INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
RICE UNIVERSITY

Computational Science: Identifying and Explaining the Mathematical Computational Methods used by the TI-83 Calculator

by

Emily Caroline Barra

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Master of Arts

APPROVED, THESIS COMMITTEE:

Richard A. Tapia, Noah Harding Professor
Computational and Applied Mathematics
Rice University

William W. Symes, Noah Harding Professor
Computational and Applied Mathematics
Rice University

Mark Embree, Assistant Professor
Computational and Applied Mathematics
Rice University

Cynthia Lanius, Executive Director
Center for Excellence and Equity in Education
Rice University

HOUSTON, TEXAS

MAY, 2002
ABSTRACT

Computational Science: Identifying and Explaining the Mathematical
Computational Methods used by the TI-83 Calculator

by

Emily Caroline Barra

This paper looks in detail at the algorithms used by the Texas Instruments TI-83 calculator to calculate the square root of \( x \), the solution to \( Ax = b \) and the trigonometric functions sine and cosine. What is exciting about these algorithms is that for various reasons, TI uses techniques that are not necessarily well-known or frequently used by today's numerical analysis community. I have written the explanations of these algorithms in such a way that high school math teachers as well as bright high school students searching for enrichment may use them.
ACKNOWLEDGMENTS

This thesis has been an incredible journey. I have learned so much about myself through this process, and I can honestly say that the writing served as a source of enrichment and joy. However, I could not have completed it without the support and help of a great number of people.

First and foremost, thank you to Richard Tapia and Cynthia Lanius for creating this topic. Were it not for your textbook, An Introduction to Computational Science: A High School Curriculum, this topic would not be pursued. Thank you, also, for spending so much time working with my drafts, making suggestions, helping guide my writing and being so supportive overall of my work.

Thank you also to Edith Aguirre and Naomi Reed for kick-starting the project by diligently communicating with Texas Instruments collecting the majority of the research materials. Your initial writing and presentation served as the foundation of this paper.

On behalf of all of us here at Rice, I would like to thank John Rodriguez at Texas Instruments for serving as our gateway into the company. Thank you also to Randy Ahlfinger and Noe Benavidez at Texas Instruments for your informative e-mail responses and reference suggestions. This paper would not have been possible without your help.

I would like to thank my committee members for taking time to support me this semester. A special thanks to Mark Embree for your comments on my writing, your extensive help with my understanding of the Ax=b section, and your entrusting me with a substantial chunk of your personal library!

Thank you to Danny Sorensen for his counsel on Crout factorization as well as loaning me his book, Numerical Linear Algebra for High-Performance Computers.
I am, of course, also grateful to my classmates and fellow students at Rice, my roommate, my friends and family and my co-workers at the University of Houston for all your support over the last two years. I would not have reached this point without you.

Thank you to the faculty and staff at Mary Baldwin College for your support during my four years of undergraduate work and to St. Agnes Academy for giving me the solid mathematics background I needed to be where I am today.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>The Square Root of $x$</td>
<td>4</td>
</tr>
<tr>
<td>Solving $Ax=b$</td>
<td>15</td>
</tr>
<tr>
<td>Sine and Cosine of $x$</td>
<td>24</td>
</tr>
<tr>
<td>Conclusion</td>
<td>32</td>
</tr>
<tr>
<td>References</td>
<td>33</td>
</tr>
</tbody>
</table>
Introduction

K-12 teaching and curriculum have become increasingly dependent on technology in the classroom. As the presence of technology grows in elementary and secondary learning environments, so grows the importance of fundamental concepts in mathematics. For example, let us say that a student is trying to graph the equation \( y = (x + 3)^2 + 5 \) using a graphing calculator. If the student does not already have a general idea of domain, range and what the parent function \( y = x^2 \) looks like, the resulting calculated graph could be invisible to the viewer simply because the calculator graph settings are fixed incorrectly for that particular problem.

Appropriate use of calculators like the Texas Instruments TI-83 greatly complements a good existing curriculum and a strong teacher. Just as a quality curriculum is enhanced with a talented teacher, the use of calculators with proper awareness of calculation methods can be beneficial to a student's understanding of the mathematical concepts. Most textbooks today require students to use technology to answer fundamental mathematical questions. It is essential that teachers lead students in an appreciation of how technology assists us in finding solutions rather than avoiding the theoretical concepts. Incorporating technology into the classroom is a balance.
In order to transform classrooms from button-pushing exercises to true consciousness of computational methods, we would like to explain and simplify information collected from Texas Instruments regarding the TI-83 calculator. Along with exposing the mathematical theory behind some of the algorithms used, we will communicate these ideas in a way in which they can be readily incorporated into standard high school math curricula.

Do students ever wonder how their calculators calculate? How many students have entered a large number into their calculator and then repeatedly pushed the $\sqrt{x}$ button until the number one appeared? How does this happen? If you try to reverse that operation, the result is certainly not your original number! It is this type of curiosity we are hoping to address, and in some cases stimulate, in the next few pages.

A secondary goal of this paper is to relate these algorithms to other topics discussed in the textbook, An Introduction to Computational Science: A High School Curriculum, written by Richard Tapia and Cynthia Lanius of the Center for Excellence and Equity in Education at Rice University. We hope that this paper, as a supplement in that text, will satiate the curiosity of any inquiring high school student and assist all mathematics teachers in further enriching their classrooms.

We began our pursuit by asking Texas Instruments about three specific functions: square root of $x$, the solution of the system of linear equations $Ax=b$, and the trigonometric functions $\sin(x)$ and $\cos(x)$ as they relate to the TI-83 calculator. Texas Instruments understandably considers specifics of the code used to calculate these functions proprietary information. However, they did provide us with information which
we used to present fairly rigorous and detailed algorithms showing how the calculator evaluates these functions.

Let us take note of some important fundamentals regarding the TI-83 calculator. Helmut Knaust [18] states that all calculators perform the following four operations inexpensively:

1. Addition and Subtraction
2. Storing data to memory and retrieving data from memory (using registers)
3. Digital shifts (multiplication and division by the base)
4. Comparisons

However, division and intermediate storage using Random Access Memory (RAM) are expensive operations. In order to optimize the efficiency of TI-83 calculator operations, Texas Instruments implemented algorithms to specifically avoid using division and intermediate storage when possible.

As we present the ideas of this paper, also remember that the TI-83 calculator has different needs from a computer. In each section, we note what method might be more suitable for a computer, and give an explanation why the lesser known and perhaps less elegant method has been fine-tuned to optimize solution efficiency in the TI-83.
1. The Square Root of $x$

One of the most common methods used to calculate the square root of $x$ in a computer is Newton's method. However, Newton's method requires division (see figure 1), and as noted in the introduction, for the TI-83, division is expensive.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$  \hspace{1cm} (Figure 1)

In calculating the square root of $x$, the TI-83 uses an algorithm called the Extraction of Roots by Repeated Subtractions. Until some time in the 1960's, school children were taught this algorithm for approximating the square root of an arbitrary number. One can use this algorithm to approximate the root in a manner similar to long division. This method of repeated subtractions is unexpectedly faster and more efficient than Newton's method in the TI-83 simply because of the division.

The extraction of the square root by repeated subtractions is an interesting algorithm. While we have seen the algorithm presented in several locations in the literature, we have not been able to locate a complete derivation of the algorithm. Moreover, since the understanding of why the algorithm works depends critically on its derivation, we are going to derive the repeated subtraction algorithm for calculating the square root of a given number in its entirety. Our derivation will be facilitated by working with a specific example. Towards this end, we choose $x = 9763.25$. Our objective is to approximate $\sqrt{9763.25}$ without performing division.
We begin by writing $x$ so that it looks like a perfect square. This is done by grouping the digits on the left and right of the decimal point in groups of two and recalling the definition of decimal expansion. For our example, we have:

$$9763.25 = 97 \cdot 10^2 + 63 \cdot 10^0 + 25 \cdot 10^{-2}.$$  \hspace{1cm} (1.1)

We next observe from (1.1) that we must have the decimal representation of the square root as follows:

$$\sqrt{9763.25} = b_1 \cdot 10^1 + b_2 \cdot 10^0 + b_3 \cdot 10^{-1} + b_4 \cdot 10^{-2} + \ldots$$  \hspace{1cm} (1.2)

Note that a "decimal digit" satisfies $0 \leq b_i \leq 9$. Also, when we say "decimal digits," we are referring to all digits – left and right of the decimal place. As an example, think about the number 625. You can easily see that

$$625 = 6 \cdot 10^2 + 25 \cdot 10^0$$ and further $\sqrt{625} = 25 = 2 \cdot 10^1 + 5 \cdot 10^0$.

Observe that in decimal notation (1.2) is written

$$\sqrt{9763.25} = b_1 b_2 b_3 b_4 \ldots$$  \hspace{1cm} (1.3)

where the decimal expansion could be infinite, i.e., the number of decimal places required is infinite. However, if we desire the square root correct to only $q$ decimal digits, we use the approximation:

$$\sqrt{9763.25} \approx b_1 \cdot 10^1 + b_2 \cdot 10^0 + b_3 \cdot 10^{-1} + \ldots + b_q \cdot 10^{-q}.$$  \hspace{1cm} (1.4)

Also observe that the leading decimal digit, $b_1$ in this case, must represent the $10^1$ place. If $b_1$ represented the $10^2$ place, the square would be greater than $b_1^2 \cdot 10^4$ (see (1.1) and (1.2)), and 9763.25 is not greater than $b_1^2 \cdot 10^4$ for any non-zero digit value of $b_1$ (even if $b_1 = 1, \ b_1^2 \cdot 10^4 = 10,000$). Similarly, if $b_1$ represented the $10^0$ place, the square would be
\[ b_1^2 \cdot 10^0 \text{ or } (b_1 b_2 b_3 b_4 \cdots)^2 \] which in turn would be strictly less than \((b_1 + 1)^2 \cdot 10^0\), which in turn would be less than or equal to \(10^2\), which is less than 9763.25.

So by forming groups of two digits, we can immediately determine what will be the position of the leading decimal digit in the square root. If we have \(n\) groups of two digits in \(x\) to the left of the decimal point, then the square root of \(x\) will have \(n\) digits to the left of the decimal point.

Our next step requires us to observe that if:

\[ x = (a_1 + a_2 + a_3 + \ldots)^2, \tag{1.5} \]

then by multiplying and grouping appropriately, we can write:

\[ x = [a_1^2] + [2a_1 a_2 + a_2^2] + [2(a_1 + a_2)a_3 + a_3^2] + [2(a_1 + a_2 + a_3)a_4 + a_4^2] + \ldots \] \(\tag{1.6}\)

Notice that the first bracketed term in (1.6) represents \(a_1^2\), the first two bracketed terms represent \((a_1 + a_2)^2\), the first three bracketed terms represent \((a_1 + a_2 + a_3)^2\), and so on.

We now square both sides of (1.2) using the grouping given by (1.6). This leads to:

\[
9763.25 = b_1^2 \cdot 10^2 \\
+ 2(b_1 \cdot 10^1)(b_2 \cdot 10^0) + b_2^2 \cdot 10^0 \\
+ 2(b_1 \cdot 10^1)(b_2 \cdot 10^0) + (b_2 \cdot 10^0)(b_3 \cdot 10^{-1}) + b_3^2 \cdot 10^{-2} \\
+ 2(b_1 \cdot 10^1)(b_2 \cdot 10^0) + (b_2 \cdot 10^0)(b_3 \cdot 10^{-1}) + (b_3 \cdot 10^{-1})(b_4 \cdot 10^{-2}) + b_4^2 \cdot 10^{-4} \\
+ 2(b_1 \cdot 10^1)(b_2 \cdot 10^0) + (b_2 \cdot 10^0)(b_3 \cdot 10^{-1}) + (b_3 \cdot 10^{-1})(b_4 \cdot 10^{-2}) + (b_4 \cdot 10^{-2})(b_5 \cdot 10^{-3}) + b_5^2 \cdot 10^{-6} \\
+ \ldots
\]

(1.7)

Recalling our groups of two, we want to write the right-hand side of (1.7) so that it resembles the grouping in (1.1), i.e., we want to write the right-hand side of (1.7) as a sum of products – a product multiplying \(10^2\), plus a product multiplying \(10^0\), plus a
product multiplying $10^{-2}$, plus a product multiplying $10^{-4}$ and so on. This motivates us
to rewrite (1.7) in the form:

$$
97 \cdot 10^2 + 63 \cdot 10^0 + 25 \cdot 10^{-2}
= \left\{ b_1^2 \right\} \cdot 10^2 \\
+ \left\{ 2(b_1 \cdot 10^1) b_2 + b_2^2 \right\} \cdot 10^0 \\
+ \left\{ 2(b_1 \cdot 10^2) + (b_2 \cdot 10^1) b_3 + b_3^2 \right\} \cdot 10^{-2} \\
+ \left\{ 2(b_1 \cdot 10^3) + (b_2 \cdot 10^2) + (b_3 \cdot 10^1) b_4 + b_4^2 \right\} \cdot 10^{-4} \\
+ \left\{ 2(b_1 \cdot 10^4) + (b_2 \cdot 10^3) + (b_3 \cdot 10^2) + (b_4 \cdot 10^1) b_5 + b_5^2 \right\} \cdot 10^{-6} \\
+ \ldots \\
+ \left\{ 2(b_1 \cdot 10^{p-1}) + (b_2 \cdot 10^{p-2}) + \ldots + (b_{p-1} \cdot 10^1) b_p + b_p^2 \right\} \cdot 10^{-2(p-2)} \\
+ \ldots 
$$

(1.8)

The idea for the repeated subtraction comes from staring at (1.8), with the hope that
something clever will jump out at us. Our initial reaction is that we would like to equate
like terms for the various powers of 10 and then solve for $b_1$ from the first equation,
$b_2$ from the second (since at this point, $b_1$ would be known), and then continue in this
fashion. This approach has the flavor of a general mathematical method called The
Method of Undetermined Coefficients. However, this will not work because such an
approach would not give the $b_i$'s as decimal digits. So we step back and look again and
come up with the following idea (again using (1.8)):

Choose $b_1$ as the largest digit satisfying

$$
b_1^2 \cdot 10^2 \leq 9763.25.
$$

Next, (using the $b_1$ you just found) determine $b_2$ as the largest digit satisfying

$$
2(b_1 \cdot 10)b_2 + b_2^2 \cdot 10^0 \leq 9763.25 - (b_1^2 \cdot 10^2).
$$
Now determine $b_3$ (using $b_1$ and $b_2$) as the largest digit satisfying
\[2(b_1 \cdot 10^2 + b_2 \cdot 10)b_3 + b_3^2 \cdot 10^{-2} \leq 9763.25 - (b_1^2 \cdot 10^2) - (2(b_1 \cdot 10)b_2 + b_2^2) \cdot 10^0.\]

We can continue in this fashion. Some reflection should convince the reader that this procedure will determine the correct $b_i$'s. Let us argue the case for $b_1$. The argument for the other $b_i$'s is similar, although not as simple. Suppose that (1.3) holds. From the properties of decimal numbers we have:
\[b_1 \cdot 0.000 \leq b_1b_2b_3 < (b_1 + 1) \cdot 10. \quad (1.9)\]

Now squaring all terms in (1.9) and recalling (1.3) leads to
\[b_1^2 \cdot 10^2 \leq 9763.25 < (b_1 + 1)^2 \cdot 10^2. \quad (1.10)\]

These inequalities demonstrate that $b_1$ is the largest digit such that $b_1^2 \cdot 10^2 \leq 9763.25$.

We now propose a modification to the proposed method which will make the algorithm simpler and cleaner and will allow the algorithm to be presented in a concise format which has the flavor of long division. This simplification leads us to the standard repeated subtraction algorithm.

Choose $b_1$ as the largest digit satisfying
\[b_1^2 \leq 97.\]

Choose $b_2$ (using the $b_1$ you just found) as the largest digit satisfying
\[2(b_1 \cdot 10)b_2 + b_2^2 \leq 9763 - (b_1^2 \cdot 10^2).\]

Choose $b_3$ (using $b_1$ and $b_2$) as the largest digit satisfying
\[2(b_1 \cdot 10^2 + b_2 \cdot 10)b_3 + b_3^2 \leq 976,325 - (b_1^2 \cdot 10^4) - (2(b_1 \cdot 10)b_2 + b_2^2) \cdot 10^2.\]
We continue in the obvious fashion guided by (1.8). Notice that in the modification we are always working with whole numbers and we bring in the digital pairs representing 9763.25 one at a time, i.e., we add one additional pair as we move to the determination of the subsequent $b_i$. This is what will allow us to derive a compact form of the algorithm with the flavor of long division.

We argued that the first form of the algorithm gives the correct $b_i$'s. We now show that the first form of the algorithm and the simplified form are equivalent, i.e., both lead to the same $b_i$'s. Hence, the simplified form gives the correct $b_i$'s.

Since $b_1$ is a digit, (1.10) is equivalent to

$$b_1^2 10^2 \leq 9700 < (b_1 + 1)^2 \cdot 10^2$$

which in turn is equivalent to

$$b_1^2 \leq 97 < (b_1 + 1)^2.$$ 

It follows that choosing $b_1$ as the largest digit less than or equal to 97 is the same as picking $b_1$ as the largest digit satisfying $b_1^2 10^2 \leq 9763.25$. The argument for other $b_i$'s is similar. So our two forms of the algorithm are equivalent.

Let us now demonstrate several steps of the algorithm.

**Step for $b_1$:**

Choose $b_1$ as the largest digit satisfying $b_1^2 \leq 97$. This gives $b_1 = 9$ since $9^2 = 81 \leq 97$ and we save the quantity $9^2 = 81$ for the next step.
Step for $b_2$:

Choose $b_2$ as the largest digit satisfying $2(b_1 \cdot 10)b_2 + b_2^2 \leq 9763 - (b_1^2 \cdot 10^2)$ or $2(90)b_2 + b_2^2 \leq 9763 - (81 \cdot 10^2)$. This simplifies to $180b_2 + b_2^2 \leq 1663$ and leads to $b_2 = 8$. We save the quantity $(180)(8) + (8)^2 = 1504$ for the next step.

Step for $b_3$:

Choose $b_3$ as the largest digit satisfying

$2(b_1 \cdot 10^2 + b_2 \cdot 10)b_3 + b_3^2 \leq 9763.25 \cdot 10^2 - (b_1^2 \cdot 10^4) - \left(2(b_1 \cdot 10)b_2 + b_2^2\right) \cdot 10^2$ or $2(9 \cdot 10^2 + 8 \cdot 10)b_3 + b_3^2 \leq 9763.25 \cdot 10^2 - (81 \cdot 10^4) - 1504 \cdot 10^2$. This simplifies to $1960b_3 + b_3^2 \leq 15,925$ and leads to $b_3 = 8$. We save the quantity $(1960)(8) + (8)^2 = 15,744$ for the next step.

Step for $b_4$:

Choose $b_4$ as the largest digit satisfying

$2(9 \cdot 10^3 + 8 \cdot 10^2 + 8 \cdot 10)b_4 + b_4^2 \leq 9763.25 \cdot 10^4 - (81 \cdot 10^6) - (1504 \cdot 10^4) - (15,744 \cdot 10^2)$. This simplifies to $19,760b_4 + b_4^2 \leq 18,100$ and leads to $b_4 = 0$. We save $(19,760)(0) + (0)^2 = 0$ for the next step.

Step for $b_5$:

Choose $b_5$ as the largest digit satisfying

$2(9 \cdot 10^4 + 8 \cdot 10^3 + 8 \cdot 10^2 + 0 \cdot 10)b_5 + b_5^2 \leq 9763.25 \cdot 10^6 - (81 \cdot 10^8) - (1504 \cdot 10^6) - (15,744 \cdot 10^4) - (0 \cdot 10^2)$. This simplifies to $197,600b_5 + b_5^2 \leq 1,810,000$ and leads to $b_5 = 9$. 
So we have calculated that to five significant digits, \( \sqrt{9763.25} \approx 98.809 \). If we square 98.809 (to check our accuracy) we obtain 9763.22.

We have purposely presented the repeated subtraction algorithm so that it can be written down in a nice compact form which resembles long division. We now present this compact form via the example we have been using.

**Repeated Subtraction Algorithm:**

Solve for \( x \):

\[
x = \sqrt{9763.25}
\]

1) Starting from the decimal point, pair off the digits.

\[
\sqrt{97\_63\_25}
\]

2) Find the largest number \( b_1 \) such that \( b_1^2 \leq 97 \). Place the value of \( b_1 \) above the left-most pair and subtract the square as if you were doing long division.

\[
\begin{array}{c}
9 \\
\sqrt{97\_63\_25} \\
\hline \\
-81 \\
\hline \\
16
\end{array}
\]

\( b_1 = 9 \)

3) Bring down the second pair of digits and concatenate with the remainder.

\[
\begin{array}{c}
9 \\
\sqrt{97\_63\_25} \\
\hline \\
-81 \\
\hline \\
16\ \ 63
\end{array}
\]

4) The next digit in the solution must be the largest digit to satisfy the equation

\[
2(90)b_2 + b_2^2 \leq 1663.
\]

Let us guess that \( b_2 = 9 \).

\[
2(90)(9) + (9)^2 = 1701, \text{ but } 1701 > 1663, \text{ so } b_2 = 9 \text{ is too large.}
\]
Let us now guess that \( b_2 = 8 \).

\[ 2(90)(8) + (8)^2 = 1504, \] and \( 1504 < 1663 \), so \( b_2 = 8 \).

Hence,

\[
\begin{array}{c c c c}
9 & 8 \\
\sqrt{97} & 63 & .25 \\
-81 & & \\
16 & 63 & \\
-15 & 04 & 2(90)(8) + (8)^2 = 1504
\end{array}
\]

5) Again, we bring down the next pair of digits and concatenate with the remainder.

\[
\begin{array}{c c c c}
9 & 8 \\
\sqrt{97} & 63 & .25 \\
-81 & & \\
16 & 63 & \\
-15 & 04 & \\
1 & 59 & 25
\end{array}
\]

6) The next digit in the solution satisfies the equation

\[ 2(9 \cdot 10^2 + 8 \cdot 10)b_3 + b_3^2 \leq 15,925. \]

Now we need to solve for \( b_3 \):

Let us guess that \( b_3 = 9 \).

\[ 2(980)(9) + (9)^2 = 17,721, \] but \( 17,721 > 15,925 \).

Let us guess that \( b_3 = 8 \).

\[ 2(980)(8) + (8)^2 = 15,744, \] and \( 15,744 < 15,925 \), so \( b_3 = 8 \).
Hence,

\[
\begin{array}{r}
9 & 8 & . & 8 \\
\sqrt{97.63} & _{25} & 25 \\
-81 \\
16 & 63 \\
-15 & 04 \\
1 & 59 & 25 \\
-1 & 57 & 44 \\
\end{array}
\]

\[
2(980)(8) + (8)^2 = 15,744
\]

7) Now, we add a pair of zeros at the right of the last decimal digit, and bring them down to concatenate with the remainder. Observe (and try yourself!) the next several iterations below (use (1.8) as a reference):

\[
\begin{array}{r}
9 & 8 & . & 8 & 0 & 9 & 1 & 5 \\
\sqrt{97.63} & _{25} & 00 & 00 & 00 & 00 \\
-81 \\
16 & 63 \\
-15 & 04 \\
1 & 59 & 25 \\
-1 & 57 & 44 \\
1 & 81 & 00 \\
- & & 0 \\
1 & 81 & 00 & 00 \\
-1 & 77 & 84 & 81 \\
3 & 15 & 19 & 00 \\
-1 & 97 & 61 & 81 \\
1 & 17 & 57 & 19 & 00 \\
-98 & 80 & 91 & 25 \\
18 & 76 & 21 & 75 & \ldots
\end{array}
\]

\[
(9)^2 = 81 \\
2(90)(8) + (8)^2 = 1504 \\
2(980)(8) + (8)^2 = 15,744 \\
2(9,880)(0) + (0)^2 = 0 \\
2(98,800)(9) + (9)^2 = 1,778,481 \\
2(988,090)(1) + (1)^2 = 1,976,181 \\
2(9,880,910)(5) + (5)^2 = 98,809,125
\]

So we have calculated 98.80915 as the square root of 9763.25 accurate to seven decimal digits.
Remarkably enough, this method can be used for any root. In calculating the cube root of $x$, $x$ may be grouped as:

$$x = (a_1 + a_2 + a_3 + ...)^3$$

and

$$x = [a_1^3] + [3a_1^2a_2 + 3a_1a_2^2 + a_2^3] + [3(a_1 + a_2)^2a_3 + 3(a_1 + a_2)a_2^2 + a_3^3]$$
$$+ [3(a_1 + a_2 + a_3)^2a_4 + 3(a_1 + a_2 + a_3)a_3^2 + a_4^3] + ...$$

And in general, for the $n^{th}$ root of $x$:

$$x = (a_1 + a_2 + a_3 + ... + a_p)^n$$

and

$$x = \sum_{k=1}^{k=p} \sum_{i=1}^{i=n} \left[ \begin{array}{c} C_i \left( \sum_{m=1}^{m=k-1} \right)^{n-i} \end{array} \right] \left[ a_i \right].$$

A most fascinating phenomenon regarding this method is that TI resurrected this "dead" algorithm and it turns out that it is the most effective algorithm to use on the TI-83! One has to wonder what other arcane algorithms are lurking out there just waiting to be re-discovered...
2. Solving $Ax=b$

There are many different methods to solve systems of equations. High school Algebra students are first taught to solve systems of equations by the method of substitution. Although this is a valuable technique, it becomes very tedious very quickly when done by hand – even when working with as few as three variables. Students are then introduced to Gaussian Elimination via linear combination. Advanced high school and some lower level college mathematics courses take the next step to teach the more advanced solution methods such as basic Gaussian Elimination, Gauss-Jordan Elimination, using the inverse of $A$ and Cramer’s Rule. These methods are generally better organized than substitution, but even they have their individual drawbacks as the size of the problem increases, say when calculating 10 unknowns by hand.

So we enter the use of computers. Computers accept instructions in the form of coded algorithms to quickly achieve results that would take much longer manually. According to the Oxford Dictionary of Mathematics [2], an algorithm is “a precisely described routine procedure that can be applied and systematically followed through to a conclusion.” Why not implement an algorithm for one of these methods to solve our problem? Gaussian Elimination (GE) is the most efficient of the four methods mentioned above. It is the method of choice when using a computer to solve for a system of equations since it is the fastest and most stable code. Does the TI-83 calculator use GE to solve a system of equations?
Texas Instruments indeed utilizes a form of the GE algorithm. "Gaussian Elimination," however, can be implemented in many ways. All implementations are based on the same technique, but each one has a variation which differentiates it from the others. TI identified an interesting variation of GE called Crout Factorization.

Crout Factorization results from Prescott D. Crout's observation that the order of the arithmetic in GE could be changed. The following is from Crout's original paper [5]:

The work of solving a system of equations (or evaluating the determinant) is largely concentrated in the determination of an "auxiliary matrix," and is roughly half that required by a matrix multiplication. The process is particularly adapted for use with a computing machine, for each element is determined by one continuous machine operation (sum of products with or without a final division). The setting down of this matrix and of the final solution is the only writing required by the process.

What makes Crout Factorization a better method for this calculator? As you know, GE begins with a matrix A and reduces the elements of A to a simplified form through various arithmetic operations. Each term $a_{ij}$ in the matrix $A$ is modified several times during standard GE. Crout requires considerably less work than other forms of GE in the TI-83 because it contains an algorithm that does all the arithmetic for each element at once. This means that all necessary operations for a particular element are completed in one string of computations. As each new element is produced, the result simply replaces the original value. This process minimizes the calculator's use of intermediate storage. We already know that intermediate storage is expensive for the TI-83, so Crout Factorization is indeed a good choice.

How does the TI-83 solve $Ax=b$ using Crout Factorization? First, the calculator accepts the data in the given matrix A in augmented form (see 2.3 below). Then, Crout performs an Elimination Algorithm to construct an Auxiliary Matrix $A'$ (see 2.4). Crout
then takes this auxiliary matrix and performs a Substitution Algorithm to produce the Solution Vector in the last column of \( A' \) (see 2.6).

Let us look at an example to illustrate these steps. Suppose that we begin with a system of equations, say:

\[
\begin{align*}
2x_1 + 4x_2 - 4x_3 &= 12 \\
x_1 - 4x_2 + 3x_3 &= -21 \\
-6x_1 - 9x_2 + 10x_3 &= -24
\end{align*}
\]

(2.1)

**Step 1:**

This system can be re-written in the form \( Ax = b \):

\[
\begin{bmatrix}
2 & 4 & -4 \\
1 & -4 & 3 \\
-6 & -9 & 10
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
=
\begin{bmatrix}
12 \\
-21 \\
-24
\end{bmatrix}
\]

(2.2)

Or as an augmented matrix:

\[
A|b = \begin{bmatrix}
2 & 4 & -4 & | & 12 \\
1 & -4 & 3 & | & -21 \\
-6 & -9 & 10 & | & -24
\end{bmatrix}
\]

(2.3)

Note: The TI-83 accepts data in the augmented matrix form.

**Step 2:**

By implementing the Elimination Algorithm to find the Auxiliary Matrix, the following values replace the values in \( A \).

\[
A'|b' = \begin{bmatrix}
2 & 2 & -2 & | & 6 \\
1 & -6 & -\frac{5}{6} & | & \frac{9}{2} \\
-6 & 3 & \frac{1}{2} & | & -3
\end{bmatrix}
\]

(2.4)

Since we have not looked at the algorithm yet, the reader will not understand how we have constructed this matrix – having the solution first, however, helps when we work
through the algorithm later. Note: Standard GE gives $A=LU$ — the LU decomposition of a matrix where $L$ is the lower triangular matrix (entries on and below the principal diagonal with zeros in the remaining spaces) and $U$ is the upper triangular matrix (entries on and above the principal diagonal with zeros in the remaining spaces):

$$A' = L \quad U$$

$$\begin{bmatrix}
2 & 4 & -4 \\
1 & -4 & 3 \\
-6 & -9 & 10
\end{bmatrix}
= \begin{bmatrix}
2 & 0 & 0 \\
1 & -6 & 0 \\
-6 & 3 & \frac{1}{2}
\end{bmatrix}
\begin{bmatrix}
1 & 2 & -2 \\
0 & 1 & -\frac{5}{6} \\
0 & 0 & 1
\end{bmatrix}$$  (2.5)

Traditionally, $L$ has the ones in the principal diagonal. The presence of ones in the principal diagonal of $U$ is a trait of Crout Factorization. The advantage of LU decomposition is the following: if $A=LU$, then $LUx=b$. If we then substitute a vector $y$ such that $Ux=y$, we can solve $Ly=b$ using forward substitution and $Ux=y$ using back substitution.

**Step 3:**

We then implement the Substitution Algorithm to find the Solution Vector which replaces last column of the Auxiliary Matrix:

$$A' | b^* = \begin{bmatrix}
2 & 2 & -2 & -4 \\
1 & -6 & -\frac{5}{6} & 2 \\
-6 & 3 & \frac{1}{2} & -3
\end{bmatrix}$$  (2.6)

So how exactly do these algorithms work? How do we calculate the Auxiliary Matrix and the Solution Vector? In the next few pages, we will look at each algorithm and use them as guides to go through the calculations for our example. Let us first look at the algorithm for the Auxiliary Matrix and then look at the algorithm for the Solution Vector.
The Elimination Algorithm to find the Auxiliary Matrix

Some definitions:

\( a_{ij} \) represents the element value in the \( i^{th} \) row and the \( j^{th} \) column of the given matrix

\( n \) is the number of rows in the matrix – determined when the user enters the matrix

\( k, j, i \) are loop counters

The algorithm:

\[
\text{for } k = 1 : n \\
\quad \text{for } i = k : n \\
\quad \quad a_{ik} = a_{ik} - \sum_{j=1}^{k-1} a_{ij} a_{jk} \tag{3}
\]

\[
\text{end} \tag{4}
\]

\[
\text{for } j = k + 1 : n + 1 \\
\quad a_{ij} = (a_{ij} - \sum_{k=1}^{k-1} a_{ik} a_{jk}) / a_{ik} \tag{6}
\]

\[
\text{end} \tag{7}
\]

The function in line (3) calculates the values on and below the principal diagonal, and the function in line (6) calculates the values to the right of the principal diagonal. Do you see the unavoidable division in this step? GE, and subsequently Crout Factorization, requires division. The benefit of Crout is that it does not require intermediate storage. All arithmetic for each entry is completed in one string of operations and the result then overwrites the original value. Note that in the beginning, the user entered all the values of the \( A \) matrix. As the values of \( A' \) are calculated, the new values simply replace the old values. For example, you see in the example that \( a_{12} = 4 \) in \( A \) (2.3). After the calculation is performed (2.4), \( a_{12} = 2 \). This is the new value for \( a_{12} \), and any time \( a_{12} \) is used in the algorithm from this time on, the value will be 2. No use of intermediate storage is the primary advantage of this method.
Expanding the Elimination Algorithm for our example:

\( k = 1 \)

\( i = 1, 2, 3 \) find the values down column 1

\( i = 1 \quad a_{11} = a_{11} \)
\[ a_{11} = 2 \]

\( i = 2 \quad a_{21} = a_{21} \)
\[ a_{21} = 1 \]

\( A' \mid b' = \begin{bmatrix} 2 & \vdots \\ \vdots & \ddots & \ddots \\ \vdots & \ddots & \ddots \end{bmatrix} \]

\( i = 3 \quad a_{31} = a_{31} \)
\[ a_{31} = -6 \]

\( j = 2, 3, 4 \) finds the values across row 1 (right of the diagonal)

\( j = 2 \quad a_{12} = a_{12} / a_{11} \)
\[ a_{12} = 4 \quad / \quad 2 = 2 \]

\( j = 3 \quad a_{13} = a_{13} / a_{11} \)
\[ a_{13} = -4 \quad / \quad 2 = -2 \]

\( j = 4 \quad a_{14} = a_{14} / a_{11} \)
\[ a_{14} = 12 \quad / \quad 2 = 6 \]

\( k = 2 \)

\( i = 2, 3 \) finds the values down column 2 (below row 1)

\( i = 2 \quad a_{22} = a_{22} - (a_{21}a_{12}) \)
\[ a_{22} = -4 \quad - \quad (1)(2) = -6 \]

\( A' \mid b' = \begin{bmatrix} 2 & 2 & -2 & \vdots \\ \vdots & \ddots & \ddots & \ddots \\ \vdots & \ddots & \ddots & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \]

\( i = 3 \quad a_{32} = a_{32} - (a_{31}a_{12}) \)
\[ a_{32} = -9 \quad - \quad (-6)(2) = 3 \]
(k = 2 continued)

\( j = 3, 4 \) finds the values across row 2 (right of the diagonal)

\[
\begin{align*}
    j = 3 & \quad a_{23} = \frac{[a_{23} - (a_{21}a_{13})]}{a_{22}} \\
    & \quad \left( a_{23} = \frac{[3 - (1)(-2)]}{-6} = -\frac{5}{6} \right) \quad A' \mid b' = \begin{bmatrix} 2 & 2 & -2 & 6 \\ 1 & -6 & -\frac{5}{6} & \frac{9}{2} \\ -6 & 3 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \\
    j = 4 & \quad a_{24} = \frac{[a_{24} - (a_{21}a_{14})]}{a_{22}} \\
    & \quad \left( a_{24} = \frac{[-21 - (1)(6)]}{-6} = \frac{9}{2} \right)
\end{align*}
\]

\( k = 3 \)

\( i = 3 \) finds the value down column 3 (below row 2)

\[
\begin{align*}
    i = 3 & \quad a_{33} = \frac{[a_{33} - (a_{31}a_{13} + a_{32}a_{23})]}{a_{32}} \\
    & \quad \left( a_{33} = 10 - \left( -\frac{6}{6} \right) + \left( \frac{3}{6} \right) = \frac{1}{2} \right) \quad A' \mid b' = \begin{bmatrix} 2 & 2 & -2 & 6 \\ 1 & -6 & -\frac{5}{6} & \frac{9}{2} \\ -6 & 3 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \\
    j = 4 & \quad a_{34} = \frac{[a_{34} - (a_{31}a_{14} + a_{32}a_{24})]}{a_{33}} \\
    & \quad \left( a_{34} = \frac{[-24 - (1)(6) + \left( \frac{3}{2} \right)]}{\frac{1}{2}} = -3 \right)
\end{align*}
\]

\( j = 4 \) finds the value across row 3 (right of the diagonal)

\[
\begin{align*}
    j = 4 & \quad a_{34} = \frac{[a_{34} - (a_{31}a_{14} + a_{32}a_{24})]}{a_{33}} \\
    & \quad A' \mid b' = \begin{bmatrix} 2 & 2 & -2 & 6 \\ 1 & -6 & -\frac{5}{6} & \frac{9}{2} \\ -6 & 3 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \\
\end{align*}
\]

**The Solution Vector**

We now have the auxiliary matrix, so we can proceed to find the solution vector. As the new values of this vector are calculated, they overwrite the old values in the last column of the Auxiliary Matrix. The elements of the Solution Vector are determined in reverse order. That is, last element first, next to last, second from last, third from last, etc. The
last element is equal to the corresponding element in the last column of $A'$. Each remaining element is equal to the corresponding element in $A'$ minus the sum of those products of elements in its row in $A'$ and the corresponding elements in its column in the final matrix.

**The Substitution Algorithm to find the Solution Vector**

This step requires nothing more from the user. It simply pulls the old elements from the last column in the Auxiliary Matrix and calculates the new values in the Solution Vector.

Some definitions:

- $F_i$ are the elements in the $i^{th}$ row of the final matrix
- $a_{ij}$ represents the element value in the $i^{th}$ row and the $j^{th}$ column of the auxiliary matrix
- $n$ is the number of rows in the matrix
- $i$ and $k$ are loop counters

The algorithm:

\[
\text{for } k = n : 1 \quad (1) \\
F_k = a_{k,n+1} - \sum_{i=k+1}^{n} a_{ki} F_i \quad (2) \\
\text{end} \quad (3)
\]

**Expanding the Substitution Algorithm for our example:**

\[
k = 3 \quad F_{31} = a_{34} \\
(F_{31} = -3)
\]

\[
A' | b' = \begin{bmatrix}
2 & 2 & -2 \\ 1 & -6 & -\frac{5}{6} \\ -6 & 3 & \frac{1}{2} \\ -3 & -6 & -3
\end{bmatrix}
\]
\[ k = 2 \quad F_{21} = a_{24} - (a_{23}F_{31}) \]
\[ F_{21} = \frac{9}{2} - (\frac{5}{6})(-3) = 2 \]
\[ A' | b' = \begin{bmatrix} 2 & 2 & -2 & | & \text{---} \\ 1 & -6 & -\frac{5}{6} & | & 2 \\ -6 & 3 & \frac{1}{2} & | & -3 \end{bmatrix} \]

\[ k = 1 \quad F_{11} = a_{14} - \left( a_{12}F_{21} + a_{13}F_{31} \right) \]
\[ F_{11} = 6 - \left[ (2)(2) + (-2)(-3) \right] = -4 \]
\[ A' | b' = \begin{bmatrix} 2 & 2 & -2 & | & -4 \\ 1 & -6 & -\frac{5}{6} & | & 2 \\ -6 & 3 & \frac{1}{2} & | & -3 \end{bmatrix} \]

Hopefully, you can now see the strengths of using Crout Factorization in the TI-83 calculator. The primary reason TI selected Crout Factorization was because it does not use intermediate storage. As you have seen in the example used, the only storage utilized is the actual augmented matrix space the user initially inputs. While division is inevitable in this algorithm, the memory we save by not utilizing intermediate storage outweighs the consequences of performing division.
3. Sine and Cosine of $x$

How does the TI-83 evaluate $\sin(x)$ and $\cos(x)$? One way to calculate $\sin(x)$ is by using infinite series expansion, i.e., $\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$. Is this how the calculator solves $\sin(x)$? No. The sum in infinite series expansion does not converge very quickly to an answer, and the approximate answer is only accurate for small values of $x$. This method is not a good method for the TI-83 or any calculator, for that matter.

Instead, Texas Instruments uses the Coordinate Rotational Digital Computer Algorithm, or the CORDIC Algorithm to compute $\sin(x)$ and $\cos(x)$. Jack E. Volder [23] developed the CORDIC Algorithm in 1959. His objective was to build a real-time navigational computer for use on aircraft.

According to Texas Instruments [22], “the CORDIC algorithm is a super example of an approach that is quite different from traditional math and which is very efficient. One often sees a very specialized problem solution like this in many areas of engineering and science, which makes exposure to ideas of this type valuable in education…”

Grant Griffin [11] compares CORDIC with the idea of rotating the angle formed when graphing a complex number, by multiplying it by a succession of constants. Let us look at the general steps CORDIC takes to evaluate the trigonometric functions in question. We know this all starts when the user enters the function to calculate (sine or cosine) and the angle desired. What happens when the user hits the “enter” key?
1. The calculator scales the given angle $\phi$ so that $0 < \phi \leq \frac{\pi}{2}$. The case where $\phi = 0$ is trivial.

2. The calculator then executes a series of "rotations" that result in finding $\tan(\phi)$.

3. The calculator then calculates sine or cosine based on the following identities:

$$\sin(\phi) = \frac{\tan(\phi)}{\sqrt{1 + \tan^2(\phi)}}, \quad \cos(\phi) = \frac{1}{\sqrt{1 + \tan^2(\phi)}}.$$

The mystery of the CORDIC algorithm lies in step 2. What does "rotation" mean? How does the calculator find $\tan(\phi)$? You know that an angle can be expressed as a vector using $X$ and $Y$ coordinates and a vector length of $R$ (see figure 1). Let us assume we are working with a unit vector, so $X = \cos(\phi)$ and $Y = \sin(\phi)$.

![Diagram](image)

**Figure 1**
(from William Egbert [8])

If we could find some way that the calculator could accept the value of $\phi$, and from that value produce the corresponding $X$ and $Y$ coordinates, we could calculate all trigonometric functions.
Figure 2  
(from Egbert [8])

Let us go through some of the background theory behind this algorithm. Suppose we are given the values $x_1, y_1, \phi_1$ (see figure 2). Further suppose we shift the coordinate point $(x_1, y_1)$ to the point $(x_2, y_2)$ with the vector tethered at the origin (i.e. through the angle $\phi_2$). The following equations result from vector geometry to find the new coordinate point:

$$
\begin{align*}
  x_2 &= x_1 \cos(\phi_2) - y_1 \sin(\phi_2) \\
  y_2 &= y_1 \cos(\phi_2) + x_1 \sin(\phi_2)
\end{align*}
$$

(3.1)

Then by factoring out $\cos(\phi_2)$:

$$
\begin{align*}
  x_2 &= \cos(\phi_2) [x_1 - y_1 \tan(\phi_2)] \\
  y_2 &= \cos(\phi_2) [y_1 + x_1 \tan(\phi_2)]
\end{align*}
$$

(3.2)

Earlier, we assumed we were working with a unit vector. When we perform this rotation, it is possible (and quite likely) that we also stretch out the vector beyond the unit circle. How do we compensate for that and ensure the quality of our data?
Let us assign new variables $x_2'$ and $y_2'$ such that:

$$x_2' = \frac{x_2}{\cos(\phi_2)} = [x_1 - y_1 \tan(\phi_2)]$$

$$y_2' = \frac{y_2}{\cos(\phi_2)} = [y_1 + x_1 \tan(\phi_2)]$$

(3.3)

What happens if we take the ratio of $x_2'$ and $y_2'$? Do we get a genuine value for $\tan(\phi_2)$? Even though $x_2'$ and $y_2'$ have this extra constant of proportionality $\frac{1}{\cos(\phi_2)}$ stretching the length of the vector, if we take the ratio of sin to cos, that constant cancels out since it is present in both terms. So we achieve the correct value of tangent:

$$\frac{y_2'}{x_2'} = \frac{y_2}{x_2} = \tan(\phi_1 + \phi_2).$$

(3.4)

The fact that we can shift an existing angle and gather the value of tangent from the new coordinates is the basic theory behind the CORDIC algorithm.

This basic idea just presented considers starting with an angle and adding to it. The actual CORDIC algorithm works backward in that it takes the final angle we want and subtracts angles from it until we get to zero. This leads us to Egbert's [8] discussion of pseudo-division. Pseudo-division is where we divide a given angle $\phi$ into smaller angles. This method has the flavor of bisection, but instead of cutting each progressive angle in half, TI chooses to use smaller angles whose tangents are powers of 10. Since the TI-83 operates in base 10, doing this makes the calculations much faster and simpler by reducing them to shifts and adds or subtracts. The TI-83 stores these special angle values in a table of values:
\[
\tan^{-1}(1) = 45^\circ \\
\tan^{-1}(0.1) \approx 5.7^\circ \\
\tan^{-1}(0.01) \approx 0.57^\circ \\
\tan^{-1}(0.001) \approx 0.057^\circ \\
\tan^{-1}(0.0001) \approx 0.0057^\circ \text{ and so on...}
\]

In pseudo-division, we begin by subtracting 45° from a given angle \( \phi \) until overdraft (in other words, until further subtracting 45° would result in a negative angle), keeping track of the number of subtractions. We then repeatedly subtract 5.7° from the resulting number until overdraft, again keeping track of the number of subtractions. This iteration is repeated with smaller and smaller angles. The resulting sum of angles looks like this:

\[
\phi = q_0 \tan^{-1}(1) + q_1 \tan^{-1}(0.1) + q_2 \tan^{-1}(0.01) + q_3 \tan^{-1}(0.001) + \ldots \quad (3.5)
\]

where \( q_i \) is the number of subtractions for each "decade." The reason we call this pseudo-division is because the process is very similar to division with a changing divisor.

Let us look at an example. Say \( \phi = 51^\circ \).

\[
51^\circ - 45^\circ = 6^\circ \quad q_0 = 1 \\
6^\circ - 5.7^\circ = 0.3^\circ \quad q_1 = 1 \\
0.3^\circ - 0.57^\circ = -0.27^\circ \quad q_2 = 0 \\
\quad \quad 0.3^\circ - 0.057^\circ = 0.243^\circ \quad q_3 = 5 \\
0.243^\circ - 0.057^\circ = 0.186^\circ \\
0.186^\circ - 0.057^\circ = 0.129^\circ \\
0.129^\circ - 0.057^\circ = 0.072^\circ \\
0.072^\circ - 0.057^\circ = 0.015^\circ
\]
\[ 0.015^\circ - 0.0057^\circ = 0.0093^\circ \]
\[ 0.0093^\circ - 0.0057^\circ = 0.0036^\circ \]

So our approximation for the angle is:

\[ \phi = 51^\circ \approx (1)\tan^{-1}(1) + (1)\tan^{-1}(0.1) + (0)\tan^{-1}(0.01) + (5)\tan^{-1}(0.001) + (2)\tan^{-1}(0.0001). \]

The algorithmic equation for this process is

\[ \phi_{n+1} = \phi_n - \tan^{-1}(k), \text{ where } k = 10^{-j} \text{ for } j = 0,1,2,\ldots \quad (3.6) \]

At this point, we have reduced our original angle through iterations of smaller angles. How does this help us? We are still looking for \( \tan(\phi) \) which will lead us to \( \sin(\phi) \) and \( \cos(\phi) \). Let us refer back to equation (3.3). To use these equations, we need to utilize data we have collected from our pseudo-division step. First, we need an initial \( X_1 \) and \( Y_1 \). These coordinates correspond to the residual angle left over from the pseudo-division in (3.5) and (3.6). This residual angle is very tiny (less than 0.0001°), so we can assume \( \phi = 0 \) and set \( X_1 = 1 \) and \( Y_1 = 0 \). Now what about \( \tan(\phi) \)? Equation (3.3) can be iterated where \( \phi \) is the angle whose tangent is \( 10^{-j} \). Egbert [8] calls this process \textit{pseudo-multiplication}. The algorithmic equation is:

\[
\begin{align*}
X_{n+1} &= X_n - Y_n k \\
Y_{n+1} &= Y_n + X_n k
\end{align*}
\quad (3.7)
\]

where \( X_1 = 1, Y_1 = 0 \) and \( k = 10^{-j} \) for \( j = 0,1,2,\ldots \). We begin with these initial values and then subtract a fraction of \( Y \) from \( X \) and add a fraction of \( X \) to \( Y \) for the number of times designated by each multiplier digit. The fractions are created by this \( 10^{-j} \) where \( j \) corresponds to that digit position.
Equation (3.7) is then iterated as many times as the pseudo-division iteration lasts.

We can then take the ratio of the final values of \( X \) and \( Y \) to find \( \tan(\phi) \). The true values for sine and cosine are then found using that \( \tan(\phi) \) (see step 3 from our original description of the CORDIC algorithm).

The following algorithm for calculating \( \tan(\phi) \) is from the TI website [22]:

\[
\begin{align*}
K &= 0 \\
X &= 1 \\
Y &= 0 \\
\text{while } (\phi \neq 0) & \quad (1) \\
\quad \text{while } (\phi < \tan^{-1}(10^{-K})) & \quad (2) \\
\quad \quad K &= K + 1 \\
\quad \text{end while} & \quad (3) \\
\phi &= \phi - \tan^{-1}(10^{-K}) \\
\quad temp &= X \\
\quad X &= X - (10^{-K} \times Y) \\
\quad Y &= Y + (10^{-K} \times temp) \\
\text{end while} & \quad (4) \\
Z &= \frac{Y}{X} & \quad (5)
\end{align*}
\]

You can see in line one that the first coordinate point with which we start is for the angle \( \phi = 0 \), or (1,0). Lines three through four compare the angle given with the table values of angles whose tangents are base 10 values. If an angle so qualifies, \( K \) is incremented.

Line six is the pseudo-division equation. This equation is simply a fixed point add/subtract. Line seven holds a temporary storage for the value of \( X \) since \( X \) is actually modified before we need to use the original value. The lines in eight find the values of the new coordinate after a given rotation. This is the pseudo-multiplication step. These equations consist of digital shifts and an add. Finally, at the very end of the algorithm
after all the arithmetic has been exhausted, the algorithm calculates $\tan(\phi)$ with only ONE division.

Let us look at our original example angle $\phi = 51^\circ$. The following coordinates result if we run this angle through the above algorithm even just a few times:

Do you see how the coordinate points are approaching the original vector? When they ultimately reach it (or come extremely close where the residual is not significant), the algorithm stops, calculates the tangent and then calculates the desired function, sine or cosine.

The primary benefit of using CORDIC is the minimal use of division. The algorithm uses one division to find $\tan(\phi)$ and only one more division to find either $\sin(\phi)$ or $\cos(\phi)$. 
Conclusion

All of us who have worked on this paper have a vested interest in K-12 education. I hope that we have successfully communicated our passion for a greater understanding of mathematic fundamentals. Computational math presents many new topics very different from the pure mathematics taught in high schools. Part of the motivation behind this paper is to create a vision for computational math in K-12 students and teachers. Many students ask the question, "How does my calculator do the things it does?" Our desire is that this paper will help answer that question for at least a few of the more common calculator functions mentioned.

We encourage you, the teacher, to provide these materials to your students. We encourage you, the student, to read this more than once. It is okay if you do not gain a complete understanding the first time through. The authors of this paper took a year to complete it—a great deal of that time was invested in understanding!

We hope that through exposing students to the materials presented, students will be inspired that mathematics is a much larger world than they might have imagined, and the opportunities in mathematics are infinite.
References


