nevertheless it is a very active group of people in Utrecht, who have developed many excellent ideas on iterative methods for solving linear systems of equations. They are responsible for several of the methods that are given here.

### *Singular-value decomposition*

The next idea that I want to discuss has to do with picture processing. Think about a checkerboard or any kind of mesh, and suppose you have a letter S, so you can see those x-es represent an S.

8								
7			×	×	×	×		
6			×					
5			×	×	×	×		
4						×		
3			×			×		
2			×	×	×	×		
1								
	1	2	3	4	5	6	7	8

pixels  $z_{ij} = 0, 1, \dots, 7$ ;

$i, j = 1, 2, \dots, m$ ;

$m^2 = N$ ;  $N = 64$

So we have pixels and those pixels can take on a certain set of integers, you have an array of that sort. What we want to do is recreate that picture in some fashion. You will see more clearly in one moment how that comes about. Now often the information is combined in some way, and there is some noise added to it in such a way that it is impossible to really see the information without doing some processing:

$$\text{filtering: } y_{ij} = \frac{1}{9} \sum_{k=i-1}^{i+1} \sum_{l=j-1}^{j+1} z_{kl} \equiv Hz,$$

$$\text{noise added: } y = Hz + e, \quad \text{with } e \sim N(0, \sigma^2 I).$$

So  $H$  is this filtering that goes on to the data. And in some fashion we want to recreate the original letter S – we will see this in just a moment. It can be shown, mathematically it is called a least-squares solution, that you need to construct the pseudo inverse of the matrix  $H$ :

$$\text{least-squares problem: } \text{minimize } \|y - Hz\|,$$

$$\text{solution: } \hat{z} = H^+ y, \quad \text{where } H^+ \text{ is the pseudo inverse of } H.$$

The solution to this is given in terms of what is called the singular-value decomposition. I just mention this now and a little further on. A singular-value decomposition is a decomposition of a matrix which breaks it up into rank-one matrices. The mathematical properties of the singular-value decomposition can be spelled out as

$$H = \sum_{i=1}^N \sigma_i u_i v_i^t, \quad \sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_N \geq 0,$$

$$u_i^t u_i = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases}, \quad v_i^t v_i = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases}.$$

If a matrix  $H$  has a given rank  $r$  than the  $(r + 1)$ st singular value  $\sigma_{r+1}$  is equal to zero. It's pseudo inverse looks like

$$H^+ = \sum_{i=1}^r \frac{1}{\sigma_i} v_i u_i^t.$$

You'll notice it reciprocates the nonzero singular values of the matrix. In this way the least-squares solution of  $Hz = y$  becomes

$$\hat{z} = \sum_{i=1}^r \frac{1}{\sigma_i} v_i (u_i^t y).$$

If  $\sigma_{p+1} \leq \varepsilon$ , then it is appropriate to compute

$$\tilde{z} = \sum_{i=1}^p \frac{1}{\sigma_i} v_i (u_i^t y).$$

And then often what happens is, one really has to make a decision about the smallest singular value. Here we face one of the most difficult problems in numerical linear algebra: what is zero. That means that we have determined a priori some parameter, and that will determine a numerical rank of this matrix. And then it becomes a rather subtle problem how to determine zero within a computer. You can take two nearby numbers, subtract them and maybe you'll get zero or maybe you get some other number. Numerical analysts and people in scientific computing have carefully studied what should zero be within a computer. It seems like such an obvious problem, and yet it is one of the fundamental problems that we have in scientific computing: what is zero, how do we know a process has terminated, what is the accuracy of a solution? These are all interrelated and connected with one another.

The singular value decomposition  $H = U \Sigma V^t$  has played a very important role in computations, in solving least-squares problems, in signal processing problems, and so on. It is just a very simple decomposition, yet it is of fundamental importance in many problems arising in technology. One of the earliest algorithms was given by Hestenes. In a later paper I gave some technique for computing the singular-value decomposition, and then I popularized it. That was one of the important things: to make people aware of what computations can do. So not all what is important in scientific computing is to develop new methods, but to let people know that you can use these methods and how effective they can be. I went to a conference last year on computing the singular-value decomposition. Half the talks were on how to avoid to compute the singular-value decomposition. So frequently people try to eliminate this composition, although they try to essentially get the same effect.

### *Cyclic reduction*

We'll go on to another important problem that I have been quite interested in: it's called Poisson solvers. It is a very simple problem, solving Poisson's equation on a rectangular domain, and the question is how can one solve these very specialized problems. It is a second-order partial differential equation, which in one dimension reads

$$-v_{xx} = f(x), \quad 0 < x < 1, \quad v(0) = \alpha, \quad v(1) = \beta.$$

One can use a finite-difference approximation on a grid with stepsize  $h = 1/(N + 1)$  to obtain

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f(x_i) \equiv g_i, \quad (i = 1, 2, \dots, N),$$

$$\text{where } v_i = v(x_i), \quad v_0 = \alpha, \quad v_{N+1} = \beta.$$

It just involves three points: the  $i$ -th,  $(i - 1)$ -st and  $(i + 1)$ -st point. This is a system of linear algebraic equations. And now I just want to explain a very simple algorithm: it's so easy to see. Take three successive equations

$$\begin{array}{rcccccl} -v_{2i-2} & +2v_{2i-1} & -v_{2i} & & = & g_{2i-1} \\ & -v_{2i-1} & +2v_{2i} & -v_{2i+1} & = & g_{2i} \\ & & -v_{2i} & +2v_{2i+1} & -v_{2i+2} & = & g_{2i+1} \end{array}$$

If you multiply the middle equation by two and add them together, you have eliminated two of the unknowns. We end up with the equation

$$-v_{2i-2} + 2v_{2i} - v_{2i+2} = g'_{2i}, \quad \text{where } g'_{2i} = -g_{2i-1} + 2g_{2i} - g_{2i+1}.$$

Notice that now you have a new tri-diagonal system of equations which only involves the even unknowns. We can do this simultaneously to all the even equations. In other words, we can eliminate half the unknowns by this very simple trick of just adding together some of the right-hand sides. We don't have to do anything to the central equation. So the matrix of coefficients is the same as before, but we have reduced the number of unknowns by a factor of two. This is the idea of cyclic reduction. We can apply this process again introducing  $g''_{2i}$ , and the process is continued. So, suppose we began with a set of unknowns  $v_1$  through  $v_7$  (i.e.  $N = 7$ ). After one reduction we just have the even unknowns ( $v_2, v_4$  and  $v_6$ ), and after two reductions we have one unknown ( $v_4$ ). So now you can solve for  $v_4$ , and given  $v_4$  you can solve for  $v_2$  and  $v_6$ . Given  $v_2, v_4$  and  $v_6$  you can solve for  $v_1, v_3, v_5$  and  $v_7$ . This is the basic idea for cyclic reduction. It is a very simple idea and it is a very useful idea.

The algorithm lends itself to parallel computing in a very nice way, because to do the one reduction we independently create three new right-hand sides. So all can be done simultaneously. What this means, perhaps some of you will notice, if you had three processors, the first time you would use three processors and then the next time you would use only one processor. So although there is a lot of parallelism in the algorithm, you are using fewer and fewer processors all the time, which makes it not a very successful algorithm from that point of view.

Well, as recently pointed out, not only could you eliminate the odd unknowns and end up with the even unknowns, but you can eliminate the even unknowns and get the odd equations. This seems like you are doing more work, but it means that you are using all the processors simultaneously. In other words, instead of continually eliminating half the unknowns, you perform the same operations on the other half of the variables, such that simultaneously you are working with all the processors at the same time. So here is another consideration: even though it is from a mathematical/numerical point of view you are doing more operations, you are simultaneously using all your processors. That plays an important role, the more things you can get done simultaneously, the better and faster will be your computation. So the dimension added by having parallel computation is really quite different and requires some different thought than one would have when using one processor.

7									$v_7$	$-(v_{xx} + v_{yy}) = f(x, y)$
6									$v_6$	
5									$v_5$	
4									$v_4$	
3									$v_3$	
2									$v_2$	
1									$v_1$	

$$v_j = \begin{pmatrix} v_{1j} \\ v_{2j} \\ \vdots \\ v_{mj} \end{pmatrix}, \quad j = 1, 2, \dots, 7$$

If you proceed to two dimensions, you cannot continue the cyclic reduction as I mentioned it earlier, but you can do what is called block elimination. In other words when one is solving Poisson's equation for two sets of variables, you can eliminate lines 1, 3, 5 and 7 simultaneously. That would leave some equations for lines 2, 4 and 6. Then you can eliminate lines 2 and 6, and finally you have one equation for line 4. So cyclic reduction allows you to sort of eliminate half the lines, and one can perform faster and faster computations. There was so much interest in this whole topic: it became sort of a college industry. Many people were attempting their hand for getting fast Poisson solvers: what could be the fastest possible solver? There is a number of different ideas, and there is now a lot of software produced.

### *Building blocks*

In scientific computing we build blocks, and here we have cyclic reduction as one block for solving Poisson's equation, and the conjugate gradient method for solving systems of linear equations that are symmetric and positive definite. Sometimes we don't have Poisson's equation, but we have something near Poisson's equation. So we combine the two ideas, cyclic reduction for solving Poisson's equation, conjugate gradients for solving the problem completely. We can combine methods and find a new method for solving elliptic problems over rectangular domains.

What about non-rectangular domains? What happens when you have the union of two rectangular domains and you want to solve Poisson's equation. That is an old mathematical problem. If you can solve a problem exactly over one subdomain and you have the union of those two subdomains, how can you solve the problem over the entire domain.

There is a method called the Schwartz alternating procedure. There is also a whole area called domain decomposition. Domain decomposition is a way of taking a problem over some domain, breaking up that domain into subdomains, solving each one of the problems independently over each one of those subdomains, and then pasting the solution back together. This just seems like a simple idea, but each year now there is a new meeting held on domain decomposition. It has become a very popular topic. It is very useful in parallel computation: that's one reason it is so popular. The other reason it is popular, is that mathematicians can prove all kinds of theorems without ever touching a computer. It has become a very popular mathematical topic. That's one offcome of fast Poisson solvers, and there are other areas where fast Poisson solvers also play a role.

### EPILOGUE

It looks like the audience has had a long day and are ready for a break. Let me just tell you that scientific computing is a wonderful subject. It combines so many different aspects of mathematics and computer science and applied mathematics, as I said earlier. It is going to grow because technology is growing and we cannot forget that. So I hope some of you will get interested in this topic: it is a very rich topic, it has many different facets and I hope you'll find it of interest. Thank you.

### References

- [1] C.F. GAUSS *Theory of the Combination of Observations Least Subject to Errors* (translated by G.W. Stewart). SIAM Classics in Applied Mathematics 11, Philadelphia, 1995.