```
X^2*(15*X+9*Y)^3        SIN(3*X)^2     (X+Y)*(5*X-Z)            (5*X+Y/3)^2*X
TAN(X)*SIN(PI/2-X)      dSIN(X)/dX     ∫(8*X+5*Y+13)^4dX        (23*(X+Y)^2+34)^2
55*(X^4-1)/(X^3-1)      (15*X-9)^4     COS(X)^3*SIN(X)^5        4*SIN(PI/2+X-Y)+2
(X^2+2*X+1)/(X^2-1)     TAN(X)         1/X+(X+3)/(X+1)+5*X      ∫55*TAN(X)/CSC(X)dX
SIN(Y)      dX^3/dX     55.3^5         (X^3)^2      COS(PI)     3*X+5*Z      dY^3/dX
967+48       COS(Y)     ∫X^6dX         SEC(X)        -(X+Y)     CSC(X)        X+X/12
COT(X)       21*X+9     X+35*Z         X*X/45                   58*X/X        SIN(2)
12+Z/3      Y*X*Y*X     ∫Z^2dX         512/54                   X*3+14       X^2-53
dTAN(-X)*SEC(-X)/dX     32*Y+X         SIN(X)                   COS(X)       56+4/3
12+3/(X-5/(X+2/X))      X+Y+53         X-X+45                   X*Y+53       TAN(1)
(X+13)*(5-X)*(X+3)      21+3^5         33*X^5                   338-99       Z+Z*45
8/X+7/X^2-3/X+12        COT(X)         (-X)^2        -(X*Y)     SEC(X)       67+763
ATN(1)                  LOG(4)         dX^6/dX      COS(-X)     (X+Z)^2    X*(X+8)
12+673                  X+45*Y         d(COT(X)*CSC(X))/dX      (X+3*Z)^2+(X+3*Y)^4
EXP(2)                  5*SIN(X+Y)     COS(X+Y)-COS(X-Y)        d(95*SEC(X)^2)/dX
d9*X/dX                 (X+Z-13)^2     X*(X+Y)*(X-3*Y)^3        (X^3-3*X^2+X+8)^2
X*Y+X*X                 3*COS(X-Y)     SIN(X)*COT(X)            X/(1+X)+X+X^2

                                                                                  tm
SQR(9)             X+34/X          dZ/dX   ∫9*COT(X)*CSC(X)dX    (Y+Z)^2       X/(X+5)
Y+18*Z             132*95        CSC(-X)   SQR((X^2+1)^2)*X/4    X+X+Y+Z       X*X+Y*Z
SIN(PI)          X/X^2+5        4*X^3+5    SIN(X)^2+COS(-X)^2    123+443       X/(X+3)
COS(-PI)        (X+18*Z)      SIN(-8*X)    (12+X+X^2+X^3)^2-X    LOG(9)         3.1415
(3+X^2)^2     4/X^2-1/X      X+X*X/X^4     X+X    ((-X))    123  X+X/13        Y+Y*33
TAN(X)^2-1   23/(X+3/X)     SEC(X)^(-2)           4*5*66         5+Z/34        Z+X+15
15*SEC(2*X)^2*SIN(X)^2      1+X+X^2+X^3           X+Y+43         COT(X/2)-COT(X)+X-X
-X+2/X-1/(X-1)+1/(X+2)     SIN(X)  34+173         COT(X)         X+X+Y-3*Y+4*X-3*Z+8  -80
(X-13)*((X+1)^2-1)^2+X     X^7/X    X+X+Z         22*X*Y         COS(X+Y)-SIN(X-Y)+2
(2*X+Y-2)^2*(X-3*Y+13)    ∫X*ZdZ    TAN(X)        X^5-18         (X+1)*(X+2)*(X+3)*X
COS(Y)    X+21    367890   ∫57dX      278*X       dPI/dX         1/X+33         SEC(X)
Y+Z-42     PI    Y/Y+30   (X-3)^3/(3*(X+1))       SIN(Y)         33^2-5        X+33*Z
2+2+22            d33/dX  SIN(X)*COS(1/2+X)       Y*X/82         X+34+Y        TAN(X)
COT(X)            357+68  (((X+3)*X-2)*X+8)*X     SQR(2)         SIN(Y)        Z+67*Z
(X+4)^5          (Y-5)^4  TAN(ATN(X-13)/(X-13))   4-(-X)         SEC(-X)      (9*Y)^4
2*X+3*Y          (X/8)^5  X+Y+2*Z       LOG(PI)   X-33/X         (4*X)^6     Y*(X+Z)
2+PI/25          CSC(-X)  X+32/X         Y+31*X   X+Y+59         (Z+9)^2      d9*Y/dX


X*Y*33    Y*Z+145   Y+2*Z/8   COS(13)   Z+X*44    8*X/223   X*X    37    X+2/X     TAN(-X)
34    59  87        35        +X        99    36  73        37*X   54    96     Y  55
TAN(X)    3*X+45    Z*Y*45    SEC(X)    X+X*21    LOG(4)    TAN(X)^2     75        3*X+56
12    45  12        74        12        12    45  12        13     53*X  12     6  12
57    56  2/(X+9)   12        632+536   38    +6  3*X*X^5   42     X/X   Y+Y*Y     Z*Y/Y+8

          84     18      X      Y*Z     39    75     25       X        73
          X*8    4*Z     8-9    56-X    73    92     77       X-Y      64
          Y+33  48*X     49 63  90 32   84    52     97      19 23     85
          44  Y*Y  68   COS(23)  33    44^5   27     63    dX^3/dX     13
          66   Y   32   42   35  14     Y*X       dY/dX    47    74    TAN(X)
```

Copyright Notice

Trademark Notice

Disclaimer

TABLE OF CONTENTS

1. INTRODUCTION

PICOMATH-80 is a package consisting of four demonstration computer-algebra programs originally written in BASIC:

1. The **polynomial** program can expand an expression such as

   $$(2\ x - y) \cdot (x + y)^2 - (28\ z - 1)^2$$

   into the equivalent polynomial

   $$2\ x^3 + 3\ x^2y - y^3 - 784\ z^2 + 56\ z - 1$$

2. The **rational** program can expand and simplify an expression such as

   $$1 + \frac{1}{x-1} - \frac{1}{x+1} + \frac{2x}{x^2-1}$$

   into the equivalent ratio of two polynomials, reduced to lowest terms

   $$\frac{x+1}{x-1}$$

   (Although the rational program also treats polynomials as a special case, the polynomial program can accommodate more variables. Moreover, the polynomial program permits higher degree on most implementations, and the polynomial program is generally more accurate when both programs are applicable.)

3. The **trigonometric** program can expand and simplify an expression such as

   $$\frac{1 + \tan^2 x}{1 + \cot^2 x}$$

   into the equivalent standard form

   $$\sec^2 x - 1$$

   (As a special case, alleged trigonometric identities may be proved or disproved by expanding and simplifying the difference in their two sides.)

4. The **Fourier** program performs trigonometric transformations that complement those provided by the trigonometric program. Specifically, the Fourier program transforms polynomials in sines and cosines of x and integer multiples thereof into a linear combination of such sines and cosines -- ie. the reverse of multiple-angle expansions. For example, the program transforms

$$64 \sin^4 x \cos^3 x$$

into

$$3 \cos x - 3 \cos(3 x) - \cos(5 x) + \cos(7 x)$$

All but the rational program can also symbolically differentiate and integrate such expressions according to the rules of calculus. However, a knowledge of trigonometry or calculus is unnecesssary for utilizing the more elementary features of the PICOMATH package.

PICOMATH is small enough to run on virtually every computer, and the original source code written in BASIC is easily adapted to other languages besides BASIC.

Section 2 of this reference manual is a user's guide, elaborating on the brief usage instructions printed by the programs. Section 3 is a statement of the generous conditions under which copying of this document and/or installing the package on an additional machine are permitted. Section 4 is a brief guide for those who wish to learn more about computer algebra in general. Section 5 is a discussion of the potential uses of computer algebra in education. Section 6 is an explanation of how PICOMATH works. Section 7 contains instructions for installing and testing the generalized programs listed in Section 8.

Mere usage of an installed version of the programs requires at most study of sections 1 and 2. Installation of the program requires the additional use of sections 7 and 8. Section 6 can be explored by those who are curious about how PICOMATH works and are experienced in programming together with numerical analysis.

2. USAGE

The polynomial program presumes familiarity with the concepts of adding, subtracting, multiplying and expanding integer powers of polynomials. The rational program further presumes familiarity with the concepts of placing rational expressions over a common denominator and cancelling the greatest polynomial factor that evenly divides the resulting numerator and denominator. The trigonometric and Fourier programs further assume familiarity with radian measure, the relations between the various trig functions, and the concept of exploiting various identities such as multiple-angle formulas to simplify trig expressions. Usage of the differentiation and integration options of the polynomial, trigonometric and Fourier programs further requires familiarity with elementary calculus.

This reference manual also presumes familiarity with use of the computer and terminal employed, together with the language in which PICOMATH is implemented: Specifically, you are assumed to know how to start the computer, load BASIC or whatever is used to implement PICOMATH, then LOAD and RUN a saved program written in that language. You are also required to know how to modify an assignment statement of such a program.

Given an appropriate math background, the easiest way to learn to use a PICOMATH program is to simply **LOAD** and **RUN** it, then follow the displayed instructions. This section is merely an elaboration of those brief instructions. This description applies to a typical interactive BASIC implementation. Other environments may entail slight or not-so-slight differences that we cannot predict. Such differences may be described in a supplement attached to this manual, prepared by whoever adapted PICOMATH to the environment.

2.1 Using the Rational Program

Typically, after loading the rational program you replace the example line 20 with an analogous line that assigning to variable A an expression that you want to simplify to a ratio of two polynomials in X, reduced to lowest terms. For example:

```
20  A = 1 + (X*(X+1)^2 + X^4 + 1) / (X^4 - 1)
```

(Note that the circumflex "^", designating raising to a power, is designated by an upward pointing arrow or a pair of asterisks on some systems.)

After replacing line 20, you enter the **RUN** command, causing the program to display a simplified equivalent such as A =

```
2 X ^ 2  +  X
--------------------------------------------
X ^ 2  -  1
```

Note that the multiplication asterisk is suppressed in output but must not be omitted in line 20. Also, you must rename the one variable in your problems as X if not already named that.

Note how the expansion process includes cancellation of the polynomial greatest common divisor from the numerator and denominator so that we do not get incompletely simplified results such as

```
2 X ^ 4  +  X ^ 3  +  2 X ^ 2  +  X
---------------------------------------------
X ^ 4  -  1
```

Moreover, results are usually **normalized** so that either the numerator or denominator has a leading coefficient of 1.

Roundoff error in the underlying BASIC arithmetic may cause some of the coefficients in the result to be inexact. For example, the above result might display as

```
2.00038 X ^ 2  +  0.999473 X  -  6.35251E-4
---------------------------------------------
X ^ 2  +  4.82352E-4 X  -  1.00079
```

In an attempt to minimize such annoying distractions, the program replaces relatively miniscule coefficients by zero, while replacing nearly integer coefficients by the nearby integers. However, an exact result could truly entail relatively small or nearly integer coefficients, so the program imposes rather stringent thresholds before making these adjustments. Thus, inexact coefficients may appear despite the adjustment process.

If the degrees of the resulting numerator and denominator are modest and if all coefficients in the unexpanded expression are integers of small magnitude, then the presence of slight inexactness in the corresponding expanded result is often obvious from inspection, in which case the appropriate "nearby" exact result is also often obvious. (For example, the above inexact result is obviously nearly $(2x^2+x)/(x^2-1)$.) In contrast, the sizes of roundoff errors in a result are usually not at all obvious if the unexpanded expression contains noninteger coefficients or if the result contains coefficients of large magnitude. Consequently, to help identify problems for which roundoff errors have yielded unacceptable accuracy, the program compares the unexpanded and expanded formulas at some sample points, displaying a warning message if any discrepancy at these samples is large relative to the largest sample magnitude. Although extremely unlikely, these discrepancies all could be relatively small even though other samples would have revealed serious inaccuracy. Thus, **always inspect results for reasonableness** as an extra precaution.

Whether or not such a warning message is printed, the program finally breaks execution, returning control to the executive command level after displaying an epilogue message instructing the user to modify line 20 and rerun if desired. This message also announces bounds on the maximum degrees of the resulting numerator and denominator that the program is capable of accommodating. These degree bounds are typically slightly less than half the number of significant decimal digits provided by the underlying BASIC arithmetic when applying the exponentiation operator, "^". This accuracy may be less than for the other arithmetic operators -- particularly in double precision, especially if "^" is evaluated using exponentials and logarithms even for integer exponents. Moreover, even for results that fall within these degree bounds, accuracy depends strongly upon whether or not the given and resulting coefficients are integers of relatively small magnitude and upon whether or not the zeros and poles of the expressions are favorably distributed between the sample points. Consequently, common **7-digit single-precision arithmetic permits only modest results to be accurately determined and displayed.** In contrast, the polynomial program accommodates results having a numerator of degree 6 in X, and the polynomial program is usually more accurate when applicable.

The accuracy of all four PICOMATH programs is more dependent upon the simplicity of the exact result than upon the simplicity of the given expression.

Limitations of the underlying arithmetic entail another hazard: To determine the expanded form of the formula on line 20, PICOMATH **evaluates** that unexpanded formula at several values of X, then **interpolates** or **fits** a ratio of two polynomials to these samples. Unfortunately, at a particular sample point the unexpanded expression or one of its subexpressions may exceed the BASIC arithmetic magnitude limit, thus causing **overflow.** This phenomenon is particularly likely if a sample happens to fall near a point where a denominator becomes zero. Depending upon the particular BASIC implementation, the resulting action may be:

1. Display a warning message, use the correctly signed number having the largest representable magnitude in place of the unrepresentable intermediate result, then proceed.

2. Display a warning message, use some less appropriate number in place of the unrepresentable result, then proceed.

3. Same as 1, except without a warning message.

4. Same as 2, except without a warning message.

5. Display an error message, then interrupt execution, returning control to the command level.

In any event, you are unlikely to proceed to a sufficiently accurate result to avoid the "serious discrepancy" warning message.

A related possibility is for a nonzero intermediate result to have a magnitude too small to represent in the arithmetic, thus causing **underflow.** Depending upon the BASIC implementation, the resulting action may be:

1. Display a warning message, use zero in place of the unrepresentable intermediate result, then proceed.

2. Same as 1, except without a warning message.

3. Display an error message, then interrupt execution, returning control to the command level.

Here too you are unlikely to proceed to a sufficiently accurate result to avoid the "serious discrepancy" warning message. Moreover, if an intermediate result is replaced by zero, a subsequent attempt to divide by that intermediate result causes a **zero-divide** error. Depending upon the BASIC implementation, the resulting action may be:

1. Display a warning message, use some arbitrary representable number in place of the undefined intermediate result, then proceed.

2. Same as 1, except without a warning message.

3. Display an error message, then interrupt execution, returning control to the command level.

Here too, you are unlikely to proceed to a sufficiently accurate result to avoid the "serious discrepancy" warning message. Moreover, a zero-divide error could arise directly from a sample coinciding with a zero of a denominator, rather than indirectly via underflow. However, this cause is quite unlikely because the program avoids trivial sample values.

Overflow, underflow, or zerodivide can occur after the expanded expression has been printed because sampling is used also to test for serious discrepancy. However, this situation is less serious because at least we then have an expanded result that was computed without incurring these difficulties.

For all four PICOMATH programs the expectation of overflow, underflow, zerodivide and excessive roundoff error is diminished by formulating the problem in a manner that employs only small-magnitude integer coefficients and low degree as much as possible. For example:

1. Perhaps a trivial common denominator can be identified by inspection and removed along with a common divisor of the resulting coefficients before submitting the problem to PICOMATH.

2. Perhaps the problem only involves even powers of X, so that the degree can be halved by merely replacing X^2 with X before submitting the problem.

Simple substitutions often permit application of this program to problems that are otherwise outside its domain. For example, replacing c^X by X transforms the expression

$$1 + \frac{c^x(c^x+1)^2 + c^{4x} + 1}{c^{4x} - 1}$$

into the one at the beginning of this subsection. The inverse substitution on the simplified result then yields the expression

$$\frac{2\ c^{2x} + c^{x}}{c^{2x} - 1}$$

The formula on line 20 can even entail irrational operations if the formula is defined at all of the sample values and if the formula can be simplified to a single rational expression interpolating those sample values. For example:

```
20 A = EXP(LOG(X^6+3*X^4+3*X^2+1))^(1/3) / TAN(ATN(X^2+1))
```

This possibility is of course of limited utility because most expressions involving irrational operations are undefined over some intervals for X or because the expressions do not simplify to a ratio of polynomials even over a somewhat restricted interval for X.

> **Warning:** Some implementations employ **integer division** when both operands are of integer type, causing (1/3) to be interpreted as zero in the above example. Experimentally determine how your implementation behaves in this regard, and if necessary be sure to make at least one operand of "/" be floating point whenever that is intended.

Here are some examples that should work on any reasonable 7-digit floating-point arithmetic, thus helping to test your implementation of the rational program or reveal weaknesses in your built-in arithmetic:

$$\frac{x^4 - 1}{x^3 - 1} \longrightarrow \frac{x^3 + x^2 + x + 1}{x^2 + x + 1},$$

$$\frac{(x+1)\cdot(x-2)\cdot(x+3)}{(x-1)\cdot(x+2)\cdot(x-3)} \longrightarrow \frac{x^3 + 2\ x^2 - 5\ x - 6}{x^3 - 2\ x^2 - 5\ x + 6},$$

$$1 + x\ /\ \{2 - x/[3+x/(4-x)]\} \longrightarrow \frac{-\ x^2 + 4\ x + 24}{x^2 - 8\ x + 24},$$

$$\frac{3 - 13/x - 10/x^2}{1 - 2/x - 15/x^2} \longrightarrow \frac{3\ x + 2}{x + 3},$$

$$\frac{4\ x}{(x+2)\cdot(x-2)} - \frac{3\ x}{(x-2)\cdot(x+1)} + \frac{2}{(x+2)\cdot(x+1)} \longrightarrow \frac{1}{x + 1},$$

$$\frac{\dfrac{x^2 - 5\ x - 6}{x^2 - 2\ x - 15} \cdot \dfrac{x^2 - 7\ x + 10}{x^2 + 5\ x + 4}}{\dfrac{2\ x - 12}{x^2 + 3\ x}} \longrightarrow \frac{x^2 - 2\ x}{2\ x + 8},$$

$$\frac{1 + \cfrac{1}{1 + \cfrac{1}{x-1}}}{\cfrac{1}{1 - \cfrac{1}{x+1}}} \to \frac{2\ x - 1}{x + 1},$$

$$\frac{2\ x - \cfrac{3\ x + 4}{x - 2}}{x - \cfrac{10\ x + 4}{2\ x + 3}} \to \frac{2\ x + 3}{x - 2},$$

2.2 Using the Polynomial Program

Usage of the polynomial program is similar to the rational program, except with the added option of requesting differentiation or integration. Typically, after loading the polynomial program you replace the example line 20 with an analogous line assigning to variable A an expression that you want to simplify to an expanded polynomial in X, Y and Z, then perhaps also differentiate or integrate. For example,

```
20 A = (X + Y) ^ 4
```

Then, you enter the BASIC **RUN** command, after which the program interactively asks the user to enter **E** for **E**xpansion, **D** for **D**ifferentiation or **I** for **I**ntegration, as desired. After receiving a valid response to this query, the program proceeds to determine then display the desired result, such as respectively

```
A = X^4 + 4 X^3 Y + 6 X^2 Y^2 + 4 X Y^3 + Y^4
```

or

```
dA/dX = 4 X^3 + 12 X^2 Y + 12 X Y^2 + 4 Y^3
```

or

```
S A dX = 0.2 X^5 + X^4 Y + 2 X^3 Y^2 + 2 X^2 Y^3 + X Y^4
```

Here "S" denotes the integral sign, "$\int$". Note that the arbitrary constant of integration is suppressed as is customary in integral tables.

The polynomial program uses techniques similar to the rational program to minimize yet detect excessive consequences of the underlying limited-precision, limited-magnitude BASIC arithmetic. For example, the polynomial program prints a warning message if evaluation of the given and expanded formulas at any of several extra sample points yields a relatively large discrepancy.

Whether or not such a warning message is displayed, the program finally breaks execution, returning control to the command level after displaying an epilogue message instructing you to modify line 20 and rerun if desired. This message also announces that in the expanded equivalent of line 20, any term containing X must be of total degree <= 6, any term containing Y must be of total degree <= 4, and any term containing Z must be of total degree <= 2. For example, X*Y*Z is not a representable term. Thus, it may be necessary to rename variables in the original statement of a problem so that X has the highest degree, Y has the next highest degree, and Z is the remaining variable, if any.

Simple substitutions often make this program applicable to problems that are initially outside its scope. For example, replacing X^2 by X permits you to determine the coefficients in the expansion of (X^2 + 1)^6. As another example, ((X+Y)^4+17)^2 can be expanded with some manual assistance by separately computing (Z+17)^2 and (X+Y)^4, then manually substituting the latter result for Z in the former result, followed by a manual expansion of Z^2 therein.

Do you interpret $-x^2 + 5$ as meaning -(x^2) + 5 or (-x)^2 + 5, which simplifies to $x^2 + 5$? As with most people and most programming language implementations, PICOMATH **output** employs the former interpretation. However, beware that some programming language implementations employ the latter interpretation, and that PICOMATH **input** utilizes the built-in precedence rules. Experimentally determine the local built-in rules, then parenthesize if necessary to accomplish the desired effects. Also, although PICOMATH does not produce nested exponentiations as output, some implementations permit unparenthesized repeated exponentiations in line 20 of the program. If so, be sure to learn whether X^2^3 is interpreted as X^(2^3) or (X^2)^3, then parenthesize if necessary to achieve the desired effect.

The polynomial program uses polynomial interpolation, just as the rational program uses rational interpolation. However, unlike rational expressions, a polynomial does not entail the embarrassing possibility of being undefined for finite values of its variables. Thus, the program can use integer values of small magnitude as sample values for the polynomial variables, which usually helps reduce roundoff errors. Unfortunately, overflow, underflow or zero-divide errors are possible. For example, even the relatively benign appearing formula

```
20 A = (1E-7 * X) ^ 6 + (X + 1E7) ^ 6
```

produces both underflow and overflow when sampled at X=1, using arithmetic that limits nonzero magnitudes to lie between about 10^{-39} and 10^{38}, as is quite common.

The formula on line 20 can entail non-polynomial subexpressions if the formula is defined at all of the sample values and if the result can be simplified to a polynomial. For example, to determine if

$$2 x^3 - 3 x + 4$$

exactly divides

$$6\ x^5 + 8\ x^4 + 13\ x^3 + 22\ x - 8,$$

(and if so to determine the resulting quotient), we can use

```
    20 A = (6*X^5 + 8*X^4 + 13*X^3 + 22*X - 8)/ (2*X^3 - 3*X + 4)
```

Then, the result is the quotient unless a "serious discrepancy" warning indicates that exact division is impossible. A zerodivide or overflow message for such an example is probably due to a sample point coinciding with a **zero** of the divisor.

Here are some examples that should work on any reasonable 7-digit floating-point arithmetic, thus helping to test your implementation of the polynomial program:

$(3x+2)^6$ —> $729\ x^6 + 2916\ x^5 + 4860\ x^4 + 4320\ x^3 + 2160\ x^2 + 576\ x + 64,$

$(x+1)\cdot(x+2)\cdot(x+3)\cdot(x+4)\cdot(x+5)\cdot(x+6)$

—> $x^6 + 21\ x^5 + 175\ x^4 + 735\ x^3 + 1624\ x^2 + 1764\ x + 720,$

$(15\ x - 16)^3$ —> $3375\ x^3 - 10800\ x^2 + 11520\ x - 4096,$

$(x + 23)^3$ —> $x^3 + 69\ x^2 + 1587\ x + 12167,$

$(x+1)^6 + (x+y+1)^4 + (x+y+z+1)^2$ —> $x^6 + 6\ x^5 + 16\ x^4 + 4\ x^3y + 24\ x^3$

$+ 6\ x^2y^2 + 12\ x^2y + 22\ x^2 + 4\ xy^3 + 12\ xy^2 + 14\ xy + 2\ xz + 12\ x$

$+ y^4 + 4\ y^3 + 7\ y^2 + 2\ yz + 6\ y + z^2 + 2\ z + 3,$

$\frac{d}{dx}\ [(x+1)^6 + (x+y+1)^4 + (x+y+z+1)^2]$

—> $6\ x^5 + 30\ x^4 + 64\ x^3 + 12\ x^2y + 72\ x^2 + 12\ xy^2$

$+ 24\ xy + 44\ x + 4\ y^3 + 12\ y^2 + 14\ y + 2\ z + 12,$

$\int [(x+1)^6 + (x+y+1)^4 + (x+y+z+1)^2]\ dx$

—> $0.142857\ x^7 + x^6 + 3.2\ x^5 + x^4y + 6\ x^4 + 2\ x^3y^2 + 4\ x^3y$

$+ 7.33333\ x^3 + 2\ x^2y^3 + 6\ x^2y^2 + 7\ x^2y + x^2z + 6\ x^2 + xy^4$

$+ 4\ xy^3 + 7\ xy^2 + 2\ xyz + 6\ xy + xz^2 + 2\ xz + 3\ x.$

2.3 Using the Trigonometric Program

Use of the trigonometric program is similar to the polynomial program, except that the given and resulting expressions are trigonometric rather than polynomials. Typically, after loading the trigonometric program the user replaces the example line 20 with an analogous line assigning to variable A an expression that the user wants simplified to a **linear combination** of the terms

$$1,\ \sin x,\ \cos x,\ \tan x,\ \cot x,\ \csc x,\ \sec x,$$
$$\sin x \cos x,\ \sin^2 x,\ \sec^2 x,\ \tan x \sec x,\ \cot x \csc x,$$
$$\sin y,\ \cos y,\ \sin x \sin y,\ \sin x \cos y,\ \cos x \sin y,\ \cos x \cos y,$$

meaning a sum wherein each term is one of the above, perhaps multiplied by a numerical coefficient. These terms are **linearly independent,** meaning none of them can be expressed as a linear combination of the others. Consequently, when the nonzero terms are displayed in a fixed order, such a linear combination provides a unique **canonical** (meaning **standard**) form for all expressions that are equivalent to any such linear combination. As a special case, any expression equivalent to zero simplifies to zero, so candidate trig identities may be verified or disproved by determining if the difference in their sides simplifies to zero. Moreover, many of the nonzero trig expressions prevalent in trigonometry texts are equivalent to such a linear combination. (For example, cos(2 x) and sin(x-y+π/2) are in this class.) Also, although simplicity is in the eye of the beholder, the resulting linear combination is often the simplest possible form of the original expression for most purposes. At any rate, the result provides a possibly different alternative to the original form, which can then serve as a point of departure for further transformations using the Fourier program or manual techniques.

When entering a trig expression on line 20, note that most BASIC implementations:

1. do not provide built-in secant, cosecant and cotangent function;

2. require all user-defined functions to be spelled beginning "FN";

3. permit only one letter following FN.

Thus, either you may have to avoid these three trig functions in favor of their reciprocals, or you may have to suffer names such as FNE for sEc, FNS for cSc, and FNO for cOt. At best, you will probably have to use names such as FNSEC, FNCSC and FNCOT. Experimentation will quickly determine the situation for a particular BASIC implementation.

As an example of program usage, you could modify line 20 to be

```
20 A = 1 - COS(PI/2-X)^2 / (1 + COS(X))
```

Note how the trig program presumes radian measure, and how we can use "PI" to represent the ratio of the circumference to diameter of a circle. **(This may be spelled "PI" or merely "P" on BASIC implementations that do not permit multiple-letter variable names.)**

Next, enter the **RUN** command, after which the program interactively asks the user to enter **E** for **E**xpansion, **D** for **D**ifferentiation, or **I** for **I**ntegration, as desired. After receiving a valid response to this query, the program proceeds to determine then display the desired result, such as respectively

A = COS X

or

dA/dX = -SIN X

or

S A dX = SIN X

where "S" denotes the integral sign, "$\int$". Note that the arbitrary constant of integration is suppressed as is customary in integral tables. Note also that although parentheses around simple function arguments are omitted in PICOMATH output for brevity, most implementations require them in line 20. PICOMATH output similarly abbreviates powers of functions, such as using SIN^2X rather than the equivalent SIN(X)^2 probably required in line 20.

As with the rational program, the trigonometric program entails the possibility of underflow, overflow or zerodivide, because line 20 may contain tangents, secants, cotangents, cosecants, and denominators that are zero or nearly so at sample points.

Here are some examples that should work on any reasonable 7-digit floating-point arithmetic, thus helping to verify your implementation of the trigonometric program:

$$\frac{\tan^2 x}{1 + \tan^2 x} \;\text{-->}\; \sin^2 x,$$

$$\tan x \sin x + \cos x \;\text{-->}\; \sec x,$$

$$\sin\left(\frac{x+y}{2}\right) \cos\left(\frac{x-y}{2}\right) \;\text{-->}\; 0.5 \sin y + 0.5 \sin x,$$

$$\frac{\sin(x+3\pi/2)}{\cos(x+\pi)} + \frac{\cos(x-3\pi/2)}{\sin(x+\pi/2)} \;\text{-->}\; -\tan x + 1,$$

$$\frac{1 + \cos(2\,x)}{\cos x} + \frac{\sin(2\,x)}{\sin x} \;\text{-->}\; 4 \cos x,$$

$$1 - \frac{\sin^2 x}{1 + \cos x} \;\text{-->}\; \cos x,$$

$$\frac{\sin x + \cos x \tan x}{\tan x} \longrightarrow 2 \cos x,$$

$$\cos^4 x - \sin^4 x \longrightarrow 1 - 2 \sin^2 x,$$

$$2 \cos^2(x/2) - 1 \longrightarrow \cos x,$$

$$(\csc^2 x - 1) \tan x \longrightarrow \cot x,$$

$$(\sec x - \cos x) \cdot (\tan x + \cot x) \longrightarrow \tan x \sec x,$$

$$(\tan x + \cot x) \sin x \cos x \longrightarrow 1,$$

$$\cot (x/2) \longrightarrow \cot x + \csc x,$$

$$\frac{1 + \tan^2 x}{1 + \cot^2 x} \longrightarrow \sec^2 x - 1,$$

$$\sec^4 x - \tan^4 x \longrightarrow 2 \sec^2 x - 1,$$

$$\cot x + \frac{\sin x}{1 + \cos x} \longrightarrow \csc x,$$

$$\frac{d}{dX} [\csc (2 x) \cos x] \longrightarrow -0.5 \cot x \csc x,$$

$$\frac{\sec x - \tan x - 1}{\tan x + \sec x - 1} \longrightarrow \tan x - \sec x,$$

$$\int \tan x \, dx \longrightarrow - \ln \cos x.$$

2.4 Using the Fourier Program

Use of the Fourier program is quite similar to the trigonometric program. In fact, the Fourier program is actually another trigonometry program that tends to transform expressions into a different form than that of the "trigonometric" program: The Fourier program transforms products and powers of sines and cosines into linear combinations of sines and cosines of X and integer multiples of X -- the opposite of multiple-angle expansions. The name derives from the applicability to **Fourier analysis** and the desire to avoid confusion with the other program.

As with the trigonometric program, you replace line 20 with an analogous line assigning a trigonometric expression in X to variable A, then enter the **RUN** command and answers the query choosing Expansion, Differentiation or Integration. The program then proceeds to display the corresponding "Fourier expansion".

The Fourier program uses techniques similar to the other programs to minimize yet detect excessive consequences of the underlying limited-precision BASIC arithmetic. For example, the Fourier program displays a warning message if evaluation of the given and expanded formulas at any of several extra evaluation points yields a relatively large discrepancy.

Whether or not such a warning message is displayed, the program finally breaks execution, returning control to the command level after displaying an epilogue message instructing the user to modify line 20 and rerun if desired. This message also announces a bound on the maximum multiplicity of angle X that the program can produce as an argument of a sine or cosine. This multiplicity is set in the program as the value of variable named M, which can be increased or decreased by the user if desired. (Total computing time increases approximately quadratically with M, and roundoff error also increases with M.)

The Fourier program uses an interpolation technique analogous to those used by the other programs. However, the terms

$$\{1, \cos x, \sin x, \cos 2x, \sin 2x, \cos 3x, \ldots\}$$

are all bounded and numerically quite distinguishable over the sampling interval, compared to the terms employed by the other programs. Consequently, the Fourier program is usually far less subject to excessive roundoff errors, overflow, underflow or zerodivide.

Expressions that can be transformed to the appropriate trigonometric form are expressions equivalent to polynomials in sines and cosines of angles having the form

$$n\,X + c$$

where n is integer and c is a constant, perhaps involving PI, which may be spelled Pl or P in some implementations. Thus

$$(2.7 + \tan(x+1.5)\ \cos(x+1.5) + \sin(-3x + \pi/13))^2$$

is within the representable class, whereas

$$\frac{\sin(1.5x)}{\sin x + \cos(2x)}$$

is not. Although some expressions are representable using both the trigonometric and Fourier programs, each program is applicable to expressions that the other is not. Consequently, it is often worth trying the other one if the first choice does not simplify the expression enough.

One use of this program is to help prove or disprove trigonometric identities: After manually simplifying the difference in the two sides of an alleged identity into a ratio of two appropriate trigonometric polynomials reduced to lowest terms, you can independently submit the numerator and denominator to the program to determine whether or not they are equivalent to zero. The alleged identity is true if only the numerator is equivalent to zero, whereas the alleged identity is false if neither the numerator nor denominator is equivalent to zero.

As already mentioned, another use of this program is for **spectral** or **harmonic Fourier analysis.** If line 20 represents a function of periodicity 2π, then the corresponding display **approximately** represents its truncated Fourier expansion -- perhaps the exact expansion if it is finite. (It is of course trivial to scale the independent variable of any periodic function so that its period is 2π, so there is no essential lack of flexibility in this regard.) For example, to determine the Fourier expansion of $\sin^8 x$, we simply use

```
20 A = SIN(X) ^ 8
```

whereas to determine the **approximate** Fourier expansion of the **rectified** sine wave $|\sin(x/2)|$ we merely use

```
20 A = ABS(SIN(X/2))
```

Here are some examples that should work on any reasonable 7-digit floating-point arithmetic, thus helping to test your implementation of the Fourier program:

$$2 \sin (5 x) \cos (3 x) \longrightarrow \sin (2 x) + \sin (8 x),$$

$$(\cos x + \sin x)^8 \longrightarrow 4.375 + 7 \sin (2 x) - 3.5 \cos (4 x) - \sin (6 x) + 0.125 \cos (8 x),$$

$$\frac{d}{dx} [\sin (5 x) \cos (3 x)] \longrightarrow \cos (2 x) + 4 \cos (8 x),$$

$$\int [\sin (5 x) \cos (3 x)]\, dx \longrightarrow -0.25 \cos (2 x) - 0.0625 \cos (8 x),$$

$$\frac{d}{dx} [(\cos x + \sin x)\hat{}8] \longrightarrow 14 \cos (2 x) + 14 \sin (4 x) - 6 \cos (6 x) - \sin (8 x),$$

$$\int [(\cos x + \sin x)\hat{}8]\, dx \longrightarrow 4.375 x - 3.5 \cos (2 x) - 0.875 \sin (4 x) + 0.166667 \cos (6 x) + 0.015625 \sin (8 x).$$

3. COPYING POLICY

The most convenient and economical way for an individual to acquire software is to personally duplicate the printed documentation and a machine-readable form of the program if copies and duplication facilities are locally available. On the other hand, software manufacturers need compensation for their creative efforts to cover the development and advertising expenses. Accordingly, we have devised a simple means of providing customers with this convenience and economy while retaining a clear conscience and incurring no anxiety about the severe penalties for copyright violation:

We hereby grant end users permission to duplicate the PICOMATH-80 Reference Manual for a royalty of $1 per copy or portion thereof, provided each copy includes at least sections 1 and 3 in their entirety.

We hereby grant end users permission to install PICOMATH-80 for a royalty of $2 per machine on which it is installed or used, provided the installation is performed in accordance with the directions of section 6 of the PICOMATH-80 Reference Manual.

If you wish to avail yourself of these opportunities, merely mail a check, money order, or even cash, to The Soft Warehouse at Box 11174, Honolulu, Hawaii 96828. We even offer a 33% discount for royalties totaling $6 or more, which should benefit schools, clubs, or organizations having more than one computer.

This generous software distribution honor system is unique so far as we know. We hope that the experiment works, which would encourage us and other software manufacturers to distribute other software in this convenient manner in the future.

Having end users duplicate the software saves us substantial expense, so we are glad to permit it. However, we plan to continually improve PICOMATH, so you may prefer to obtain a new copy of the program at a computer store or from your licensed hardware manufacturer or from Programma International at 2908 N. Naomi St., Burbank, California, 91504. Each copy includes an extensive reference manual explaining how PICOMATH works and how to adapt the generalized program listings therein to other languages and computers. Also, the machine-readable version provided with the manual may have enhancements that exploit extra accuracy, memory space, or special features of the particular implementation for which it is intended. The Soft Warehouse does not directly distribute manuals or machine-readable versions of PICOMATH to end users.

Licenses for distributing machine-readable versions for additional specific languages on specific machines are available from the Soft Warehouse to hardware manufacturers or major software distributors.

4. HOW TO LEARN MORE ABOUT COMPUTER ALGEBRA

We expect that many who experience PICOMATH or a full-fledged computer algebra system will want to learn more about this fascinating subject or will want to try a more powerful system. Accordingly, here is a brief guide to the relevant professional society, the literature, and some widely available systems.

4.1 The Professional Societies

The **A**ssociation for **C**omputing **M**achinery **S**pecial **I**nterest **G**roup on **S**ymbolic and **A**lgebraic **M**anipulation is the major international professional society for computer algebra. Their **SIGSAM Bulletin** is the most concentrated and up-to-date source for abstracts and working papers together with announcements of meetings and systems. For information about joining, including special student rates, write the ACM at 1133 Avenue of the Americas, New York, NY 10036.

Some European groups devoted to computer algebra are:

1. SAM-AFCET: Contact M. Bergman, Faculte des Sciences de Luminy, Case 901, 13009; or contact J. Calmet, Universitat Karlsruhe, Institut for Informatik I, 75 Karlsruhe 1, Postfact 6380, West Germany.

2. NIGSAM: Contact Y. Sundblad, Department of Numerical Analysis and Computer Science, KTH S-10044 Stockholm, Sweden.

3. SEAS/SMC: Contact J. A. van Hulzen, Twente University of Technology, P.O. Box 217, 7500 AE enschede, The Netherlands.

4.2 The Literature

Regrettably, there is not yet a textbook devoted to computer algebra, and the few textbooks that contain relevant material are rather advanced. Most of the information is sparsely scattered in research journals or less accessible conference proceedings and reports. However, the following relatively accessible references contain surveys, bibliographies, and collections of articles that should serve as a good point of departure for exploring most facets of the literature:

1. ACM SIGSAM Bulletin, ACM, New York, all issues.

2. Communications of the ACM 14, No. 10, August 1971.

3. SIAM Journal on Computing 8, No. 3, August 1979.

4. Communications of the ACM 9, No. 10, August 1966.

5. Journal of the ACM 18, No. 4, October 1971.

6. P.S. Wang, editor, **Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation,** ACM Order No. 505810, P.O. Box 64145, Baltimore, MD 21264, $23.

7. E.W. Ng, editor, **Symbolic and Algebraic Computation,** Lecture Notes in Computer Science, 72, Springer-Verlag, New York, 1979.

8. R.D. Jenks, editor, **Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation,** ACM, New York, 1976.

9. V.E. Lewis, editor, **Proceedings of the 1979 MACSYMA User's Conference,** M.I.T. Laboratory for Computer Science, 545 Technology Square, Cambridge, Massachusetts, 1979.

10. C.M. Anderson, editor, **Proceedings of the 1977 MACSYMA User's Conference,** NASA CP-2012, 1977.

11. Knuth, D.E., **The Art of Computer Programming, Volume II, Seminumerical Algorithms,** Addison-Wesley, Reading, Mass., 1980.

12. Stoutemyer, D.R. and Yun, D.Y.Y., "Symbolic Mathematical Computation", **Encyclopedia of Computer Science and Technology,** J. Belzer, A.G. Holzman and A. Kent, editors, M. Dekker, New York, Supplementary Volume 15, pp. 235-310.

4.3 Widely Available Systems

Despite an almost total lack of publicity, computer algebra has been available for large mainframe computers since 1951 when the first symbolic differentiation program was written. Since then there have been numerous major general-purpose computer algebra systems implemented. In contrast to the PICOMATH demonstration package:

1. They require up to 400 times as much memory space.

2. They accommodate a significantly larger class of expressions, including equations, multivariate and matrix or tensor expressions.

3. They provide their own indefinite-precision arithmetic to avoid the serious limitations of finite-precision arithmetic.

4. They accommodate much larger expressions having hundreds or even thousands of terms, with coefficients having hundreds of digits.

5. They provide a larger suite of built-in and optional transformations, including factoring and partial-fraction expansions for example.

6. They permit the user to save symbolic results as values of variables for use in subsequent expressions, thus helping the user to perform a sequence of related operations.

7. They provide convenient facilities permitting the user to write function definitions and simplification rules to extend the built-in capabilities by enlarging the allowable class of expressions or the variety of available transformations.

Moreover, some of these systems are more interactive than PICOMATH, permitting an exploratory dialogue wherein the user enters a sequence of expressions, assignments, function definitions and simplification rules at a terminal, viewing each corresponding result before deciding how to proceed. Each result, function definition or simplification rule is available for immediate use in each subsequent expression. Interaction is less crucial for very large well-defined problems, but interaction is highly desirable for "one of a kind" problems or moderate-sized problems that could with some effort be done manually. Interactive systems are also far more motivating for educational purposes, where the problems tend to be small, numerous and varied.

In approximate order of increasing memory requirements, here are some of the most widely available general-purpose systems that are currently supported:

1. **muMATH-79**^{tm} is an interactive system that runs on microcomputers based on the 6502, 8080, 8085, or Z80 microprocessors, provided they have enough memory and an appropriate disk operating system. These include CP/M with at least 32 kilobytes of RAM memory, the Radio Shack TRS-DOS with at least 32 kilobytes of such memory, or the Apple computer with at least 48 kilobytes of such memory. muMATH is distributed to end users, computer stores and hardware manufacturers by Microsoft at 10800 N.E. Eighth, Suite 819, Bellevue Washington 98004, and muMATH is also distributed by the authors: The Soft Warehouse, at Box 11174, Honolulu, Hawaii 96828. The less common CP/M disk formats are available from Lifeboat Associates, 1651 Third Avenue, New York, N.Y. 10028.

2. **SAC-2** is a non-interactive system which runs on any computer that can directly run a 1966 standard FORTRAN program of at least about 120 kilobytes. Information about SAC-2 is available from Professor George Collins, Computer Sciences Department, University of Wisconsin, 1210 West Dayton St., Madison, Wisconsin 53706.

3. **FORMAC** runs on any IBM 360 or 370 that can accommodate a PL/I program of at least about 150 kilobytes. FORMAC is semi-interactive on some operating systems. Information about FORMAC is available from Knut Bahr at GMD/IFV, D-6100, Darmstadt, Germany.

4. **ALTRAN** is a non-interactive system which runs on any computer that can directly run a 1966 standard FORTRAN program of at least about 270 kilobytes. Information about ALTRAN is available from the Computing Information Library, Bell Laboratories, 600 Mountain Avenue, Murray Hill, N.J. 07974.

5. **REDUCE** is an interactive system that runs on the IBM 360 or 370, DEC 10 or 20, Univac 1100 series, Control Data Cyber series, Burroughs 6700, and several other computers, requiring a minimum of about 350 kilobytes. For information about REDUCE, write Anthony Hearn, Rand Corporation, 1700 Main Street, Santa Monica, California 90401.

Additional systems are announced in back issues of the ACM SIGSAM Bulletin.

5. COMPUTER ALGEBRA IN EDUCATION

It should be clear to anyone who has experienced a general-purpose computer-algebra that it has enormous potential for use in education as well as research. Not only can computer algebra make computing more attractive to mathematically inclined students; computer algebra can make mathematics more attractive to computer enthusiasts. It provides a great opportunity for mutual reinforcement and cross motivation between math and computer education.

Personal computers are becoming so prevalent that students, engineers, scientists, and mathematicians will soon be using computer algebra extensively. It should not be more than a year or two before general-purpose computer algebra is available on pocket calculators, because:

1. Several manufacturers now make low wattage "CMOS" versions of most popular 8-bit microprocessors, and CMOS versions of some 16-bit microprocessors are currently under development.
2. There are already hand-held terminals with 32 kilobytes of low wattage memory.
3. There are already hand-held calculators with a sufficiently large low watttage liquid-crystals to display a reasonably large mathematical expression -- perhaps one term at a time.

Eventually some enterprising manufacturer will surely merge these three technologies, producing a hand-held calculator capable of running a full-fledged computer-algebra system such as muMATH. Thus, it behooves every math and computer science educator to explore how this revolutionary tool can be used to aid education.

It is undeniably true that most students are far more intrigued and motivated by the artificial intelligence and game playing applications of computers than by the accounting and numerical applications that currently account for most computer usage. Thus, it is advisable to exploit this strong preferential interest to help teach both mathematics and computer science. If more good math, science, and engineering students are attracted to computers and more good computer-oriented students are attracted to math, then more students will ultimately learn to use computers effectively for both numeric and nonnumeric purposes.

Computer algebra makes a highly motivating introductory computer programming course for math, science and engineering students. Computer algebra is also an ideal principal language for such students, because numbers and arithmetic comprise appreciably less than half of the kindergarden through calculus math curriculum. Moreover, the limited-precision integer and floating-point arithmetic typical of traditional programming languages is not the kind of arithmetic taught in this curriculum or used in everyday life.

Some educators may fear that computer algebra might cause algebraic skills to atrophy or prevent them from developing in the first place. Similar concerns were undoubtedly expressed about Arabic numerals, multiplication tables, logarithms, Laplace transforms and pocket numerical calculators; but we have survived their convenience. The National Council for Teachers of Mathematics strongly supports the use of numerical pocket calculators in classrooms, and every reason for this support is even more true of computer algebra.

Automatic symbolic mathematics makes it possible for students to concentrate on basic mathematical concepts rather than spending an inordinate amount of time mechanically performing transformations. Computer algebra lets students explore such fundamental concepts as commutativity, associativity, groups and rings. Moreover, the extensive algebraic capabilities of computer algebra enables students to investigate larger examples than is otherwise practical. Patterns thus revealed may suggest useful theorems. Conjectured patterns thus violated provide counterexamples against false hypotheses. Thus, computer algebra can contribute to teaching **mathematical discovery.**

Existing computer-algebra systems can also make other educational contributions:

1. Trace packages can be used to let students witness each step of an algebraic simplification, rather than merely the final result.

2. The very fact that algebra and calculus can be automated should encourage average and poor math students that the flashes of inspiration characteristic of quick students are unnecessary for those operations -- there is revealed hope for the slower more methodical students.

3. For students who know how to program in the language in which the computer algebra packages are written, inspection of the underlying algorithms can help them learn methods for accomplishing the operations. Moreover, by programming extensions to the built-in facilities, students can reinforce understanding of the built-in and extended operations.

The above are ways that existing computer algebra systems can be used right now. However, there is a potential for much more. In conjunction with a computer-aided instruction package, existing computer algebra systems could be used for extremely flexible and intelligent algebra drill, testing, and tutorial dialogue.

None of the existing computer algebra systems is by itself a computer-aided math instruction system. Thus, someone experienced in computer-aided instruction could employ PICOMATH as part of a set of interactive math lessons or examinations.

6. HOW PICOMATH WORKS

Reading this section is completely unnecessary for usage or routine installation of PICOMATH. This explanation of how PICOMATH works is included primarily to satisfy the curiousity of those who are interested. The design goal of extreme compactness necessitated reliance upon built-in approximate floating-point arithmetic and use of indirect synthetic techniques that are quite different from those generally employed manually or by more sophisticated large symbolic math systems. Consequently, a full appreciation of this explanation requires substantial experience in both computer science and numerical analysis. Thus, read on if you wish to pick up whatever you can, but don't despair about points that presume more experience or training than you possess.

PICOMATH generally uses the technique of **evaluating** the given user-supplied formula at several numerical values of its independent variables, then **interpolating** these sample values by an expanded **polynomial, rational expression, trigonometric expression,** or **Fourier expression.**

Interpolation is most often used to estimate numerical values of a physical quantity between experimentally-measured samples, or to estimate numerical values of an irrational function between tabulated approximate values: After determining the coefficients in the **interpolant,** it is evaluated at the desired intermediate numerical values of its variables in order to produce a **numerical** result. In contrast, after determining these coefficients PICOMATH displays them interspersed with literal character-string constants such as "X^" or "SIN(", in order to display a **symbolic** representation of the interpolant **expression.** Provided the given expression is mathematically equivalent to one in the employed class of interpolants, and provided the degree or multiplicity of the interpolant is sufficient, the latter is mathematically equivalent to the given unexpanded expression, to within cumulative roundoff error. In the polynomial, trigonometric and Fourier case, it is trivial to derive the coefficients of the derivative or an antiderivative of the interpolant from the coefficients of the interpolant, so these options are provided there. The following subsections give a more specific explanation for each program:

6.1 The Polynomial Program

For multivariate polynomial interpolation using the same general form of interpolant repeatedly, the most straightforward technique is to use a **preconditioned** formula derived using the **method of undetermined coefficients.** Since this general method is also used in the trigonometric program, we describe it here in some generality: Let $\mathbf{w}$ denote the vector of independent variables, such as (x, y, z) in our polynomial program. We wish to interpolate an expression $s = f(\mathbf{w})$ by a linear combination of n given independent **basis** functions:

$$s = a_1 f_1(\mathbf{w}) + a_2 f_2(\mathbf{w}) + \ldots + a_n f_n(\mathbf{w}),$$

based on samples

$$s_1 = f(\mathbf{w}_1),\ s_2 = f(\mathbf{w}_2),\ \ldots,\ s_n = f(\mathbf{w}_n).$$

This implies that

$$a_1\ f_1(\mathbf{w}_1) + a_2\ f_2(\mathbf{w}_1) + \ldots + a_n\ f_n(\mathbf{w}_1) = s_1,$$
$$a_1\ f_1(\mathbf{w}_2) + a_2\ f_2(\mathbf{w}_2) + \ldots + a_n\ f_n(\mathbf{w}_2) = s_2,$$
$$\ldots$$
$$a_1\ f_1(\mathbf{w}_n) + a_2\ f_2(\mathbf{w}_n) + \ldots + a_n\ f_n(\mathbf{w}_n) = s_n.$$

These are n simultaneous linear algebraic equations for n unknowns a_1, a_2, ..., a_n. Consequently, in matrix notation,

$$\mathbf{F}\ \mathbf{a} = \mathbf{s}.$$

Thus, to determine the interpolant coefficients, we merely have to solve these equations.

In order to display results fully expanded, the polynomial program uses a basis of **monomials** such as x^6, x^5, x^4, x^3y, etc. Given this basis, we would like to choose sample points that minimize the expectations of overflow, underflow, or severe roundoff error. No fixed set of sample points can be adequate for all possible polynomial expressions on line 20 or the program, but we would like a set that is good for the largest percentage of examples that people are likely to try with this program. Here are some considerations that can help us choose a good set of points:

People tend to have a strong preference for formulating mathematical models scaled such that the expressions primarily or entirely entail integer coefficients of small magnitude -- with a bias toward positive coefficients, especially the coefficient 1. It is especially easy for us to grasp the significance of the integers 0 through 100, so we tend to choose origins and units that produce such results. For example, this explains the popularity of percentages over the equivalent less arbitrary decimal fractions between 0 and 1.

For similar reasons, if the **zeros** of a polynomial have physical significance or are of mathematical interest, which is often the case, then there is a tendency for people to formulate the mathematical model in such a way that the zeros occur at x=0 or at points x having absolute values that are neither extremely small nor large compared to 1.

For polynomial interpolation, the expectation of excessive roundoff error is minimized if the sample abscissas **interlace** the zeros of the polynomial: In the **univariate** case if the zeros are all simple and real, a good set of sample abscissas is roughly midway between each pair of adjacent zeros and also outside each **end** zero by an amount roughly equal to the distance between each end zero and its adjacent interior sample abscissa. Multiple, complex, and multivariate zeros complicate the issue, but the same general principle applies -- we would like the sample points scattered between the zeros more or less proportionally to the density of the zeros, taking into account multiplicities. However, we do not know the locations of these zeros at the beginning, and we do not want to waste time and increase the risk of overflow or underflow in an attempt to isolate them. Thus, in view of the widespread predilection for small-magnitude integers, a reasonable strategy is to use the origin and uniformly spaced points clustered about the origin at distances neither quite small nor quite large compared to 1. Points having small integer components have the additional advantage that if the coefficients of the given unexpanded polynomial are also integer and if for integer exponents the "^" operator is computed using repeated multiplication rather than logarithms and exponentials, then no roundoff errors are incurred when executing line 20 of the program until the magnitude of intermediate results exceed the largest integer that is exactly representable in the underlying floating-point arithmetic.

The matrix **F** is the same every time the program is run. Consequently, this is one of those few instances where it is worth **inverting** once and for all so that the program need merely multiply the vector **s** by the precomputed matrix $\mathbf{F}^{-1}$ in order to determine the interpolant coefficients **a**. Moreover, to minimize roundoff error it is desirable and worthwhile to compute $\mathbf{F}^{-1}$ exactly, which may be done using more sophisticated computer algebra programs such as those listed in section 4.3, because the sample points all have small-magnitude integer components. (This is another reason for choosing such sample points.) In our polynomial case this exact inverse turns out to have relatively simple fractions as entries, with a large portion of entries that are zero. Consequently, to save space and time the sample values **s** are stored as individual scalar variables, and the elements of $\mathbf{F}^{-1}$ are not stored in an array. Instead, the coefficients are computed directly in terms of the sample values in a manner that exploits factoring as much as possible to further reduce roundoff error. Also, the computed coefficients are subjected to a smoothing process: If a coefficient is within 0.01 of an integer, then the program rounds the coefficient to that integer in an attempt to reduce the annoying effects of roundoff error discussed in section 2.1.

Here is a brief outline of the program and a dictionary of its variables:

Outline of the Polynomial Program

Lines	Purpose
5-10	Declare double precision and integer variables, then skip around a subroutine and the prefatory remarks
20-30	Subroutine assigning a polynomial in X, Y and Z to A
40-180	Prefatory remarks
190	Initializations, including array dimensioning
200-250	Query user for choice between Expand, Derivative or Integral, and display left side of result equation
260-320	Collect enough sample values to determine the coefficients of x^6, x^5, and x^4
330-350	Determine coefficients of x^6, x^5, x^4, & display nonzero terms
360-450	Collect more samples to determine more coefficients
460-530	Determine additional coefficients & display nonzero terms
540-550	Collect 2 more sample points, recycling variables E, D and B
560-600	Determine additional coefficients & display nonzero terms
610	Collect two final sample points, recycling variable C
620-660	Determine remaining coefficients, displaying nonzero terms
670	Display a zero if no terms were displayed before
680	Terminate display line
690-750	Compare given and derived expressions at 4 transcendental points, then display a warning if any discrepancy is not relatively small considering Accuracy("^")
760-790	Display a message explaining how to proceed to next example
800	Stop
810-960	Subroutine for saving smoothed coefficient and displaying corresponding term if nonzero
970-980	Subroutine for smoothing a coefficient
990-1040	Subroutine comparing given & resulting expressions at sample points, updating maximum discrepancy & expression magnitudes
1050	End

Str0 input expression
Str9 result

Dictionary of Variables in Polynomial Program

Names	Purposes
A	Latest sample ordinate
L6 array A	Interpolant coefficients
(θ: 1247 ~~Str8~~ A$	Response to Expand, Derivative or Integral query
B,C	Sum and Difference of 2 opposing ordinates
Str8 B$	Cofactor of the numerical coefficient and power of X
D,E,H,I,J,K, L,M,P,Q,R,S,T	Saved sample ordinates
F	Saved sample ordinate or maximum ordinate magnitude
G	Saved sample ordinate or maximum discrepancy
N	Dummy subscript
O	Output Flag: Has a term already been displayed?
U	Candidate smoothed coefficient
V	Multiplier for coefficient when derivative or integral
W	Multipurpose temporary
X, Y, Z	Independent variables

6.2 The Trigonometric Program

The trigonometric program uses the same general technique as the polynomial program, except that it employs a different set of sample points and a different set of basis functions. The six fundamental trigonometric functions have numerous interdependencies, so a certain amount of judgement, care and experimentation is required to select a **linearly independent** basis that spans the most useful class of composite trigonometric functions.

Because of nontrivial denominators allowed in line 20 and because of the tangents, cotangents, secants and cosecants among its basis, the trigonometric program entails the hazard of sampling at or near a **pole.** Thus, we must avoid sampling at the particularly likely poles $x = 0$, $\pm\pi/2$ or π. On the other hand, other multiples of $\pi/6$ or $\pi/4$ lead to especially simple entries in matrix **F**, making it possible to invert **F** exactly. These are the reasons behind our particular choice of sample points. However, it is still possible for a sample point to coincide with a pole of the expression in line 20 even though the corresponding simplified expression is in the representable class. For example, evaluation of

$$\frac{1 - \tan^2 x}{1 - \cot^2 x}$$

at $x=\pm\pi/4$ may cause a zerodivide or overflow even though the expression simplifies to $1 - \sec^2 x$, which is in the representable class.

Here are a brief outline and dictionary of variables in the program:

Outline of the Trigonometric Program

Lines	Purpose
5-10	Declare double precision and integer variables, then skip around a subroutine and the prefatory remarks
20-30	Subroutine assigning a trig expression in X and Y to A
40-180	Prefatory remarks
190	Initializations, including array dimensioning
200-220	Definitions of secant, cosecant and cotangent functions
230-250	Query user for choice between Expand, Derivative or Integral
260-290	Display answer announcement and left side of result equation
300-470	Collect sample values
480	Recycle X, Y and P as auxiliary constants
490-960	Determine coefficients, displaying nonzero terms
970	Display a zero if no terms were displayed before
980-1010	Compare given and interpolated expressions at 4 more points
1020-1050	Display a warning if any discrepancy is relatively large considering Accuracy(ATN).
1060-1110	Display a message explaining how to proceed to another example
1120	Stop
1130-1200	Subroutine for smoothing and saving a coefficient, then displaying the corresponding term if it is nonzero
1210-1270	Subroutine for comparing given & interpolated expressions at a point, updating maximum discrepancy & expression magnitudes
1280	End

Dictionary of Variables in Trigonometric Program

Names	Purposes
A	Latest ordinate sample
array A	Interpolant coefficients
A$	Response to query about Expand, Integral or Derivative

B,C;D,E;F,G;
H,I;J,K;L,M;
Q,R;S,T;W,Z
	Sums and Differences of opposing ordinates
	F, G and H are also used in discrepancy testing

B$	Cofactor of numerical coefficient
N	Dummy subscript
O	Output flag: Has a term already been displayed?
P	PI/4 or 2 * 2^(1/2) - 2
PI	Ratio of circumference to diameter of a circle
U	Candidate smoothed coefficient
V	Multiplier of coefficient for integral or derivative
X	Independent variable or 2^(1/2)
Y	Independent variable or 3^(1/2) / 2

6.3 The Fourier Program

The Fourier program uses an interpolant of the form

$$s = u_0 + v_1 \sin x + u_1 \cos x + v_2 \sin(2x) + u_2 \cos(2x) + \ldots + u_m \cos(mx)$$

together with sample points $x = 0, \pi/n, 2\pi/n, \ldots, (n-1)\pi/n$, where $n=2m+1$. This form of expression is widely used in science and engineering for portraying the **spectrum** or **harmonics** of a periodic function. Being canonical and **linear** in the sines and cosines, it is a particularly convenient form to use for many purposes. Moreover, for such equally-spaced cyclic sample points, there is a known closed-form expression for each element of $\mathbf{F}^{-1}$ defined in section 6.1. This leads to the general closed-form solution

$$u_0 = \sum_{j=1}^{n} u_j / n,$$

$$u_k = 2 \sum_{j=1}^{n} s_j \cos(2jk\pi/n) / n,$$

$$v_k = 2 \sum_{j=1}^{n} s_j \sin(2jk\pi/n) / n,$$

for $k=1,2,\ldots,m$, where the s_j are the sample ordinates. Arrays **u** and **v** together comprise the **Discrete Fourier Transform** of array **s**. Because of the periodicity of the sine and cosine functions in conjunction with the uniformly-spaced cyclic sample points, there are **Fast Fourier Transform** methods that are faster for large n than using straightforward summation to evaluate the above formulas. However, these more complicated methods are not worth the required extra space in this program for adjustable n as small as 8. Nonetheless, the program does exploit the periodicity to reduce roundoff error by exactly reducing jk **modulo** m before computing the sine and cosine of $2jk\pi/n$ in the above formulas. For efficiency these sines and cosines are precomputed and saved as arrays C and S. Also for efficiency, this precomputation is done using the **recurrence** relations

$$\cos(k\,x) = 2 \cos x \cos[(k-1)x] - \cos[(k-2)x],$$
$$\sin(k\,x) = 2 \cos x \sin[(k-1)x] - \sin[(k-2)x],$$

for $k = 2, 3, \ldots, n-1$.

Here is an outline of the program and a dictionary of its variables:

Outline of the Fourier Program

Lines	Purposes
5-10	Declare double precision and integer variables, then skip around a subroutine and prefatory remarks
20-30	Subroutine assigning a (usually trig) expression in X to A
40-180	Prefatory remarks
190	Initializations
200	Automatically determine Accuracy(ATN)
210	More initializations, including array dimensioning
220-230	Query user for choice between Expand, Derivative or Integral
240-250	More initializations, then terminate line
260-280	Use recurrences to get sines & cosines of sample abscissas
290-320	Determine sample ordinates, while accummulating their total
330	Determine & smooth non-trig term
340-380	Announce answer & display left side of result equation
390-400	Display non-trig term if nonzero
440-580	Use Discrete Fourier Transform while displaying nonzero terms
590	Display 0 if no terms were displayed before
600-680	Determine largest discrepancy and ordinate magnitude for some extra abscissas
690-730	Display a warning if discrepancy is not relatively small, considering Accuracy(ATN)
740-800	Display a message explaining how to proceed to next example
810	Stop
820-940	Subroutine for printing a trig term
950-970	Subroutine for smoothing a coefficient
980	End

Dictionary of Variables in Fourier Program

Names	Purposes
A	Most recent ordinate sample
array A	Ordinate samples
A$	Response to Expand, Derivative or Integral query
B$	Leading portion of cofactor of numerical coefficient
array C	Cosines of the sample abscissas
F	PI/4 or Maximum ordinate magnitude
G	Maximum discrepancy magnitude
H	Nontrig coefficient
J, K	Dummy subscripts
M	Maximum allowable angle multiplicity
N	Number of samples
O	Output flag: Has a term already been displayed?
PI	Ratio of circumference to diameter of a circle
Q	Accuracy(ATN) ^ (1/2)
array S	Sines of the sample abscissas
T	Temporary
array U	Cosine coefficients
array V	Sine coefficients
W	$2 \cos (2\pi/n)$
X	Independent variable
Y	Candidate rounded coefficient
Z	Temporary

6.4 The Rational Program

The method of undetermined coefficients is not as attractive for rational interpolation because having some of the undetermined coefficients in the denominator of the interpolant makes the resulting equations nonlinear:

$$s_k = \frac{u_m x_k^{\,m} + u_{m-1} x_k^{\,m-1} + \ldots + u_0}{v_n x_k^{\,n} + v_{n-1} x_k^{\,n-1} + \ldots + v_0}, \qquad \text{for } k=1,\ 2,\ \ldots,\ n+m+2.$$

The good news is that these equations can be linearized by multiplying each equation by the denominator of its right side. The bad news is that the resulting linear equations are **singular.** However, more good news is that the equations are also **homogeneous** so that we are blessed with not merely one but a whole infinite family of solutions. The reason for the extra degree of freedom is that given any set of coefficients that satisfy the above equations, then by cancellation any nonzero common multiple of these coefficients also satisfies the equations. Thus, we are free to impose an extra **normalization** condition such as insisting that the leading nonzero coefficient of the denominator be 1. Regrettably, we do not know ahead of time which one is the leading nonzero coefficient. We can of course first try setting v_n to 1, then setting v_{n-1} to 1, etc., until one of these choices enables the equations to be solved. However, this means that we cannot invert the coefficient matrix ahead of time in order to speed the rational program. Moreover, in the arena of inexact arithmetic, roundoff errors often turn values that should be zero into nonzero values. Thus the program would have to incorporate judgement about whether or not to regard a candidate leading coefficient as negligible. The resulting program size and execution time would be inconsistent with the other PICOMATH programs.

For the above reasons the program employs an utterly different technique using **inverted differences.** F.B. Hildebrand describes the method on pages 494 to 502 of the second edition of his book titled **Introduction to Numerical Analysis,** which is published by McGraw-Hill. Therefore, only a brief example is presented here to suggest the overall idea. Those who are familiar with the method of divided differences for polynomial interpolation will notice great similarity.

Suppose that the right side of the assignment on line 20 of the program is

$$2 + \frac{8\ x}{4\ x^2 - 1} + \frac{4}{4\ x^2 - 1}$$

and that our sample abscissas are $x = -2, -1, 0, 1, 2$. The first step is to form the following **inverted difference table:**

i	x_i	1	2	3
1	-2	1.2		
2	-1	0.667	-1.875	
3	0	-2	-0.625	0.8
4	1	6	0.625	0.8
5	2	5.333	1.875	0.8

The column labeled "1" is simply the sample ordinates. The columns to its right are computed as follows: Let $d_{i,j}$ denote the entry in row i, column j. Then

$$d_{i,j} = \frac{x_i - x_j}{d_{i,j-1} - d_{j-1,j-1}}, \qquad \text{for } j>1 \text{ and } i>=j.$$

Readers who seriously want to understand the method should take the time here to verify the entries manually according to this formula.

If the expression on line 20 of the program is equivalent to a rational expression, then to within cummulative roundoff error we must eventually arrive at a column of identical entries. If this happens at column t, then the **continued fraction** form of the rational interpolant is

$$s = d_{1,1} + \cfrac{x - x_1}{d_{2,2} + \cfrac{x - x_2}{\cfrac{x - x_3}{\ddots \; d_{t-1,t-1} + \cfrac{x - x_t}{d_{t,t}}}}}$$

Thus for our example we have

$$s = 1.2 + \cfrac{x - (-2)}{-1.875 + \cfrac{x - (-1)}{0.8}}$$

In order to express the interpolant as a ratio of two expanded polynomials, the program places successive terms over a common denominator, beginning with the last two terms at the bottom right of the expression. The process can be summarized by the following recurrence relation for a sequence of polynomials P_k:

$P_{t+1} = 1;$
$P_t = d_{t,t};$

$P_k = d_{k,k}\, P_{k+1} + (x-x_k)\, P_{k+2},$ for $k=t-1, t-2, \ldots, 1.$

Then P_1/P_2 is the desired ratio. We encourage you to verify that this gives $(2\ x + 1)\ /\ (x -0.5)$ for our example. Note how the greatest common divisor is automatically canceled and how the leading nonzero coefficient of either the numerator or denominator is automatically normalized to 1, according to whether t is odd or even respectively. If a displayed answer does not have the proper normalization, then it is because the unit leading coefficient was replaced by zero for being negligible compared to other coefficients in the same polynomial.

To save space, only the rightmost inverted difference in each row is retained at any time, thus permitting a singly-subscripted rather than double-subscripted array. Similarly, only the two most recently computed polynomials P_k are retained, with polynomial P_k overlaying polynomial P_{k+2}. To save additional space, the abscissas, ordinates, inverted differences and polynomials P_k all overlay each other in the same shared array named A.

Roundoff errors generally destroy exact equality in the column that should have identical entries. Thus what we want to detect is equality to "within cumulative roundoff error" in order to decide where to stop in order to avoid excessive roundoff error, overflow or division by zero when attempting to compute the next column. Also, to minimize the expectation of having a sample abscissa near or on a zero of any denominator in program line 20, the program begins with the **transcendental** abscissa ln 2, then takes successive abscissas 1 less, then 1 more, then 2 less, then 2 more, etc. Moreover, the program may skip any of these candidate abscissas if an estimated bound on the cumulative roundoff error in an inverted difference suggests that the inverted difference might be mostly roundoff noise. The program stops computing additional inverted differences when 3 successive candidate abscissas have been rejected for this reason.

The error bound estimates for the inverted differences in array A are stored in corresponding elements of array E. These estimates are computed using **perturbed** abscissas to estimate bounds on the roundoff errors in the ordinates, then using linearized rules bounding propogation of roundoff in subsequent operations.

Although limited to one variable and lacking differentiation or integration, the less straightforward interpolation technique together with the array overlays and the self-contained error analysis make the logic significantly more complicated than for the other PICOMATH programs. Subject to this warning, here is a brief outline and dictionary of variables for the rational program:

Outline of the Rational Program

Lines	Purpose
5-10	Declare double precision and integer variables, then skip around a subroutine and the prefatory remarks
20-30	Subroutine assigning a rational expression in X to A
40-180	Prefatory remarks
190	Initialize some variables
200-220	Automatically determine implementation-dependent accuracy tolerances and corresponding appropriate dimensions
230	Display first line of answer announcement
240-360	Derive up to N-1 successive inverted differences and their error estimates until 3 successive candidates are negligible or would lead to unacceptable accuracy
370	Display rest of answer announcement
380-510	Derive coefficients of numerator and denominator from the sample abscissas and the inverted differences
520-570	Display the numerator, a horizontal bar, then the denominator
580-680	Determine largest discrepancy and ordinate magnitudes for abscissas midway between trial interpolation abscissas
690-720	Display a warning if the discrepancy is not small relative to the largest ordinate magnitude
730-760	Display a message explaining how to proceed to next example
770	Stop
780-1040	Subroutine for smoothing then displaying a polynomial, with line 790 as a secondary entry point
1050	End

Dictionary of Variables in the Rational Program

Names	Purpose
A	Ordinate sample
array A	Ordinate, inverted differences & abscissas; coefficients
C	Maximum discrepancy
D	Implementation accuracy of "^"
array E	Estimated bounds on absolute errors in array A
F	Maximum coefficient magnitude
G,I,J,K	Dummy subscripts
H	A(H) = coefficient of Highest degree term
L	A(L) = coefficient of Lowest degree term
M	Index so that final inverted difference has least error
N	Maximum # of terms, including numerator & denominator
O	Output flag: Has a term already been displayed?
R,U,W,Y	Multi-purpose temporaries
S	Signed abscissa increment
T	2.0/3.0, to machine accuracy
X	Perturbed abscissa or a temporary
Z	Nominal abscissa

7. INSTALLATION AND ADAPTATION GUIDE

Section 8 contains the PICOMATH program listings, written in BASIC. The specific dialect of BASIC is ANSI MINIMAL BASIC enhanced by a few widely used extensions that greatly increase readability, compactness or accuracy. This section 7 explains how to adapt those generalized listings to specific implementations of BASIC. Given a moderate familiarity with BASIC, the explanation should also enable translation of the programs to another programming language with which the reader is proficient.

The programs make no use of sophisticated string operations, matrices, or other advanced features. Only singly-subscripted arrays the common floating-point arithmetic operations, and the ability to display literal string constants are required.

7.1 Precision

The first line of all four programs declares that variable names beginning with certain letters are double precision, whereas variable names beginning with certain other letters are integer. Double precision increases the allowable problem size beyond which accuracy becomes intolerable, whereas using integer variables for subscripts and flags saves space and increases speed slightly. However, the program does not require double precision or integer variables, so delete the double precision or integer declarations if the implementation does not support them. Moreover, many BASIC implementations that offer double precision do so for the operators "+", "-", "*", and "/", but not for the operator "^", or for the built-in trig functions, or for user-defined functions. On such implementations double precision usually yields only modest improvement over single precision in instances where program line 20 or any other line employs any of the single precision operations or functions.

In contrast to such declarations or something like them, some implementations provide a command (named perhaps SIZES) that establishes the precisions of all floating-point and of all integer variables. If so, try the greatest allowable precision for floating point, decreasing it only if this yields insufficient space or unacceptable computing time. On the other hand, magnitudes of integer variables never exceed 99, so 2 decimal digits or 1 byte is sufficient for them. Note that single precision is always adequate for variables not mentioned in the declarations, but greater precision merely increases storage and execution time slightly.

Rather than declarations, some implementations require a suffix such as # on all double precision variables, and/or a suffix such as % on all integer variables. If so, delete the declarations and modify the corresponding variable names throughout the programs. It also may be necessary to identify integer or double precision constants or functions by some such technique in program line 20 and throughout the program in order to avoid degradation of some intermediate results to single precision.

7.2 Assignments

Some versions of BASIC require each assignment statement to begin with the word LET. If necessary modify all assignments accordingly. In contrast, some implementations permit multiple assignments using a construct such as

$$\text{variable}_1 = \text{variable}_2 = \ldots = \text{variable}_n = \text{expression}$$

or

$$\text{variable}_1,\ \text{variable}_2,\ \ldots,\ \text{variable}_n = \text{expression}.$$

If so, there are several obvious opportunities to exploit this feature.

7.3 Printed Messages and Remarks

Although most terminals that display only upper case will automatically convert to upper case for display, some implementations do not permit lower-case letters in printed messages or remarks. If necessary, use upper-case letters only. Also, readability and convenience is aided by breaking printed messages so that they fit within the width and height of the smallest terminal display area that is customarily used. Similarly, readability of the program is aided by breaking lines within the width of that terminal. Moreover, to prevent relevant output from scrolling off the screen prematurely, it may be advisable in some places to use a PRINT and INPUT statement requesting the user to enter anything in order to continue.

Some BASIC implementation use a comma or something else other than a semicolon to suppress tabs followed by a new line in PRINT statements. If necessary, make appropriate changes.

BASIC implementations that do not provide for displaying an arbitrary INPUT **prompt** string require the INPUT statements in the programs to be subdivided into a PRINT statement then an INPUT statement.

Some versions of BASIC have a single-character **raw input** statement (named perhaps GET) that avoids the necessity of pressing an ENTER or RETURN key after typing an E, D, or I in response to the program query, thus improving interactivity.

Some versions of BASIC permit the LIST command within a program, enabling the program to automatically list line 20 for the user before or after displaying the expanded result.

The three question marks in the remark on line 80 of each program should be replaced by a phrase identifying the computer and language dialect, so that anyone copying your implementation will realize that it is not necessarily in the original form and that it may require modifications for other computers or language dialects. For example, an appropriate phrase would be "Applesoft BASIC".

7.4 Array Dimensions and Subscripts

a. Many implementations do not permit adjustable dimensions such as DIM A(N+N+1), E(N). If necessary, temporarily insert a "PRINT N" statement immediately before DIM statements to determine the appropriate constant to use for N, then use that value in the DIM statements, preceeded by a remark such as "REM The dimension of array A must be at least N+N+1 and the dimension of array E must be at least N", in order to forewarn anyone who attempts to transport your implementation to a different computing environment without benefit of this manual.

b. Some implementations do not permit a DIM statement -- perhaps because only one prenamed array, singly-subscripted, is allowed. If necesssary, delete the DIM statements. Moreover, if only one array is allowed, change all array references to the name of the one array, while increasing the subscripts of former references to other arrays by appropriate amounts so as to pack the arrays end-to-end in the one array. For example, in the rational program, change E(K) to A(K+N+N+1) in program line 220.

c. Rather than parentheses, some implementation use square brackets "[" and "]" or some other delimiters to delimit subscripts. If necessary modify all array references accordingly.

d. Some implementations require a separate DIM statement for each array. If necessary, modify the program accordingly.

e. Some implementations do not permit use of a name as both an array and a scalar. If necessary, change scalar A to some other name throughout the programs to avoid conflict with array A.

7.5 Exponentiation

Some implementations use "**" or something else other than "^" to indicate raising to a power. If necessary modify the programs accordingly. Most PASCAL and "C" language implementations do not provide an exponentiation operator. In such cases one should define an exponentiation **function** for use in program line 20.

Some implementations are unable to raise negative numbers to a power even though the power is integer. For such implementations, be sure to display a message warning the user to use repeated multiplication rather than exponentiation, or to use a specially-written exponentiation function defined in each program to overcome this limitation.

7.6 Multiple-statement Lines

a. Some implementations use a separator different than ":" between multiple statements on a line. If necessary modify the program accordingly.

b. Some implementations do not permit multiple statements per line. If necessary enter the statements one per line, with appropriately interpolated line numbers. For a multiple-statement line of the form

IF condition THEN statement$_1$: statement$_2$: ...: statement$_n$

the corresponding equivalent to use is

```
      IF NOT(condition) THEN line
      statement1
      statement2
      ...
      statementn
 line ...
```

Beware that some versions of BASIC, such as Northstar, permit multiple statements per line but treat them differently following THEN. Such IF statements on such implementations must be rearranged as illustrated above.

c. Some implementations permit only a line number as the THEN clause of an IF statement. However, a statement of the form

IF condition THEN statement

can always be transformed to the equivalent form

```
      IF condition THEN line1
      GOTO line2
line1 statement
line2 ...
```

7.7 Relational Operators

a. Some implementations represent the "not equals" relational operator "<>" by "><", "#", "¬=" or "NE". Also, some implementations represent the "less than or equals" operator "<=" by "=<" or "LE", while some implementations represent the "greater than or equals" operator ">=" by "=>" or "GE". If any of these situations prevail, make appropriate substitutions throughout the programs.

b. Some implementations do not provide a full set of relational operators. However, programs can always be rearranged so that the only necessary relational operator is one of the set {<, >, <=, >=}. For example, the form

IF expression$_1$ <= expression$_2$ THEN line$_1$

can be rewritten as

```
      IF expression₁ > expression₂ THEN line₂
      GOTO line₁
line₂ ...
line₁ ...
```

If necessary, modify the programs accordingly.

7.8 Logical Operators

a. Some implementations require use of another symbol such as "+", "|", or "V" for the logical operator named "OR" in the program. Similarly, some implementations require use of another symbol such as "*", "&", or "Λ" for the logical operator named "AND", while some implementations require use of another symbol such as "-", "¬", or "~" for the logical operator named "NOT". If any of these situations prevail, make appropriate substitutions throughout the programs.

b. Some implementations do not support one or more of the logical operators named "AND", "OR", or "NOT". However, a line of the form

```
      IF condition₁ OR condition₂ THEN statement
```

can be replaced by the equivalent pair

```
      IF condition₁ THEN statement
      IF condition₂ THEN statement
```

Similarly, a line of the form

```
      IF condition₁ AND condition₂ THEN statement
```

can be replaced by the equivalent construct

```
      IF condition₁ THEN line₁
      GOTO line₂
 line₁ IF condition₂ THEN statement
 line₂ ...
```

Moreover, a line of the form

```
      IF NOT(condition) THEN statement
```

can be replaced by the equivalent construct

```
      IF condition THEN line
      statement
line  ...
```

7.9 Built-in Functions

Beware that in BASIC function INT(u) is defined as the largest integer that does not exceed u. In many other languages and in a few implementations of BASIC, this is called FLOOR(u), with INT(u) or FIX(u) or TRUNC(u) being reserved for integer **truncation**, which is different for negative arguments. (Truncation always rounds toward zero.) If an appropriate FLOOR-type INT is not available, then instances of INT in the programs can be simulated as follows:

1. Isolate each usage of INT as an assignment of the form

```
y  =  INT(x)
```

2. Replace each such assignment by the pair of statements

```
y = TRUNC(x)
IF x<0 AND y<>x THEN y=y-1
```

Some implementations do not provide a built-in SGN function, which is used in the rational program. However, the argument is never zero there, permitting SGN(Z) to be safely defined there as ABS(Z)/Z.

As in most BASIC implementations, LOG denotes the **natural** logarithm and SQR denotes the square **root.** These may be named differently in some implementations.

Most BASIC implementations do not provide built-in secant, cosecant and cotangent functions, and most BASIC implementations limit user-defined functions to one letter following "FN". Consequently, the corresponding definitions in the trigonometric program may have to be changed to names such as FNE for sEc, FNS for cSc and FNO for cOt.

The MOD operator in the Fourier program is named REM for REMainder in some implementations. Alternatively, MOD or REM is often provided as a function rather than an operator. If no such facility is available, an appropriate function MOD(u,v) can be defined for this program where u and v are nonnegative as u - INT(u/v), with INT being either a FLOOR or TRUNC variant, as described at the beginning of this subsection.

7.10 String Variables and Comparisons

All but the rational program make trivial use of a string variable named A$ to capture the users choice between Expansion, Differentiation, and Integration. Some BASIC implementations do not support string variables or do not permit equality comparisons for them. In such cases, a trick that often works is to initialize the **numeric** variables E, D, and I to values such as 0, 1, and -1 respectively. Then when the user types one of these letters he is actually providing its numerical value, which can be captured with another numeric variable then tested.

(Naturally, in instances where E, D or I are already employed for other purposes, it is necessary to rename variables in order to free E, D and I for this purpose.) If this technique doesn't work, it is always possible to ask the user to directly enter -1, 0 or 1 for integration, expansion and differentiation respectively.

Three of the programs make use of a string variable named B$ to hold all or part of the cofactor to be displayed with each coefficient. However, if necessary the same effect can be accomplished somewhat awkwardly using only string constants by displaying this portion of the cofactor after returning from the coefficient display subroutine.

Some implementations use single rather than double quotes to delimit string constants, necessitating corresponding changes in the programs.

Some implementations require string delimiters around input strings even though they are an entire line. If so, the prompt message for choosing between Expansion, Integration or Differentiation should also include the delimiters.

Some implementations implement strings as arrays of single characters, necessitating corresponding minor changes to the programs.

7.11 Space Requirements

For many microcomputer BASIC implementations that permit multiple statements per line, the Fourier, rational, polynomial and trigonometric programs require about 3400, 3800, 4100 and 5200 bytes of storage respectively, including full PRINT messages, remarks, indentation and spacing. Various implementations require somewhat more or less than these values. Thus some space-saving modifications may be necessary for one or more of the programs on programmable calculators or on the very tiniest computers. These modifications reduce the readability, auto-tutorial level, or robustness of the program, so first try an unabridged adaptation to determine how little, if any, needs to be sacrificed. (Many BASIC implementations provide a command named something such as FREE, MEMORY or PGMSIZE that helps determine space availability.) We may be judged by your adaptation of our software, so we ask your co-operation in changing the program as little as possible, employing each of the following techniques only after the preceeding ones have been fully exploited, yet more savings are needed.

a. For implementations having a built-in MAX function that returns the larger of its two arguments, there are many statements of the form

   IF expression_1 > variable_1 THEN variable_1 = expression_1

   that can be converted to the more compact equivalent:

   variable_1 = MAX (variable_1, expression_1) .

b. Some BASICs don't require an END statement, saving a little space.

c. If the implementation is more nearly a pure **interpreter** than a pure **compiler,** then renumbering the lines from 1 with an increment of 1 may save space. However, this makes later insertions more awkward. Consequently, if your implementation does not provide an automatic renumbering facility, then experiment first using a small segment of the program to determine if there are any such space savings before going to the trouble and hazard of changing our line numbers.

d. If the implementation is more nearly a pure interpreter than a pure compiler, various syntactic abbreviations may save space in the loaded program. For example, the following table indicates some commonly allowed BASIC abbreviations:

Construct	Abbreviation
THEN	"," or " " or "T"
PRINT	"?" or "!" or "P"
REM	"'" or "!"
INT(u/v)	u\v or v\u
NEXT variable	"NEXT" or "N"
NEXT variable$_1$ NEXT variable$_2$	NEXT variable$_1$, variable$_2$

Similarly, reduction of the indentation amounts may save space for some interpreters. Also, we have not exploited the opportunities for multiple statements per line where to do so would obscure the program structure, and many BASIC or FORTRAN implementations even permit all spaces to be omitted, yielding an incredibly ugly program. (Please don't do this unless absolutely necessary.)

e. It is relatively straightforward to delete the integration and/or differentiation facilities from any of the three programs that provide it. For pre-calculus students these facilities are merely a distraction anyway, which teachers may wish to disable with a removable GOTO bypassing the query, accompanied by a remark explaining how to restore the program to full power.

f. The remarks and printed messages can be abbreviated slightly without becoming too terse for comprehension, and the displayed instructions are more appropriate as comments in batch-oriented implementations that do not accept input from a terminal. However, we absolutely insist that at least The Soft Warehouse name, address, copyright notice, and conditional duplication permission be included in each program.

g. At the expense of forcing users to remember various different oddball line numbers in place of "line 20", the initial GOTO can be deleted if the subroutine beginning at line 20 is moved to just before the end of the programs.

After becoming familiar with the programs in their original form, experienced programmers with computers having sufficient memory may wish to merge the polynomial program with the rational program and merge the trigonometric program with the Fourier program, or even to merge all four programs into one. The most straightforward technique is to add say 2000 to all of the line numbers of the second program, etc. then essentially append the programs end to end, with a few obvious deletions such as internal END statements. One can then choose the second program for example by entering RUN 2010 rather than merely RUN. Better yet, the selection can be menu driven. However, there are obvious opportunities to share portions of the programs, perhaps at the expense of renaming some variables. More ambitiously, one can arrange so that the program uses a single RUN command but tries fitting the formula on line 20 with successive candidates until one category results in a sufficiently small discrepancy. However, these are not minor adaptations, and full exploitation of the opportunities for sharing sections of code requires an understanding of section 6. Consequently, we strongly recommend against these adaptations until after the individual programs are successfully installed and much spare time is available.

7.12 Other Languages

PICOMATH can be adapted to most computer programming languages. However, the programs are most pleasant to use in an environment that permits a program to be modified then rerun in less than 1 minute of elapsed time. For most computing environments this suggests languages such as APL, BASIC, FORTH, LISP or muSIMP rather than languages such as C, FORTRAN, PASCAL or PL/I. A **user** of PICOMATH should be relatively indifferent to the choice of implementation language except insofar as it affects interactivity and precision. Thus, since PICOMATH is used many times once implemented, it is appropriate to select from the available implementation languages on the basis of the user's needs more than the implementor's comfort of familiarity.

Some language implementations offer an EXECute statement or its equivalent, which permits a character-string to be executed as if it were a program statement. This facility can be used to make PICOMATH more interactive as follows: Rather than having the expression to be simplified be on program line 20, a character string of the form

"A = expression"

is input interactively from the terminal as the value of a string variable named perhaps C$. Then, each instance of "GOSUB 20" is replaced by "EXEC C$" to give the same effect without requiring any program modification by the user. This enables the STOP statement to be replaced with a GOTO statement causing the program to repeatedly loop back to the beginning for additional input examples until interrupted manually by striking the appropriate **break** key on the terminal. This further permits the use of abbreviated prompt strings after every interaction cycle that does not produce serious discrepancy.

Other languages and some BASIC implementations provide an error trap facility named perhaps ON ERROR or ERRORSET that permits graceful recovery from overflow, underflow and zerodivide. If such a facility is present, perhaps it can be used to proceed from the point of difficulty using the most reasonable value, after printing a more specifically helpful message than would otherwise occur.

7.13 Testing

Besides the examples in this manual, good test cases are the following:

Polynomial: x^6, x^5; x^4, x^3y, x^2y^2, xy^3, y^4; x^3, x^2y, xy^2, y^3; x^2, xy, xz, y^2, yz, z^2; x, y, z, 1, 0, -1.

Rational: x^m, x^{m-1}, ..., x, 1, 0, -1, x^{-1}, x^{-2}, ..., x^{-n}; where m and n are the exponent limits displayed in the epilogue message.

Fourier: -1, 0, 1, cos x, sin x, cos(2x), sin(2x), cos(3x), ..., cos(nx), sin(nx); where n is the multiplicity limit displayed in the epilogue message.

Trigonometric: -1, 0, 1, tan(x)*sec(x), cot(x)*csc(x), $\sec^2 x$, cot x, tan x, sec x, csc x, sin y, cos y, cos x, cos(x)*sin(y), cos(x)*cos(y), sin x, sin(x)*sin(y), sin(x)*cos(y), sin(x)*cos(x), $\sin^2 x$.

8.1 Rational Program

```
10 GOTO 190
20 A=(2*X-(3*X+4)/(X-2))/(X-(10*X+4)/(2*X+3))
30 RETURN
40 REM  RATIONAL program of the PICOMATH-80 demonstration symbolic
50 REM  math package.  Copyright (c) 5/20/80 and trademark by
60 REM  The Soft Warehouse, Box 11174, Honolulu, Hawaii 96828.
70 REM
80 REM  Adapted with permission for ??? language dialect.
90 REM  Permission to install PICOMATH-80 or a translation of it on
100 REM  additional machines is hereby granted for a royalty of $2
110 REM  each, payable to The Soft Warehouse, provided the remarks and
120 REM  printed messages are all included, modified only as necessary.
130 REM  The reference manual contains more usage information, the latest
140 REM  unabridged program listings, an adaptation guide, and an explanation
150 REM  of how PICOMATH works.  This manual and machine-readable versions
160 REM  customized for many popular machines are available at modest cost
170 REM  from some hardware manufacturers, most computer stores, and
180 REM  Programma International at 2908 N. Naomi St., Burbank, CA 91504.
190 Z=LOG(2): X=Z: T=2.0/3.0: U=T^3: N=0: K=0: D=1: S=1: M=1
200 N=N+1: D=D/13: IF U<>(D+T)^3 THEN 200
210 DIM A(N+N+1), E(N): A(1)=0
220 D=9*D/8: IF U=(D+T)^3 THEN 220
230 PRINT "Expanded on a common denominator, reduced to lowest terms,"
240 K=K+1: IF K>N THEN 370
250 J=3
260 J=J-1: IF J<0 THEN 370
270 X=Z: GOSUB 20: R=A
280 X=Z*(1+D): GOSUB 20: W=ABS(A-R)
290 X=Z*(1-D): GOSUB 20: Y=ABS(A-R): IF Y>W THEN W=Y
300 Y=Z: Z=Z-S: S=-S-SGN(S): I=1: GOTO 350
310 R=R-A(I): U=ABS(R): W=W+E(I)+U*D: IF 9*W>=U THEN 260
320 X=Y-A(N+I): R=X/R: T=ABS(R): W=(T+ABS(X)/U)*D+W*T/U
330 IF 9*W>=T THEN 260
340 I=I+1
350 IF I<K THEN 310
360 M=K: A(K)=R: A(K+N)=Y: E(K)=W: GOTO 240
370 PRINT "  the formula on line 20 is approximately A ="
380 PRINT: A(N+M)=1: I=M: K=M-1: GOTO 510
390 I=I-1: A(N+I) = A(K)*A(I+1) - A(N+K)*A(N+I+1)
400 IF I=M-1 THEN 440
410 FOR J=I+1 TO M-1
420   A(N+J) = A(N+J) + A(K)*A(J+1) - A(N+K)*A(N+J+1)
430 NEXT J
440 IF K=1 THEN 540
450 A(I) = A(K-1)*A(N+I) - A(N+K-1)*A(I+1)
460 IF I=M-1 THEN 500
470 FOR J=I+1 TO M-1
480   A(J) = A(J) + A(K-1)*A(N+J) - A(N+K-1)*A(J+1)
490 NEXT J
500 A(M)=A(M)+A(K-1): K=K-2
510 IF K>0 THEN 390
520 L=I: H=M: GOSUB 780
```

```
530 L=N+I: H=N+M: GOTO 560
540 L=N+I: H=N+M: GOSUB 780
550 L=I+1: H=M
560 PRINT "----------------------------------------------------------------"
570 GOSUB 790
580 C=0: R=0: S=SGN(Z): X=Z-S/2: PRINT
590 FOR I=1 TO M
600   W=0: Z=0: GOSUB 20: IF ABS(A)>R THEN R=ABS(A)
610   FOR G=K TO J STEP -1
620     W=W*X+A(G)
630   NEXT G
640   FOR G=H TO L STEP -1
650     Z=Z*X+A(G)
660   NEXT G
670   X=X-S: W=ABS(A-W/Z): IF W>C THEN C=W
680 NEXT I
690 IF 9*C<=SQR(SQR(D))*R THEN 730
700 PRINT "Sampling suggests significant discrepancy.  Perhaps line 20"
710 PRINT "  is irrational, of excessive degree, or too sensitive to"
720 PRINT "  limitations of this BASIC arithmetic."
730 PRINT "For comparison, list line 20.  If desired, alter it to assign"
740 PRINT "  A any expression that simplifies to a polynomial in X of"
750 PRINT "  degree <=";INT(N/2);"divided by a polynomial in X of degree <=";
760 PRINT    INT((N-1)/2);"  Then rerun but do NOT save altered program."
770 STOP
780 J=L: K=H
790 F=0: O=1
800 FOR G=L TO H
810   Y=ABS(A(G)): IF Y>F THEN F=Y
820 NEXT G
830 FOR G=L TO H
840   C=INT(A(G)+.5): X=ABS(A(G)-C): IF X<.01 AND 99*X<F THEN A(G)=C
850 NEXT G
860 IF H=L THEN 1000
870 FOR G=H TO L+1 STEP -1
880   R=A(G):  IF R=0 THEN 990
890   IF O=0 THEN 930
900   IF R=-1 THEN PRINT "-";
910   IF ABS(R)=1 THEN 960
920   GOTO 950
930   IF R<0 THEN PRINT " - ";:  R=-R:  GOTO 950
940   PRINT " + ";
950   IF R<>1 THEN PRINT R;
960   PRINT " X ";
970   IF G>L+1 THEN PRINT "^"; G-L;
980   O=0
990 NEXT G
1000 IF O=1 THEN PRINT A(L): RETURN
1010 IF A(L)>0 THEN PRINT " + "; A(L)
1020 IF A(L)<0 THEN PRINT " - "; -A(L)
1030 IF A(L)=0 THEN PRINT
1040 RETURN
1050 END
```

7.2 Polynomial Program

```
10 GOTO 190
20 A = (X+1)^6 + (X+Y+1)^4 + (X+Y+Z+1)*(X+Y+Z-1)
30 RETURN
40 REM  POLYNOMIAL program of the PICOMATH-80 demonstration symbolic
50 REM  math package.  Copyright (c) 5/20/80 and trademark by
60 REM  The Soft Warehouse, Box 11174, Honolulu, Hawaii 96828.
70 REM
80 REM  Adapted with permission for ??? language dialect.
90 REM  Permission to install PICOMATH-80 or a translation of it on
100 REM  additional machines is hereby granted for a royalty of $2
110 REM  each, payable to The Soft Warehouse, provided the remarks and
120 REM  printed messages are all included, modified only as necessary.
130 REM  The reference manual contains more usage information, the latest
140 REM  unabridged program listings, an adapatation guide, and an explanation
150 REM  of how PICOMATH works.  This manual and machine-readable versions
160 REM  customized for many popular machines are available at modest cost
170 REM  from some hardware manufacturers, most computer stores, and
180 REM  Programma International, 2908 N. Naomi St., Burbank, CA 91504.
190 DIM A(26): O=1: N=0: Y=0: Z=0
200 PRINT "Enter E for Expansion, D for Derivative with respect to X,"
210 INPUT; "  or I for Integral with respect to X:", A$: PRINT
220 IF A$="E" THEN PRINT "Approximately,  A = ";: GOTO 260
230 IF A$="D" THEN PRINT "Approximately,  dA/dX = ";: GOTO 260
240 IF A$<>"I" THEN 200
250 PRINT "Approximately,  S A dX = ";
260 X=3: GOSUB 20: B=A
270 X=-3: GOSUB 20: C=B-A: B=B+A
280 X=2: GOSUB 20: D=A
290 X=-2: GOSUB 20: E=A
300 X=1: GOSUB 20: F=A
310 X=-1: GOSUB 20: G=A
320 X=0: GOSUB 20: H=A
330 V=6: B$="": A=(((B/6-D-E)/5+(F+G)/2)/2-H/3)/12: GOSUB 820
340 V=5: A=((C/4+E-D)/5+(F-G)/4)/12: GOSUB 820
350 V=4: A=((D+E-(B/3+13*(F+G))/4)/2+H/3*7)/6: GOSUB 820
360 Y=2: GOSUB 20: I=A
370 Y=-2: GOSUB 20: J=A
380 Y=1: GOSUB 20: K=A
390 Y=-1: GOSUB 20: L=A
400 X=-2: GOSUB 20: M=A
410 X=1: GOSUB 20: P=A
420 Y=1: GOSUB 20: Q=A
430 X=-1: GOSUB 20: R=A
440 Y=-2: GOSUB 20: S=A
450 Y=-1: GOSUB 20: T=A
460 V=3: B$="Y": A=((F-E+M-P)/3+G-H+L-T)/2: GOSUB 820
470 B$="": A=((13*(G-F)-C)/8+D-E)/6: GOSUB 820
480 V=2: B$="Y^2": A=((P+Q+R+T)/2-F-G-K-L)/2+H: GOSUB 820
490 B$="Y": A=((Q-P+R-T)/2+L-K)/2: GOSUB 820
500 B$="": A=((B/9-(D+E)/2*3)/5-H/9*49+3*(F+G))/4: GOSUB 820
510 V=1: B$="Y^3": A=((K-J+S-R)/3+G-H+L-T)/2: GOSUB 820
520 B$="Y^2": A=((P-R+Q-T)/2+G-F)/2: GOSUB 820
```

```
530 B$="Y": A=((5*T+Q)/2+(E-F+J-K-M-S-(P+R)/2)/3)/2-G+H-L: GOSUB 820
540 E=E-D: Z=-1: Y=0: GOSUB 20: D=A
550 X=0: GOSUB 20: B=A
560 B$="Z": A=D-B+H-G: GOSUB 820
570 B$="": A=((3*E+C/3)/5+3*(F-G))/4: GOSUB 820
580 V=0: B$="Y^4": A=(((I+J)/4-K-L)/3+H/2)/2: GOSUB 820
590 B$="Y^3": A=((I-J)/2+L-K)/6: GOSUB 820
600 B$="Y^2": A=(2*(K+L)-(I+J)/8)/3-H/4*5: GOSUB 820
610 Z=1: GOSUB 20: C=A: Y=-1: GOSUB 20
620 B$="YZ": A=L-H+C-A: GOSUB 820
630 B$="Y": A=((J-I)/4+2*(K-L))/3: GOSUB 820
640 B$="Z^2": A=(C+B)/2-H: GOSUB 820
650 B$="Z": A=(C-B)/2: GOSUB 820
660 B$="": A=H: GOSUB 820
670 IF O=1 THEN PRINT A: GOTO 690
680 PRINT
690 F=0: G=0: X=ATN(1): Y=EXP(1)/5: Z=LOG(2): GOSUB 1000
700 X=-X: GOSUB 1000: Y=-Y: GOSUB 1000: Z=-Z: GOSUB 1000
710 F=F*F: G=G*G+F: IF F=0 THEN 770
720 IF (G/F)^3=1 THEN 770
730 PRINT "Sampling suggests significant discrepancy.  Perhaps line 20 is"
740 PRINT "  not equivalent to a polynomial in X, Y and Z, or the polynomial"
750 PRINT "  is of excessive degree or the expression is too sensitive"
760 PRINT "  to limitations of this BASIC arithmetic."
770 PRINT "For comparison, list line 20.  If desired, alter it to assign to"
780 PRINT "  A any expression equivalent to an expanded polynomial in X, Y"
790 PRINT "  and Z.  The total degree of any expanded term containing X, Y"
800 PRINT "  and Z should not exceed 6, 4 and 2 respectively."
810 STOP
820 GOSUB 980: N=N+1: A(N)=A
830 W=V: IF A$="E" THEN 870
840 IF A$="I" THEN W=V+1: A=A/W: GOTO 860
850 A=V*A: W=W-1
860 GOSUB 980
870 IF A=0 THEN RETURN
880 IF O=0 THEN 910
890 IF A=-1 THEN PRINT "-";: A=-A
900 GOTO 930
910 IF A<0 THEN PRINT " - ";: A=-A: GOTO 930
920 PRINT " + ";
930 IF A<>1 OR W=0 AND B$="" THEN PRINT A;
940 IF W>0 THEN PRINT "X";
950 IF W>1 THEN PRINT "^";W;
960 PRINT B$;
970 O=0: RETURN
980 U=INT(A+.5): IF ABS(U-A)<.01 THEN A=U
990 RETURN
1000 GOSUB 20: IF ABS(A)>F THEN F=ABS(A)
1010 W=((A(14)*Y+A(9)*X+A(15))*Y+(A(6)*X+A(10))*X+A(16))*Y+((A(4)*X+A(7))*X+A(11))*X
1020 W=(W+A(18))*Y+(((((A(1)*X+A(2))*X+A(3))*X+A(5))*X+A(8))*X+A(13))*X
1030 W=ABS(W+A(21)+(A(12)*X+A(17)*Y+A(19)*Z+A(20))*Z-A)
1040 IF W>G THEN G=W
1050 RETURN
1060 END
```

7.3 Trigonometric Program

```
10 GOTO 190
20 A = SIN(X-Y+PI/2) + COS(2*X) - FNCOT(X)*FNSEC(X)/FNCSC(X)
30 RETURN
40 REM  TRIGONOMETRIC program of the PICOMATH-80 demonstration
50 REM  symbolic math package.  Copyright (c) 5/18/80 and trademark
60 REM  by The Soft Warehouse, Box 11174, Honolulu, Hawaii 96828.
70 REM
80 REM  Adapted with permission for ??? language dialect.
90 REM  Permission to install PICOMATH-80 or a translation of it on
100 REM  additional machines is hereby granted for a royalty of $2
110 REM  each, payable to The Soft Warehouse, provided the remarks and
120 REM  printed messages are all included, modified only as necessary.
130 REM  The reference manual contains more usage information, the latest
140 REM  unabridged program listings, an adaptation guide, and an explanation
150 REM  of how PICOMATH works.  This manual and machine-readable versions
160 REM  customized for many popular machines are available at modest cost
170 REM  from some hardware manufacturers, most computer stores, and
180 REM  Programma International at 2908 N. Naomi St., Burbank, CA 91504.
190 DIM A(18): P=ATN(1): PI=4*P: V=1: O=1: N=0: Y=0
200 DEF FNSEC(X)=1/COS(X)
210 DEF FNCSC(X)=1/SIN(X)
220 DEF FNCOT(X)=1/TAN(X)
230 PRINT "Enter E for Expand, D for Derivative with respect to X,"
240 INPUT;"  or I for Integral with respect to X:", A$
250 IF A$<>"E" AND A$<>"D" AND A$<>"I" THEN 230
260 PRINT: PRINT "Approximately, line 20 is equivalent to  "
270 IF A$="E" THEN PRINT "A = ";
280 IF A$="D" THEN PRINT "dA/dX = ";
290 IF A$="I" THEN PRINT "S A dX = ";
300 X=P: GOSUB 20: D=A
310 X=-X: GOSUB 20: E=D-A: D=D+A
320 Y=P: GOSUB 20: S=A
330 Y=-Y: GOSUB 20: Q=A
340 X=-X: GOSUB 20: T=S-A: S=S+A
350 Y=-Y: GOSUB 20: R=Q-A: Q=Q+A
360 X=3*X: Y=Y+Y: GOSUB 20: W=A
370 X=-X: Y=-Y: GOSUB 20: Z=W-A: W=W+A
380 Y=0: GOSUB 20: K=A
390 X=-X: GOSUB 20: J=K+A: K=A-K
400 X=2*PI/3: GOSUB 20: H=A
410 X=-X: GOSUB 20: I=H-A: H=H+A
420 X=X/2: GOSUB 20: G=A
430 X=-X: GOSUB 20: F=G+A: G=A-G
440 X=X/2: GOSUB 20: B=A
450 X=-X: GOSUB 20: C=B-A: B=B+A
460 X=5*X: GOSUB 20: M=A
470 X=-X: GOSUB 20: L=M+A: M=A-M
480 X=SQR(2): Y=SQR(3)/2: P=X-2+X
490 B$="TAN X SEC X": IF A$="I" THEN B$="SEC X"
500 IF A$="D" THEN B$="(2 SEC^3 X - SEC X)"
510 A=3*((C+M)/2-(E+K)*X+(G+I)*Y)/16: GOSUB 1140
520 B$="COT X CSC X": IF A$="D" THEN B$="(CSC X - 2 CSC^3 X)"
```

```
530 IF A$="I" THEN B$="CSC X": V=-1
540 A=3*((F-H)/2+(J-D)*X+(B-L)*Y)/16: GOSUB 1140
550 B$="SEC^2 X": V=1: IF A$="I" THEN B$="TAN X"
560 IF A$="D" THEN V=2: B$="TAN X SEC^2 X"
570 A=3*((B+F+H+L)/2-D-J)/8: GOSUB 1140
580 B$="COT X": IF A$="I" THEN B$="LN SIN X"
590 IF A$="D" THEN B$="CSC^2 X": V=-1
600 A=(Y*(G-I+3*(C-M))+3*(K-E))/8: GOSUB 1140
610 B$="TAN X": V=1: IF A$="D" THEN B$="SEC^2 X"
620 IF A$="I" THEN V=-1: B$="LN COS X"
630 A=(Y*(C-M+3*(G-I))+3*(K-E))/8: GOSUB 1140
640 B$="SEC X": V=1: IF A$="D" THEN B$="TAN X SEC X"
650 IF A$="I" THEN B$="LN TAN(X/2+PI/4)"
660 A=(9*(F-H)/2+3*(J-D)*X+(B-L)*Y)/16: GOSUB 1140
670 B$="CSC X": IF A$="I" THEN B$="LN TAN(X/2)"
680 IF A$="D" THEN V=-1: B$="COT X CSC X"
690 A=(9*(C+M)/2-3*(E+K)*X+(G+I)*Y)/16: GOSUB 1140
700 B$="": IF A$="D" THEN V=0
710 IF A$="I" THEN B$="X"
720 A=(3*(B+L)-F-H)/8+(Q-J+W)/4+(S-X*D)/P/2: GOSUB 1140
730 B$="SIN Y": IF A$="I" THEN B$="X SIN Y"
740 A=((R-K+Z)/2+(E*X+T)/P)/2: GOSUB 1140
750 B$="COS Y": IF A$="I" THEN B$="X COS Y"
760 A=((J-Q-W)/2+(D*X-S)/P)/2: GOSUB 1140
770 B$="SIN X": V=1: IF A$="E" THEN B$="COS X"
780 IF A$="D" THEN V=-1
790 A=(3*(H-F)/2+(L-B)*Y)/2+(((3*X-6)*J+X*Q+(X-2)*(W-S))/2+(1-P)*D)/P
800 GOSUB 1140
810 B$="SIN X SIN Y": IF A$="E" THEN B$="COS X SIN Y"
820 A=(K-Z)*X/4-(X*(R-T)+2*(E+T))/P/2: GOSUB 1140
830 B$="SIN X COS Y": IF A$="E" THEN B$="COS X COS Y"
840 A=(D-X*Q/2+(X-2)*(J+S-W)/2)/P: GOSUB 1140
850 B$="SIN X": IF A$<>"E" THEN V=-V: B$="COS X"
860 A=X*K-E-(3*(C+M)/2+Y*(G+I))/2-(R+T)/P: GOSUB 1140
870 B$="COS X SIN Y": IF A$="E" THEN B$="SIN X SIN Y"
880 A=(Q-S)/2: GOSUB 1140
890 B$="COS X COS Y": IF A$="E" THEN B$="SIN X COS Y"
900 A=(E+E+R+T)/P: GOSUB 1140
910 B$="SIN X COS X": V=1: IF A$="D" THEN B$="(1 - 2 SIN^2 X)"
920 IF A$="I" THEN V=1/2: B$="SIN^2 X"
930 A=2*(E-K)+Y*(I-G+M-C): GOSUB 1140
940 B$="SIN^2 X": IF A$="I" THEN B$="(X - SIN X COS X)"
950 IF A$="D" THEN V=2: B$="SIN X COS X"
960 A=2*(D+J)-(3*(B+L)+F+H)/2: GOSUB 1140
970 IF O=1 THEN PRINT A;
980 F=0: G=0: X=EXP(1): Y=LOG(2): GOSUB 1220
990 X=-1: GOSUB 1220
1000 Y=-1: GOSUB 1220
1010 X=1: GOSUB 1220
1020 F=F*F: H=G*G+F: PRINT: IF F=0 THEN 1070
1030 IF G<.0001 OR ATN(H/F)=ATN(1) THEN 1070
1040 PRINT "Sampling suggests significant discrepancy.  Perhaps line 20"
1050 PRINT "  is not equivalent to an allowable expression, or line 20"
1060 PRINT "  is too sensitive to limitations of this BASIC arithmetic."
1070 PRINT "For comparison, list line 20.  If desired, alter it to assign"
```

```
1080 PRINT "  to A any expression equivalent to a linear combination of the"
1090 PRINT "  terms:  TAN X SEC X, COT X CSC X, SEC^2 X, COT X, TAN X,"
1100 PRINT "  SEC X, CSC X, 1, SIN Y, COS Y, COS X, COS X SIN Y,"
1110 PRINT "  COS X COS Y, SIN X, SIN X SIN Y, SIN X COS Y, SIN X COS X,"
1120 PRINT "  and SIN^2 X.  Then, reRUN but do NOT save altered program."
1130 STOP
1140 GOSUB 1200: N=N+1: A(N)=A: A=A*V: GOSUB 1200: IF A=0 THEN RETURN
1150 IF O=1 AND A>0 THEN 1180
1160 IF A<0 THEN PRINT " - ";: A=-A: GOTO 1180
1170 PRINT " + ";
1180 IF A<>1 OR B$="" THEN PRINT A;
1190 PRINT B$;: O=0: RETURN
1200 U=INT(A+.5): IF ABS(U-A)<.001 THEN A=U
1210 RETURN
1220 GOSUB 20: IF ABS(A)>F THEN F=ABS(A)
1230 C=COS(X): S=SIN(X): T=TAN(X): U=SIN(Y): V=COS(Y)
1240 W=(A(1)*T+A(3)/C+A(6))/C+(A(2)/T+A(7))/S+A(4)/T+A(5)*T+A(9)*U
1250 W=W+(A(14)+A(15)*U+A(16)*V+A(17)*C+A(18)*S)*S+A(10)*V
1260 W=ABS(W+A(8)+(A(11)+A(12)*U+A(13)*V)*C-A)
1270 IF W>G THEN G=W
1280 RETURN
1290 END
```

7.4 Fourier Program

```
10 GOTO 190
20 A=(COS(X)+SIN(X))^8
30 RETURN
40 REM  FOURIER program of the PICOMATH-80 demonstration
50 REM  symbolic math package.  Copyright (c) 5/20/80 and trademark
60 REM  by The Soft Warehouse, Box 11174, Honolulu, Hawaii 96828.
70 REM
80 REM  Adapted with permission for ??? language dialect.
90 REM  Permission to install PICOMATH-80 or a translation of it on
100 REM  additional machines is hereby granted for a royalty of $2
110 REM  each, payable to The Soft Warehouse, provided the remarks and
120 REM  printed messages are all included, modified only as necessary.
130 REM  The reference manual contains more usage information, the latest
140 REM  unabridged program listings, an adaptation guide, and an explanation
150 REM  of how PICOMATH works.  This manual and machine-readable versions
160 REM  customized for many popular machines are available at modest cost
170 REM  from some hardware manufacturers, most computer stores, and
180 REM  Programma International at 2908 N. Naomi St., Burbank, CA 91504.
190 F=ATN(1): PI=4*F: Q=1
200 Q=Q/9: T=ATN(1+Q): IF T<>F THEN 200
210 M=8: N=M+M+1: DIM A(N),C(N),S(N),U(M),V(M)
220 INPUT;"Enter E for Expand, D for Derivative or I for Integral:",A$
230 IF A$<>"E" AND A$<>"D" AND A$<>"I" THEN 220
240 Q=SQR(Q): T=0: F=0: O=0: S(1)=0: C(1)=1
250 W=2*PI/N: S(2)=SIN(W): C(2)=COS(W): W=2*C(2): PRINT
260 FOR K=3 TO N
270   C(K)=W*C(K-1)-C(K-2): S(K)=W*S(K-1)-S(K-2)
280 NEXT K
290 FOR K=1 TO N
300   X=2*(K-1)*PI/N: GOSUB 20: A(K)=A
310   T=T+A: X=ABS(A): IF X>F THEN F=X
320 NEXT K
330 T=T/N: GOSUB 950: H=T
340 PRINT "When the formula on line 20 is approximately transformed to a"
350 PRINT "  linear trigonometric polynomial,"
360 IF A$="D" THEN PRINT "dA/dX = ";: GOTO 440
370 IF A$="I" THEN PRINT "S A dX = ";: GOTO 400
380 PRINT "A = ";: IF T=0 THEN 440
390 O=1: PRINT T;: GOTO 440
400 IF T=0 THEN 440
410 O=1: IF T=-1 THEN T=1: PRINT "-";
420 IF T<>1 THEN PRINT T;
430 PRINT "X";
440 K=1
450 T=0
460 FOR J=1 TO N
470   T = T + A(J) * C(1+((K*J-K)MOD N))
480 NEXT J
490 T=2*T/N: B$="SIN(": IF A$="E" THEN B$="COS("
500 GOSUB 950: U(K)=T: IF A$="D" THEN T=-T
510 GOSUB 820: T=0
520 FOR J=1 TO N
```

```
530   T = T + A(J) * S(1+((K*J-K)MOD N))
540 NEXT J
550 T=2*T/N: B$="COS(": IF A$="E" THEN B$="SIN("
560 GOSUB 950: V(K)=T: IF A$="I" THEN T=-T
570 GOSUB 820
580 K=K+1: IF K<=M THEN 450
590 IF O=0 THEN PRINT 0;
600 F=0: G=0: X=EXP(1): PRINT
610 FOR K=1 TO 5
620   X=X/2: GOSUB 20: T=H
630   FOR J=1 TO M
640     T = T + U(J)*COS(J*X) + V(J)*SIN(J*X)
650   NEXT J
660   Z=ABS(A): IF Z>F THEN F=Z
670   Z=ABS(A-T): IF Z>G THEN G=Z
680 NEXT K
690 IF G<=9*Q*F THEN 740
700 PRINT "Sampling suggests significant discrepancy.  Perhaps line 20"
710 PRINT "  is not equivalent to a trigonometric polynomial in X,"
720 PRINT "  or it requires too large of a multiplicity, or it is"
730 PRINT "  too sensitive to limitations of this BASIC arithmetic."
740 PRINT "For comparison, list line 20.  If desired, alter it to assign"
750 PRINT "  A any expression equivalent to a polynomial in sines and"
760 PRINT "  cosines of radian angles of the form  n*X+c, where n is any"
770 PRINT "  integer of magnitude <="; M; "and c is a constant, which may"
780 PRINT "  involve the variable PI, representing the ratio of
790 PRINT "  circumference to diameter of a circle.  Then reRUN but do"
800 PRINT "  NOT save altered program."
810 STOP
820 IF A$="D" THEN T=T*K
830 IF A$="I" THEN T=T/K
840 GOSUB 950
850 IF T=0 THEN RETURN
860 IF O=0 THEN 930
870 IF T>0 THEN PRINT " + ";
880 IF T<0 THEN T=-T: PRINT " - ";
890 IF T<>1 THEN PRINT T;
900 O=1: PRINT B$;
910 IF K>1 THEN PRINT K;
920 PRINT "X)";: RETURN
930 IF T=-1 THEN PRINT "-";: T=1: GOTO 900
940 GOTO 890
950 Y=INT(T+.5): X=ABS(T-Y): IF X<.01 AND X<=Q*ABS(T) THEN T=Y
960 IF ABS(T)<Q*F THEN T=0
970 RETURN
980 END
```

INDEX