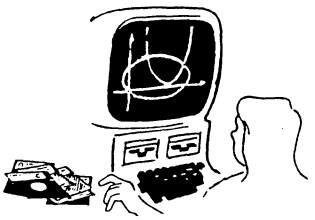


COMPUTER CORNER

Edited by

David A. Smith
Department of Mathematics, Duke University, Durham, North Carolina 27706

R. Stephen Cunningham
Department of Computer Science, California State University, Stanislaus, Turlock, California 95380

Harley Flanders
Department of Mathematics, University of Michigan, Ann Arbor, Michigan 48109-1092

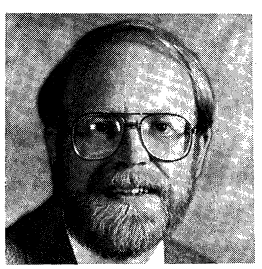
In this column, readers are encouraged to share their expertise and experiences with computers as they relate to college level mathematics. Articles that illustrate how computers can be used to enhance pedagogy, solve problems, and model real-life situations are especially welcome.

The **Algorithm of the Bi-Month** *will feature algorithms which solve mathematical problems through skillful computer use. It will also illustrate how the mathematical point of view may change in order to implement algorithms on a computer. Readers are invited to share interesting (not necessarily original) algorithms in any reasonably structured language or "pseudocode," with explanatory text to make their purpose and validity clear, and enough comments to enable easy translation to other languages. Such algorithms should be sent to Harley Flanders, who may, at his discretion, translate them into Standard Pascal for presentation.*

Software for review (please include descriptive literature) should be sent to R. S. Cunningham. All other material for this column should be sent to the journal editor. (See inside cover for details on submitting manuscripts.)

Why Should We Pivot in Gaussian Elimination?

Edward Rozema



Edward Rozema received his undergraduate training at Calvin College in Grand Rapids, Michigan. He received his Ph.D. in 1972 from Purdue University under the direction of Richard B. Holmes. He taught one year at Purdue University Calumet Campus before moving south to the University of Tennessee at Chattanooga, where he is a Professor of Mathematics.

The Gaussian elimination procedure for solving a system of n linear equations in n unknowns is familiar to most precalculus students: (1) Write the system as an augmented matrix, (2) reduce the system to upper triangular form by the elementary row operations, and (3) solve for the variables by back substitution. The method is simple and terminates in a finite number of steps, with either the exact answer or the information that there is no unique solution. This straightforward procedure seems ideal for computer implementation. As long as we pivot (that is, interchange rows) to avoid division by zero, what can possibly go wrong? Well, almost everything can go wrong, as illustrated by the following examples.

Example 1. Consider the matrix equation

$$\begin{bmatrix} \varepsilon & 1 & 1 \\ 1 & -1 & 1 \\ .5 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 2.5 \end{bmatrix},$$

where ε is a constant. When $\varepsilon = 0$, a standard "naive" Gaussian elimination algorithm, which pivots only to avoid division by zero, will yield the exact solution $(x_1, x_2, x_3) = (1, 1, 1)$. When ε is positive but much smaller than 0.5, the answer should remain near to this solution: this system is "well-conditioned" for such ε in the sense that small changes in the coefficients give rise to small changes in the answer. Indeed, we see this from the exact solution

$$(x_1, x_2, x_3) = \left(1 + \frac{4\varepsilon}{2 - 4\varepsilon}, 1 + \frac{\varepsilon}{2 - 4\varepsilon}, 1 - \frac{3\varepsilon}{2 - 4\varepsilon}\right). \tag{1}$$

However, an implementation of the algorithm on an Apple Macintosh in the binary version of Microsoft Basic (which stores real numbers with 24 bit mantissas—about seven significant decimal digits) produced the following results for small values of ε:

$\varepsilon = 2^{-23}$	$(x_1, x_2, x_3) = (1.000\,000, 0.999\,9999, 1.000\,000)$
$\varepsilon = 2^{-23.5}$	$(x_1, x_2, x_3) = (2.828\,427, 0.999\,9998, 1.000\,000)$
$\varepsilon = 2^{-24}$	$(x_1, x_2, x_3) = (0.000\,000, 2.000\,000, 0.000\,000)$
$\varepsilon = 2^{-25}$	(x_1, x_2, x_3) doesn't exist, as the coefficient matrix is computed to be singular!

Notice that for $\varepsilon \approx 0$, the coefficient matrix is not even close to being singular, since the system is well-conditioned. In fact, for $\varepsilon = 0$, the coefficient matrix and its inverse are given by

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & -1 & 1 \\ .5 & 1 & 1 \end{bmatrix} \quad \text{and} \quad A^{-1} = \begin{bmatrix} -2 & 0 & 2 \\ -0.5 & -0.5 & 1 \\ 1.5 & 0.5 & -1 \end{bmatrix}.$$

Very different and even more startling things happen to the *exact* solution when ε is close to 0.5, for then the coefficient matrix is nearly singular. (If $\varepsilon = 0.5$, then the first and third rows of our original matrix are identical.) In order to analyze this case, let $\varepsilon = 0.5 + \delta$, where δ is a small nonzero constant. Then the exact solution of our matrix equation is

$$(x_1, x_2, x_3) = \left(-\frac{1}{2\delta}, -\frac{1}{8\delta} + \frac{3}{4}, \frac{3}{8\delta} + \frac{7}{4}\right). \tag{2}$$

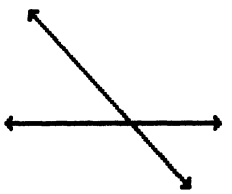
Observe, for example, the *exact* solution corresponding to each of the values δ:

$\delta = 2^{-24}$ $\quad (x_1, x_2, x_3) = (-8\,388\,608, -2\,097\,151.25, 6\,291\,457.75)$

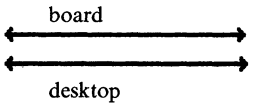
$\delta = -2^{-24}$ $\quad (x_1, x_2, x_3) = (8\,388\,608, 2\,097\,152.75, -6\,291\,454.25).$

These spectacular changes in the exact solution are typical of ill-conditioned (that is, non well-conditioned) systems. Surprisingly, the same program that performed so erratically for $\varepsilon \approx 0$ performed beautifully for $\varepsilon \approx .5$, as we will see in Table 2.

The tremendous changes in the exact solution can be explained geometrically in terms of the "tipping" of one of the two nearly parallel planes given by the first and third equations: Think of holding a long board on the palms of your hands just above a desktop. If you drop your right hand down slightly, the plane of the board will intersect the plane of the desktop to the right of the desktop. Then if you drop your other hand down, the line of intersection will move to the other side of the desktop. A small change in the balancing of the board results in a large change in the line of intersection between the planes of the board and the desktop. In the well-conditioned case considered earlier, these two planes intersect at an angle of about 20°. The figures illustrate the siutation in two dimensions.



A well-conditioned system: a small change in the coefficients produces a small change in the point of intersection.



An ill-conditioned system: a small change in the coefficients (tipping the board down on the left or right slightly) produces a large change in the point of intersection.

The well-conditioned case, $\varepsilon \approx 0$. What can account for the wide diversity of incorrect answers we see when $\varepsilon \approx 0$? An obvious answer is, "round-off errors caused by computer arithmetic." Although there are such errors, this does not adequately explain our solution results nor does it help us to alleviate the difficulties. A less obvious, but correct, explanation is that when we used the first equation to eliminate x_1 from the second and third equations, we divided by a small, nearly zero "pivot" (namely, ε). Unfortunately, many numerical analysis textbooks go no further than making this observation. There are at least three compelling reasons to pursue this observation further. Not doing so (i) leaves students with the impression that dividing by a small number, in and of itself, causes roundoff errors, (ii) it does not adequately account for the results above, and (iii) stopping at this point misses a valuable opportunity to underline the most common causes of roundoff errors and the interplay between them—namely, the subtraction of almost equal quantities (the most common source of devastating roundoff errors) and the addition of a small, important number to a large, relatively unimportant number.

Our intent here is to pursue this observation and demonstrate how small pivots may result in:

1. loss of significant figures due to subtraction of almost equal quantities during the back substitution process,
2. loss of significant figures due to the addition of large and small quantities and subsequent subtraction of almost equal quantities during the forward elimination procedure.

The first of these problems can be easily illustrated by the following.

Example 2. Solve

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Naive Gaussian elimination tells us to replace row 2 by (row 2) $-$ $1/\varepsilon$ (row 1), yielding

$$\left[\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 1 & -1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{cc|c} \varepsilon & 1 & 1 \\ 0 & -1-\dfrac{1}{\varepsilon} & -\dfrac{1}{\varepsilon} \end{array}\right].$$

Then back substitution yields

$$x_2 = \frac{-\dfrac{1}{\varepsilon}}{-1-\dfrac{1}{\varepsilon}} \quad \text{and} \quad x_1 = \frac{1-x_2}{\varepsilon}.$$

If $1/\varepsilon$ is so large that the computer replaces $-1-(1/\varepsilon)$ by $-1/\varepsilon$, then x_2 will be assigned the value 1. This is quite accurate (taking $\varepsilon = 10^{-8}$, for example, we see that $x_2 = -10^8/(-1-10^8) = .999\,999\,990\ldots$.) Note, however, that the assignment $x_2 = 1$ results in the computation $x_1 = 0$. This is very bad since, in fact, $x_1 = x_2$; there are no significant figures left in the computed solution. The loss of significance occurred during the back substitution process when almost equal quantities (namely, 1 and x_2) were subtracted.

Let's consider the general solution of

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & & \ddots & \vdots \\ & \mathbf{0} & & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix},$$

where the reduced upper triangular matrix has been obtained by elimination. Then the back substitution formula for x_i is

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}},$$

where the x_j $(i+1 \le j \le n)$ have already been computed by back substitution. If a_{ii} is small, then its reciprocal is large. If x_i is *not* large, then the numerator, $b_i - \sum_{j=i+1}^{n} a_{ij}x_j$, must be small. Finally, if b_i is not small, then it must follow that $\sum_{j=i+1}^{n} a_{ij}x_j$ and b_i must be nearly equal. Therefore, a small pivot a_{ii} may result in the loss of significant digits due to subtraction of almost equal quantities during the back substitution process. This is what occurred in Example 2, for $i = 1$.

Remark. There is more than one way to compute x_i. We may first compute $\sum_{j=i+1}^{n} a_{ij}x_j$ and then subtract it from b_i, as suggested above. We may also start with b_i and subtract each of the terms $a_{ij}x_j$, one at a time. The author obtained the same results by both methods, but this may vary somewhat on different computers.

Example 2 *continued.* If we interchange the first two rows of the original augmented matrix, the pivot will no longer be small, and subtraction of almost equal quantities will no longer occur:

$$\left[\begin{array}{cc|c} 1 & -1 & 0 \\ \varepsilon & 1 & 1 \end{array}\right] \to \left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0 & 1+\varepsilon & 1 \end{array}\right].$$

Then back substitution yields

$$x_2 = \frac{1}{1+\varepsilon} \approx 1 \quad \text{if } \varepsilon \textit{ is very small},$$

and

$$x_1 = x_2 \approx 1.$$

To see what may go wrong during the elimination phase of the algorithm, we need an example with three equations and three unknowns.

Example 1 *revisited.* If we apply naive Gaussian elimination to the augmented matrix in Example 1, we get

$$\left[\begin{array}{ccc|c} \varepsilon & 1 & 1 & 2 \\ 0 & -1-\dfrac{1}{\varepsilon} & 1-\dfrac{1}{\varepsilon} & 1-\dfrac{2}{\varepsilon} \\ 0 & 1-\dfrac{.5}{\varepsilon} & 1-\dfrac{.5}{\varepsilon} & 2.5-\dfrac{1}{\varepsilon} \end{array}\right]. \tag{3}$$

If $1/\varepsilon$ is large, then all the entries with a division by ε involve the subtraction of a large number from a (relatively) small number. If the computer replaces all these entries by the larger number, the resulting matrix is

$$\left[\begin{array}{ccc|c} \varepsilon & 1 & 1 & 2 \\ 0 & -\dfrac{1}{\varepsilon} & -\dfrac{1}{\varepsilon} & -\dfrac{2}{\varepsilon} \\ 0 & -\dfrac{.5}{\varepsilon} & -\dfrac{.5}{\varepsilon} & -\dfrac{1}{\varepsilon} \end{array}\right].$$

Applying elimination one more time yields the matrix:

$$\left[\begin{array}{ccc|c} \varepsilon & 1 & 1 & 2 \\ 0 & -\dfrac{1}{\varepsilon} & -\dfrac{1}{\varepsilon} & -\dfrac{2}{\varepsilon} \\ 0 & 0 & 0 & 0 \end{array}\right],$$

so the computer reports that there is no unique solution. This is evidently what was observed earlier with $\varepsilon = 2^{-25}$.

If the computer does not replace entries such as $-1-(1/\varepsilon)$ by $-1/\varepsilon$, then continuing the elimination from the augmented matrix labeled (3) would yield:

$$\left[\begin{array}{ccc|c} \varepsilon & 1 & 1 & 2 \\ 0 & -1-\dfrac{1}{\varepsilon} & 1-\dfrac{1}{\varepsilon} & 1-\dfrac{2}{\varepsilon} \\ 0 & 0 & \left(1-\dfrac{.5}{\varepsilon}\right)-\left[\dfrac{1-\frac{.5}{\varepsilon}}{-1-\frac{1}{\varepsilon}}\right]\left(1-\dfrac{1}{\varepsilon}\right) & \left(2.5-\dfrac{1}{\varepsilon}\right)-\left[\dfrac{1-\frac{.5}{\varepsilon}}{-1-\frac{1}{\varepsilon}}\right]\left(1-\dfrac{2}{\varepsilon}\right) \end{array}\right].$$

The computations in the third row may result in a loss of significant digits due to the subtraction of almost equal quantities. There are at least two possibilities:

(i) x_3 is computed with fewer significant digits than desired, and consequently the computations of x_2 and x_1 during the back substitution process also have fewer digits than desired. When we let $\varepsilon = 2^{-24}$, the last entry in column four was computed to be zero, and the resulting computations were

$$x_3 = 0, \quad x_2 = \frac{-2/\varepsilon}{-1/\varepsilon} = 2, \quad \text{and} \quad x_1 = 0.$$

(ii) x_3 is computed as nearly 1, and x_2 is also computed as nearly 1. However, x_1 is computed by the back substitution formula, $x_1 = (2 - x_2 - x_3)/\varepsilon$, to be far from 1 because of the subtraction of almost equal quantities. When we let $\varepsilon = 2^{-23.5}$, this is exactly what happened. (Curiously, x_1 was computed to be $2\sqrt{2}$; perhaps $2 - x_2 - x_3$ was computed to be 2^{-22} rather than $2^{-23.5}$.) Here we are seeing the same back substitution phenomena as illustrated in Example 2.

The strategy that should be employed is now fairly evident:

> *Before eliminating below a diagonal element, first interchange (if necessary) the row containing the diagonal element with a row below it that will make the absolute value of the diagonal element as large as possible.*

This will greatly reduce the number of times nearly equal quantities are subtracted. This commonly used strategy is called "partial (or column) pivoting." ("Complete pivoting" involves both row and column interchanges to secure the pivot with maximal absolute value among *all* remaining candidates. This is much less frequently used.)

Let's illustrate partial pivoting for Example 1. Since 1 is the largest entry in the first column, we interchange rows 1 and 2 in the original augmented matrix to obtain

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ \varepsilon & 1 & 1 & 2 \\ .5 & 1 & 1 & 2.5 \end{array}\right].$$

Applying elimination yields

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 0 & 1+\varepsilon & 1-\varepsilon & 2-\varepsilon \\ 0 & 1.5 & .5 & 2 \end{array}\right].$$

Since ε is small, continuing the elimination process will not result in subtractions of almost equal quantities. In particular, if ε is so small that the computer drops it, the augmented matrix becomes

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1.5 & .5 & 2 \end{array}\right].$$

Since 1.5 is the largest entry in column 2 on or below the diagonal, partial pivoting requires us to interchange rows 2 and 3. This gives

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 0 & 1.5 & .5 & 2 \\ 0 & 1 & 1 & 2 \end{array}\right].$$

Assume, for simplicity, that the computer uses seven decimal digits for the mantissas. Then elimination will yield

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 0 & 1.5 & .5 & 2 \\ 0 & 0 & .6666667 & .6666667 \end{array}\right].$$

Therefore, back substitution now gives $x_3 = 1$, $x_2 = 1$, and $x_1 = 1$, all of which are close to the true answers given in equation (1).

Computer Experiments

The author coded a fairly standard algorithm for Gaussian elimination on an Apple Macintosh in Microsoft's binary version of Basic. (The algorithm was adapted from the Fortran in [1, pp. 220–223].) The results are summarized in the following tables.

Example 1 one more time. All of the different types of behavior described above were observed for different values of ε. In Table 1, the values obtained with pivoting are correct to the seven digits shown (in each case); the error reported is just the difference between the results obtained with pivoting and those obtained without pivoting. The table is in order of decreasing ε, so the worst case is last.

Table 1. The well-conditioned case: $\varepsilon \approx 0$

		Without pivoting	With pivoting	Error w/o pivoting
$\varepsilon = 2^{-5}$	x_1	1.066 666	1.066 667	.000 001
	x_2	1.016 666	1.016 667	.000 001
	x_3	0.950 0005	0.950 0000	−.000 0005
$\varepsilon = 2^{-13}$	x_1	1.000 000	1.000 244	.000 244
	x_2	0.999 8779	1.000 061	.000 1831
	x_3	1.000 000	0.999 8168	−.000 1832
$\varepsilon = 2^{-23.5}$	x_1	2.828 427	1.000 000	−1.828 427
	x_2	0.999 9998	1.000 000	.000 0002
	x_3	1.000 000	0.999 9997	−0.000 0003
$\varepsilon = 2^{-24}$	x_1	0.000 000	1.000 000	1.000 000
	x_2	2.000 000	1.000 000	−1.000 000
	x_3	0.000 000	1.000 000	1.000 000
$\varepsilon = 2^{-25}$	x_1	no solution	1.000 000	—
	x_2	no solution	1.000 000	—
	x_3	no solution	1.000 000	—

The ill-conditioned case, $\varepsilon \approx .5$. Ill-conditioned systems are a lot tougher to handle than well-conditioned ones. However, since the coefficient matrix of an ill-conditioned system is nearly singular, it makes sense to believe that no matter what pivoting strategy is employed, all the coefficient entries in the last row of the final augmented matrix will be nearly zero. Hence a small pivot will be difficult to avoid, and roundoff errors may easily destroy any confidence we have in the computed solution.

To see what happens in the ill-conditioned case, we ran the program using $\delta = 2^{-24}$ and $\delta = -2^{-24}$ (recall that $\varepsilon = .5 + \delta$). These numbers were chosen so that ε could be entered without roundoff.

Examination of the numbers in Table 2 shows a remarkable result:

The usual pivoting strategy gave much poorer results!

Table 2. The ill-conditioned case, $\varepsilon \approx .5$

		Without pivoting	With pivoting	True solution
$\delta = 2^{-24}$	x_1	−8,388,608	−11,184,810	−8,388,608
	x_2	−2,097,151	−2,796,201	−2,097,151.25
	x_3	6,291,457	8,388,608	6,291,457.75
$\delta = -2^{-24}$	x_1	8,388,608	11,184,810	8,388,608
	x_2	2,097,153	2,796,204	2,097,152.75
	x_3	−6,291,455	−8,388,608	−6,291,454.25

The reason for this result is that interchanging rows 1 and 2 and doing the elimination yields

$$\left[\begin{array}{ccc|c} 1 & -1 & 1 & 1 \\ 0 & 1.5+\delta & .5-\delta & 1.5-\delta \\ 0 & 1.5 & .5 & 2 \end{array}\right].$$

Now the 2×2 system in the bottom two rows is very nearly singular; continuing the elmination procedure results in the loss of significant digits due to the subtraction of almost equal quantities.

On the other hand, solving the system without pivoting yields the matrix (3) in Example 1, after elimination in the first column:

$$\left[\begin{array}{ccc|c} \varepsilon & 1 & 1 & 2 \\ 0 & -1-\dfrac{1}{\varepsilon} & 1-\dfrac{1}{\varepsilon} & 1-\dfrac{2}{\varepsilon} \\ 0 & 1-\dfrac{.5}{\varepsilon} & 1-\dfrac{.5}{\varepsilon} & 2.5-\dfrac{1}{\varepsilon} \end{array}\right]. \tag{3}$$

As $\varepsilon = .5 + \delta \approx .5$, the second row is computed without loss of significance. Although the third row involves the subtraction of almost equal quantities, hardly any loss of significance occurs since

$$\frac{.5}{\varepsilon} = \frac{1}{1+2\delta} = 1 - 2\delta + 4\delta^2 - \cdots \approx 1 - 2\delta,$$

yielding

$$1 - \frac{.5}{\varepsilon} \approx 2\delta.$$

This is very close to the actual value of $\delta/(.5 + \delta)$. Similarly, the other entries can be written in terms of δ yielding

$$\left[\begin{array}{ccc|c} .5+\delta & 1 & 1 & 2 \\ 0 & -3+4\delta & -1+4\delta & -3+8\delta \\ 0 & 2\delta & 2\delta & .5+4\delta \end{array}\right].$$

The 2×2 system in the bottom two rows is no longer ill-conditioned since the two lines in the x_2x_3-plane are not nearly parallel. In order to continue the elimination

and to model the behavior of a computer, third-order terms will be omitted and, in each computation below only the two lowest-order terms will be retained; that is, $a + b\delta + c\delta^2$ will be replaced by $a + b\delta$ and $(a/\delta) + b + c\delta$ will be replaced by $(a/\delta) + b$. Let's continue the elimination, replacing row 3 by

$$\begin{aligned}(\text{row } 3) - \frac{2\delta}{-3+4\delta}(\text{row } 2) &= (\text{row } 3) - \frac{2\delta}{-3}\left(\frac{1}{1-\frac{4\delta}{3}}\right)(\text{row } 2) \\ &\approx (\text{row } 3) + \frac{2\delta}{3}\left(1+\frac{4\delta}{3}\right)(\text{row } 2) \\ &= (\text{row } 3) + \left(\frac{2\delta}{3} + \frac{8\delta^2}{9}\right)(\text{row } 2)\end{aligned}$$

to obtain

$$\left[\begin{array}{ccc|c} .5+\delta & 1 & 1 & 2 \\ 0 & -3+4\delta & -1+4\delta & -3+8\delta \\ 0 & 0 & \frac{4\delta}{3}+\frac{16\delta^2}{9} & .5+2\delta \end{array}\right].$$

Back substitution gives

$$\begin{aligned} x_3 &= \frac{1}{\frac{4\delta}{3}+\frac{16\delta^2}{9}}(.5+2\delta) \approx \frac{3}{8\delta}+1 \\ x_2 &= \frac{-3+8\delta+(1-4\delta)x_3}{-3+4\delta} \approx -\frac{1}{8\delta}+1 \\ x_1 &= \frac{2-x_2-x_3}{.5+\delta} \approx -\frac{1}{2\delta}+1, \end{aligned}$$

which are in remarkable agreement with both the exact answer given in equation (2) and the computed answer given in Table 2.

Should we conclude from this example that we shouldn't pivot when solving ill-conditioned problems? No! For if we had *started* with the first two rows interchanged and then solved the sytem with or without pivoting, we would get exactly the same poor results. We can only conclude that ill-conditioned systems are hard to solve accurately with finite precision and a fixed strategy.

Postscript. Proper use of partial pivoting requires that the coefficients in different rows be somewhat comparable in size. For example, if we multiply the first equation in Example 2 by $1/\varepsilon$, then we would not be required to pivot, since each entry in the first column would be one. However, if we were to carry out the elimination and back substitution on the resulting system, we'd observe again that $x_2 = 1$ and $x_1 = 0$. One should start Gaussian elimination with a division of each row by the coefficient in that row which is largest in absolute value. This procedure is called *scaling*, and it makes the absolute value of the largest entry in each row equal to 1. Our examples were chosen so that scaling was unnecessary.

The classic treatment of rounding errors is by J. H. Wilkinson [3]. A thorough treatment of matrix computation can be found in [2].

Acknowledgement. This paper was written while on partial release time provided by the UTC Department of Mathematics and with equipment partially supplied by the UTC Center of Excellence for Computer Applications.

REFERENCES

1. Ward Cheney and David Kincaid, *Numerical Mathematics and Computing*, Second Edition, Brooks/Cole Publishing Company, Belmont, CA, 1985.
2. Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.
3. James H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

ALGORITHM OF THE BI-MONTH

Drawing a Circle

Harley Flanders, University of Michigan, Ann Arbor

The problem we consider here is to plot on the computer screen a circle with center (A, B) and radius R. We shall discuss several procedures for doing so and compare their speeds. To simplify matters, let us first assume that the screen is square, 201 by 201 pixels, described as all integer pairs (X, Y) such that $-100 \le X \le 100$ and $-100 \le Y \le 100$. The circle's size will be limited to radius 100. We shall also assume that there exist two primitive procedures

```
PIXEL(X, Y: Integer);
LINE(X0, Y0, X1, Y1: Integer);
```

the first for plotting a point at (X, Y), and the second for drawing the segment from $(X0, Y0)$ to $(X1, Y1)$.

We begin by exploring algorithms based on regular polygon approximations. If there are enough sides, a polygon will appear to be as good a (continuous) circle as can be drawn on the (discrete) screen. The routine CIRCLE1 uses the standard trigonometric parametrization of the circle to compute successive vertices in the most direct way possible.

Our choice of 80 sides seems reasonable; Choosing 160 sides results in no visible difference, whereas choosing 40 sides results in a visible polygon, not a circle.

```
procedure  CIRCLE1(A, B: Integer; R: Real);

   const  SIDE_NO = 80;

   var                  T, DT: Real;
       J, X, Y, OLD_X, OLD_Y: Integer;

   begin
   T := 0.0;  DT := 2.0*PI/SIDE_NO;
   OLD_X := Round(R);  OLD_Y := 0;
   for  J := 1  to  SIDE_NO  do
      begin
      T := T + DT;
      X := Round( R * Cos(T) );
```