Ways to Implement Computer Algebra Compactly

A Personal History

by

David R. Stoutemyer

July, 2008

Outline

1. Why I am interested in **compact** computer algebra.
2. Computers and programmable calculators from 1975-1981.
3. Why compact implementation and data are still important.
4. Special-purpose methods:
    a) Univariate polynomials & series via dense coefficient arrays.
    b) Finessing computer algebra via evaluation and interpolation.
5. General-purpose methods:
    a) String-based computer algebra.
    b) Contiguous stack-based computer algebra.
    c) Compact linked-data computer algebra.

Why I am interested in compact computer algebra

- Before about 1977, computer algebra was available only on the largest mainframe computers.
- I wanted my engineering students to become familiar with this tool.
- My entire semester course computing budget was exhausted by the first assignment on our campus mainframe .
- Beginning 1975, programmable hand-held calculators by TI and HP, together with the MITS Altair personal computer kit offered opportunities to implement computer algebra more affordably.
- Computer algebra on hand-held calculators would make it practical to use in an ordinary classroom on a spontaneous as-needed basis.

For student problems, a compact program is more important than compact data or speed.

TI and HP hand-held programmable calculators introduced from **1975-77**:

- 49 to 960 interpreted “assembly-language” steps.
- 8 to 100 direct-address floating-point variables named, ~ R1 … R*n*
- A one-line display just wide enough to display 1 floating-point number.

Mass market personal computers from **1977 to 1981**
(Apple II, Commodore Pet, TRS-80, Atari 400 & 800)

- 64 kilobyte address space.
- 4 to 16 kilobytes consumed by ROM containing Basic or a subset of Basic.
- Entry-level systems often came with only 4 to 16 kilobytes of RAM
- 4 kilobytes was enough for about 100 lines of Basic.
- For storage of programs and data: Audio cassette, then later one or two floppy disk drives holding about 128 kb.

Why compact implementation is still important:

- Hoare’s law of large programs:

  “**Inside every large program is a small program struggling to emerge.**”

- Incomprehensibility increases with size.
- Development and maintenance time and cost grow with size.
- Carbon footprint grows with silicon footprint.
- Techniques for implementing computer algebra compactly are relevant to other kinds of programs.
- History is important: “**Those who cannot remember the past are condemned to repeat it**.” – Santayana
- # cell phones > # PDAs > # graphing calculators.
- Growth in of cell-phone and internet access speed << Moore’s law.

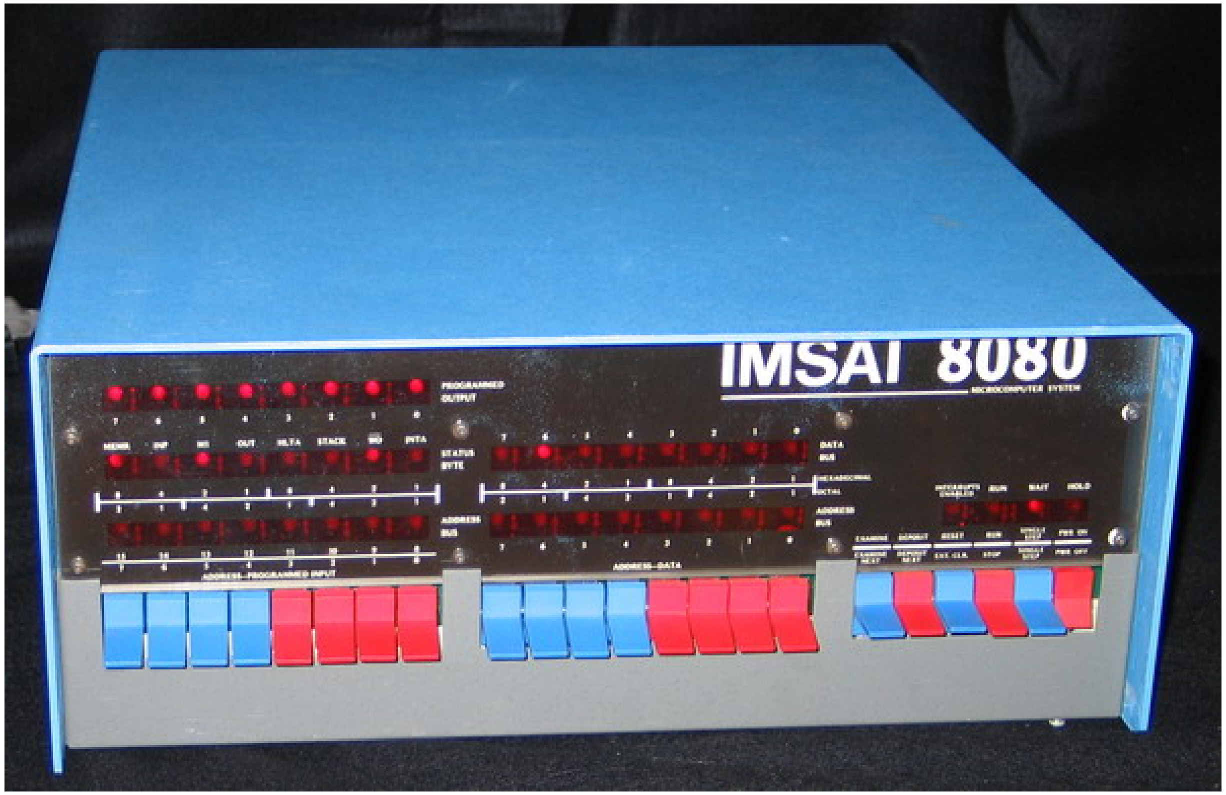
Special purpose algebra based on coefficient arrays

- Univariate series and polynomials can be implemented even on the first programmable hand-held calculators.
- Knuth gives algorithms for such polynomials and series in volume 2 of *The Art of Computer Programming*.
- Henrici (**1977**) describes such a series package for the HP 25, which had only 49 program steps and registers for 12 floating-point numbers.
- Unaware of that, I implemented a similar package on the HP-67, which offered 225 steps and 24 registers.

Size & speed of my **1977** HP-67 Maclaurin series operations

Operation	**Number of steps**	**Seconds**
subtraction	4	20
copying	5	10
negation	10	10
addition	14	10
substitute number	16	10
integration	16	10
multiplication	38	60
division	49	60
reversion	62	180

- muMath (**1978**) ran on computers using the Intel 8080 or Zilog Z-80 processor, with the CP/M operating system and 32 to 64 kilobytes of RAM.



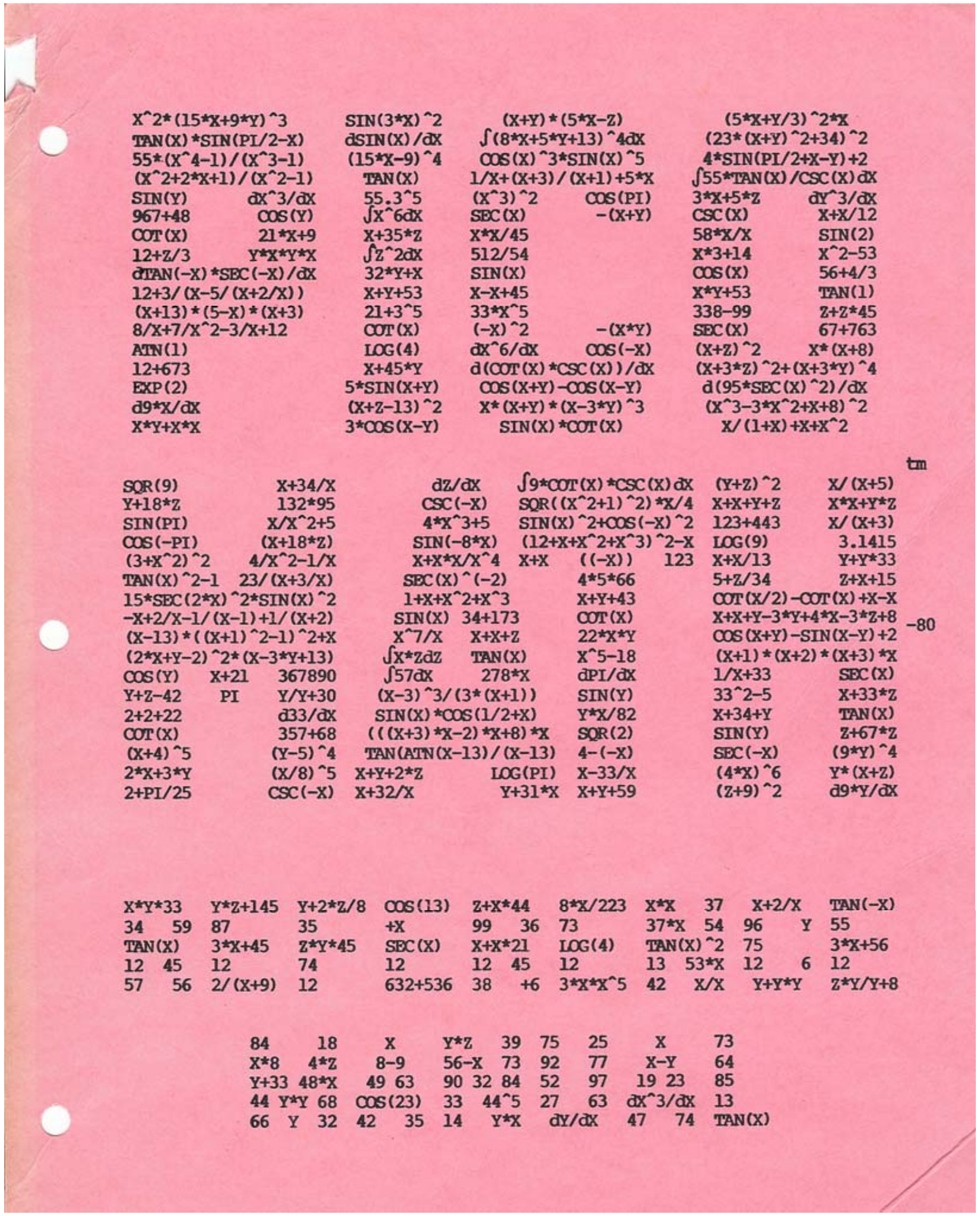
- These were hobbiest machines rather than mass market,

- Microsoft licensed muMath for the Radio Shack TRS-80 mass market computer, which used a Z-80 processor.

- In **1980** the Microsoft Z-80 card enabled CP/M and muMath to run on the Apple II computer.

- To introduce the mass market audience to computer algebra, I wrote some less ambitious computer algebra systems in their built-in Basic.

One of them was a shareware demonstration program:



```
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)
```

PicoMath^{tm} consists of 4 special purpose Basic programs.

Each program ran even on the Radio-Shack **pocket computer** (**1980**) that had only **1.5 kilobytes** of RAM.

The **Rational** program can expand and reduce over a common denominator a rational expression in *x*.

For example:

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

and

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

The **Polynomial** program can expand polynomials in *x*, *y* and *z*, with optional integration or differentiation.

For example:

$$(x+1)^6+(x+y+1)^4+(x+y+z+1)^2 \ \rightarrow\ x^6+6x^5+16x^4$$
$$+4x^3y+24x^3+6x^2y^2+12x^2y+22x^2+4xy^3+12xy^2$$
$$+14xy+2xz+12x+y^4+4y^3+7y^2+2yz+6y+z^2+2z+3,$$

$$\int\left((x^3-1)(x^3+1)+y^4+z^2\right)dx \ \rightarrow\ 0.142857x^7+xy^4+xz^2-x,$$

$$\frac{d}{dx}\left(x^6+xy^3+z^2\right) \rightarrow\ 6x^5+y^3.$$

The **Trigonometric** program can expand, many trigonometric expressions in *x* and *y*, with optional integration or differentiation.

$$\text{For example:}$$

$$\int 2\sin\left(\frac{x+y}{2}\right)\cdot\cos\left(\frac{x-y}{2}\right)dx \;\rightarrow\; x\cdot\sin(y)-\cos(x),$$

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

$$\frac{d}{dx}\left(\frac{1+\cos(2x)}{\cos(x)}+\frac{\sin(2x)}{\sin(x)}\right) \;\rightarrow\; -4\sin(x).$$

The **Fourier** program implements trigonometric collection for polynomials in sinusoids of *x*, with optional integration or differentiation:

For example:

$$64\sin(x)^4\cos(x) \;\rightarrow\; 3\cos(x)-3\cos(3x)-\cos(5x)+\cos(7x),$$

$$\int -4\sin(2x)^2\cos(x)-4\cos(x)^3+5\cos(x)\,dx \;\rightarrow\; 0.2\sin(5x),$$

$$\frac{d}{dx}\left(-4\sin(2x)^2\cos(x)-4\cos(x)^3+5\cos(x)\right) \;\rightarrow\; 5\sin(5x).$$

All four programs begin with the lines:

```
 10 GOTO 40
 20 A=expression the user wants simplified, integrated or differentiated
 30 RETURN
 40 …
```

No computer-algebra parsing is required: The program calls subroutine 20 for different pre-selected values of the independent variable(s), then interpolates an expression of the implemented class.

If the expression on line 20 and the interpolated expression have relatively close values at some additional points, then the coefficients of the interpolant are displayed separated by string constants such as "x^2 +" or "cos(x) +" to display the simplified, integrated or differentiated expression.

Otherwise a message is displayed encouraging the user to edit the expression on line 20 to be equivalent to the appropriate class.

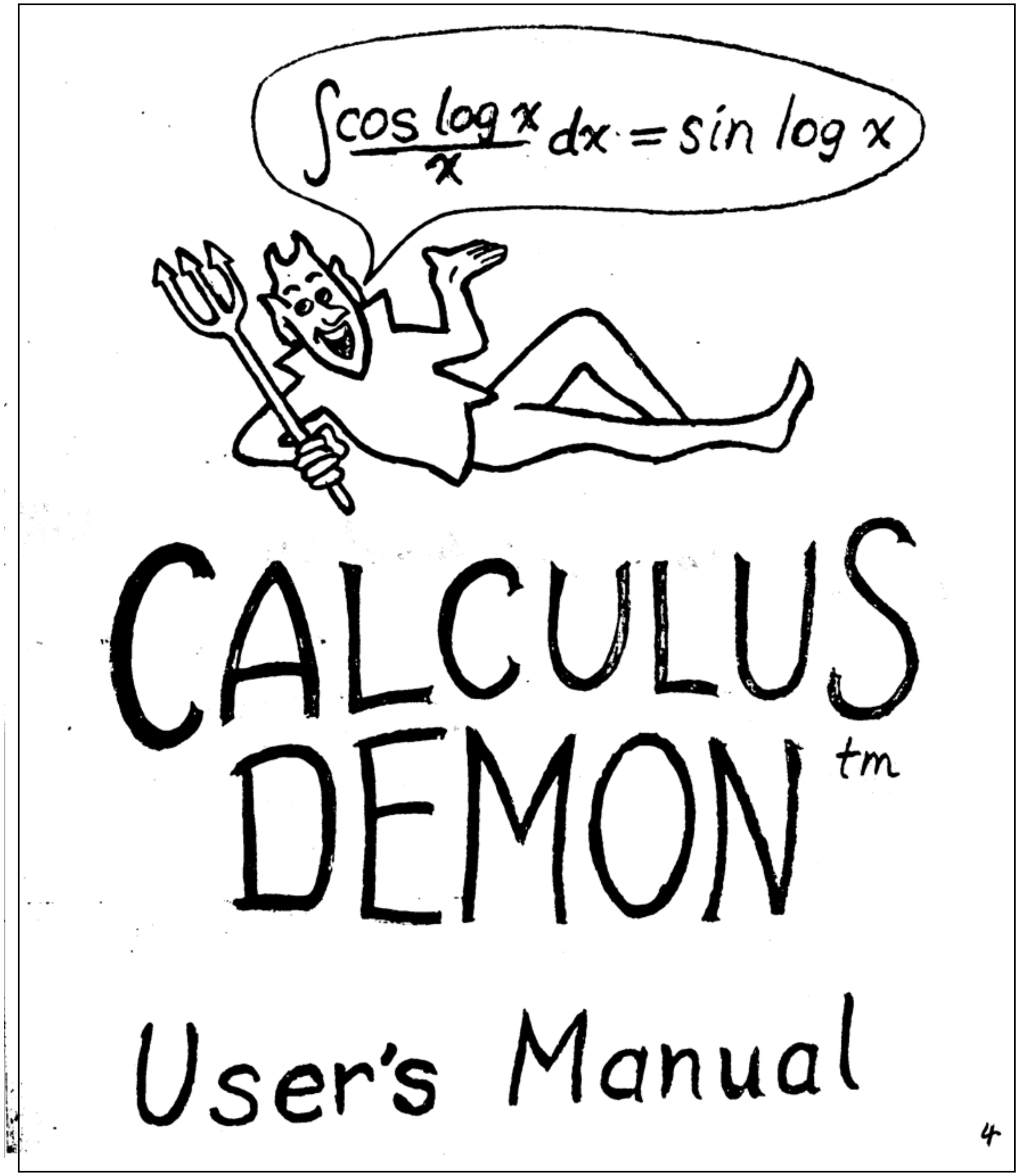
General-purpose Methods.
A: String-based computer algebra.

- It is possible to do computer algebra directly on strings containing expressions in ordinary infix notation.
- D. Strebe wrote such a program for the HP-41 hand-held calculator in **1980**.
- There were several earlier compact programs for doing differentiation using the pioneering string-processing language Snobol.
- Although there is the advantage of using the same representation internally as for I/O, it is inefficient to repeatedly scan to locate operators between operands.

General-purpose Methods
B: Contiguous stack-based computer algebra.

- Expression trees can be represented in Polish as a contiguous stack of tokens containing no parentheses.
- A parser can convert a string such as "$(\sin x + \cos x)^2 - 1$" into a stack of tokens such as $- \; 1 \; \wedge \; + \; \sin x \; \cos x \; 2$, with 2 at the bottom and the "–" at the top.
- Computer algebra can be done directly on this Polish representation, producing a simplified Polish representation such as $\cdot \; 2 \; \cdot \; \sin x \; \cos x$.
- This simplified Polish representation can then be converted for display into the string "$2{\cdot}\sin x{\cdot}\cos x$".

Calculus Demon used this method:



CALCULUS DEMON tm

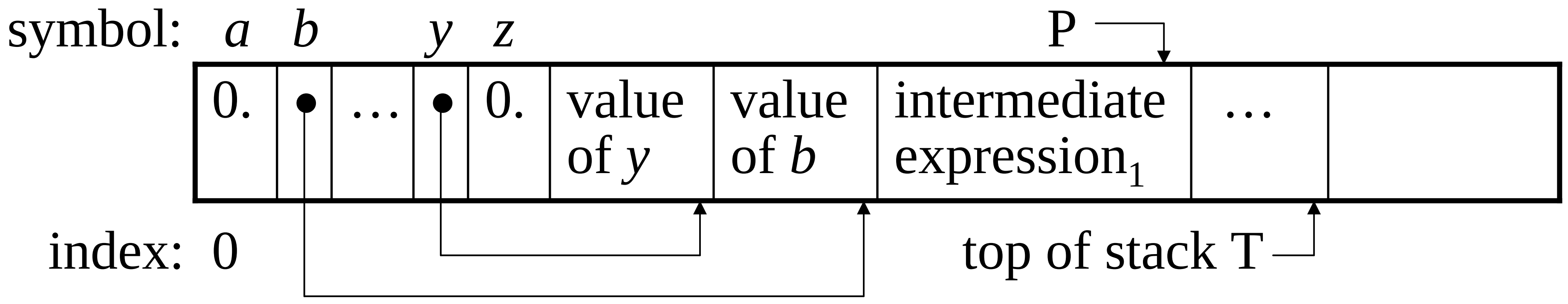
User's Manual

4

Calculus Demon for the Atari 400 and 800 computers:

- About 800 lines of Basic, several statements per line.
- It can differentiate, modestly integrate and simplify multivariate general expressions, including fractional powers, exponentials, logarithms, trigonometric functions and their inverses: **Formac Jr**.
- Optional transformations include polynomial expansion and trigonometric collection.
- The stack of Polish expressions is an array of floats.
- Tags: 0.0 means the float below it represents a number, 1.0 through 26.0 represent shifted character codes of one-letter variables, 27.0 means the expression below it is the argument of LN, etc.
- These tags for numbers and variables, unary operators and functions, then binary operators and functions are grouped together so that mere range checks determine the number of operands or arguments.

More details about Calculus Demon



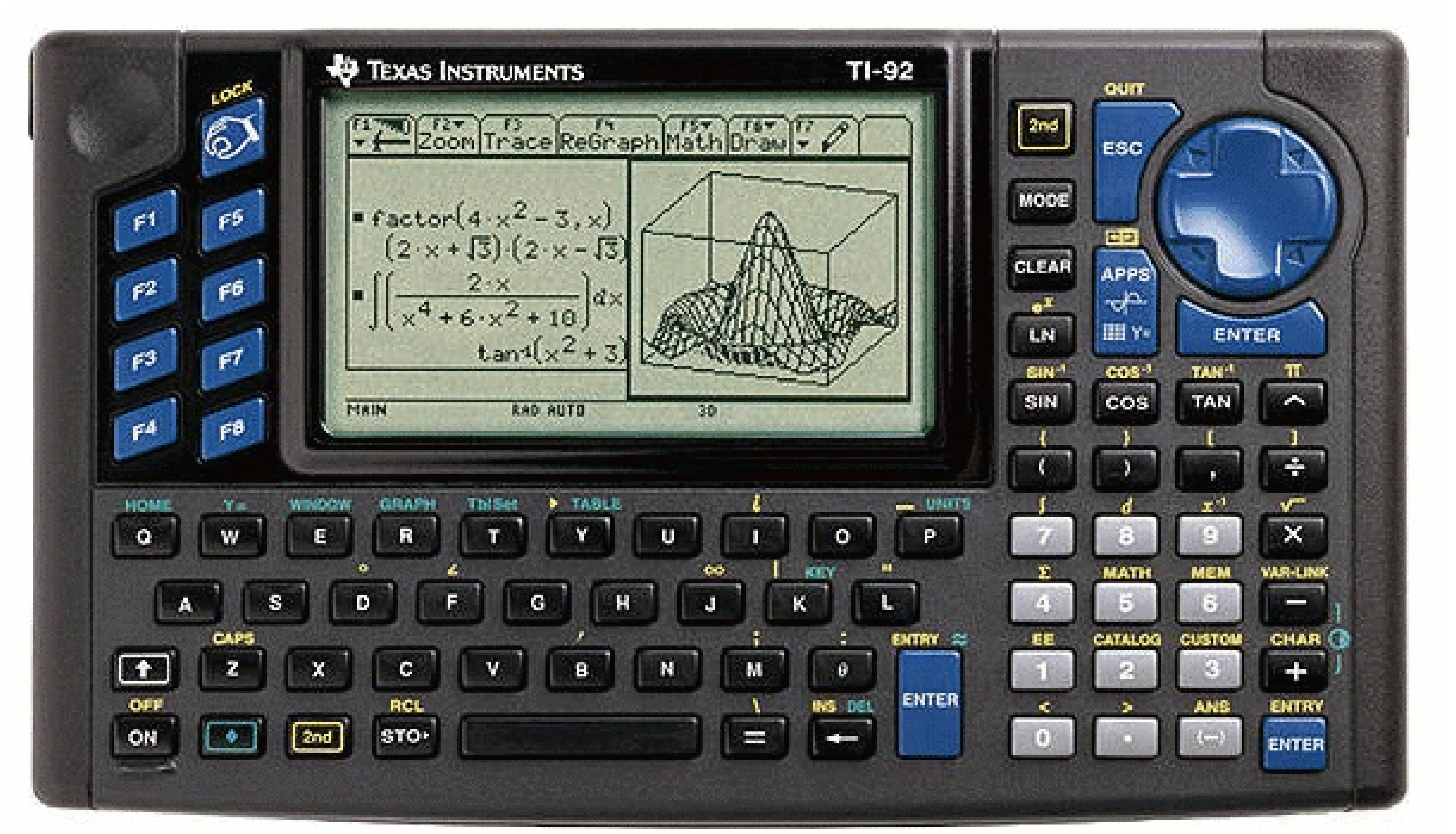
- The deepest 26 elements of the expression stack, indexed 0 through 25 are a symbol table of indices for values of user variables *a* through *z*.
- A table entry of 0.0 means the corresponding variable has no assigned symbol value.
- Assigned values are stored contiguously above that.
- The working stack where simplified results are developed is above the assigned values.
- To save time, indices of deeper operands are saved into local variables. (A **remember stack** rather than a pure stack discipline.)

The computer algebra in TI products uses this method, but:

- It is written in C rather than Basic.
- Rather than floating point, the expression stack is of 8, 16 or 32 bit unsigned integers, depending on the product.
- Numbers are fixed-precision BCD floats or arbitrary-precision rational numbers containing length fields.
- It is significantly more powerful than Calculus Demon – roughly comparable overall to *Derive*.
- More implementation details are available in the TI-89/TI-92 Plus *Developers Guide* at

  http://education.ti.com/educationportal

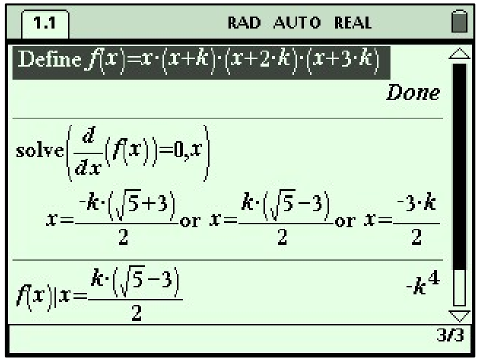
The TI-92, released in **1995**



TI-Nspire™ Applications

PC, Macintosh and handheld: Same functionality

Optional CAS



Define $f(x)=x\cdot(x+k)\cdot(x+2\cdot k)\cdot(x+3\cdot k)$ *Done*

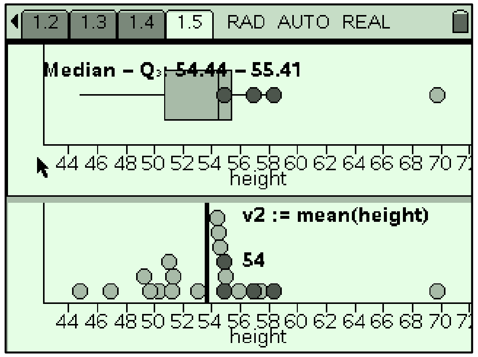
$\text{solve}\left(\frac{d}{dx}(f(x))=0,x\right)$

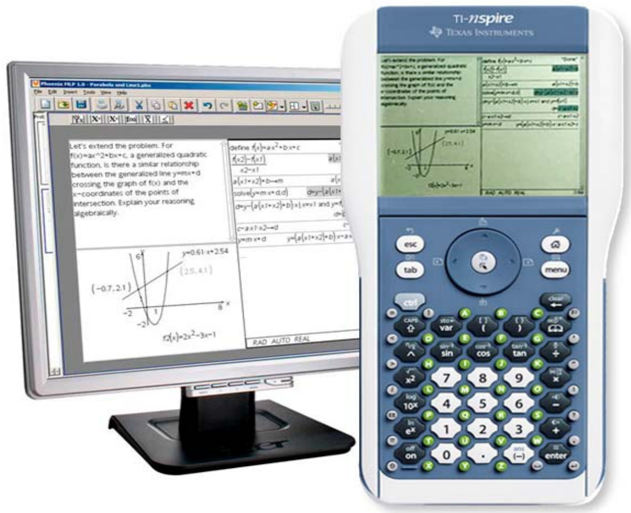
$x=\frac{-k\cdot(\sqrt{5}+3)}{2}$ or $x=\frac{k\cdot(\sqrt{5}-3)}{2}$ or $x=\frac{-3\cdot k}{2}$

$f(x)|x=\frac{k\cdot(\sqrt{5}-3)}{2}$ $\quad -k^4$

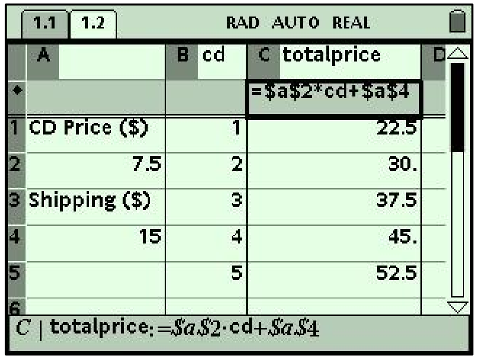
3/3

Data and Statistics





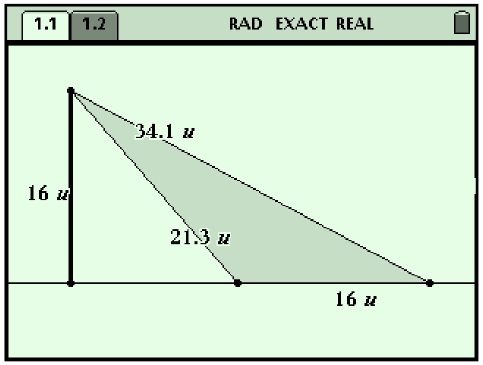
Lists and Spreadsheet



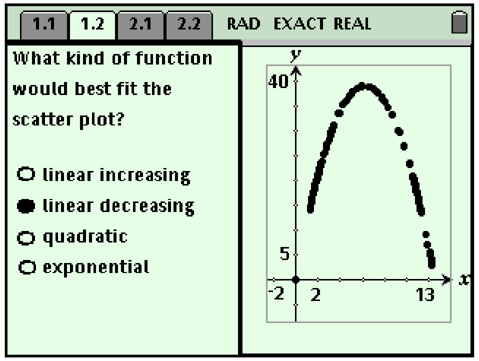
	A	B cd	C totalprice
◆			=a2*cd+a4
1	CD Price ($)	1	22.5
2	7.5	2	30.
3	Shipping ($)	3	37.5
4	15	4	45.
5		5	52.5

C | totalprice:=*a2*·cd+*a4*

Graphs & Geometry



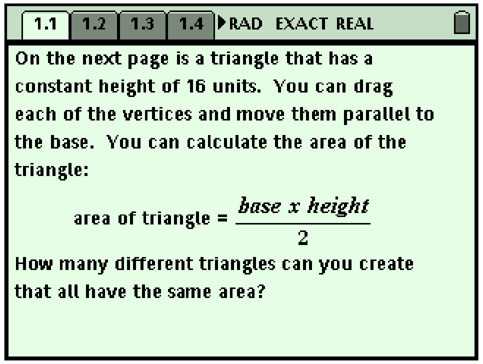
Exercises & Exams



What kind of function would best fit the scatter plot?

○ linear increasing
● linear decreasing
○ quadratic
○ exponential

Notes

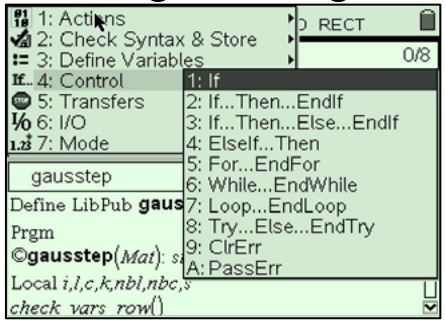


On the next page is a triangle that has a constant height of 16 units. You can drag each of the vertices and move them parallel to the base. You can calculate the area of the triangle:

area of triangle = $\frac{\textit{base x height}}{2}$

How many different triangles can you create that all have the same area?

Programming



```
gausstep
Define LibPub gaus
Prgm
©gausstep(Mat): s
Local i,l,c,k,nbl,nbc,s
check_vars_row()
```

Data Collection

Some advantages of contiguous Polish over linked storage:

- No space is wasted on pointers within an expression.
- Contiguous data improves speed for caches and virtual memory.
- The data is relocatable, which is also good for serialization.
- It isn’t necessary to implement garbage collection or reference counts.
- There are no annoying pauses for garbage collection during plotting, etc., making it particularly suitable for real-time applications.

Some disadvantages compared to linked storage:

- There is no sharing of common sub-expressions within an expression.
- Access is slower to deeper operands to which pointers haven’t been saved into local variables.
- Deletion of garbage on the stack is a ubiquitous responsibility: Almost every procedure must leave no garbage between what was the top of the expression stack upon entry and any alleged result expressions left above that upon exit.

C: Implementing linked-data computer algebra compactly:

Lisp-like languages have advantages for implementing CAS:

- They include a built-in garbage collector that is fast even for small blocks consisting of two pointers, which are most common.
- They include arbitrary-precision arithmetic.
- They include built-in data-base primitives for fast querying of information about variables and operators, such as their type, precedence, associativity, commutativity, linearity and symmetries.
- They include an interpreter that permits run-time definition of new functions and operators.
- They include powerful mapping functions that efficiently factor out common looping or recurrence code-space overhead.

These features can be added to C or C++, but that additional work must be done efficiently before implementing computer algebra.

Features of **muLisp** that lead to compact implementation (and speed):

- Symbols that don’t have assigned values evaluate to themselves rather than an error. This is what you want for computer algebra.
- Excess trailing formal parameters are local variables initialed to Nil.
- ((functionName$_1$ …) …)
  …
  ((functionName$_n$ …) …)
  ≡
  (Cond ((functionName$_1$ …) …)
  …
  ((functionName$_n$ …) …))

(((functionName$_1$ …) …)
…
((functionName$_n$ …)…))
≡
(ProgN (Cond ((functionName$_1$ …) …)
…
((functionName$_n$…)…)))

- Cons cells, symbols, their print names, numbers and their binary data are stored in separate contiguous areas so that type-checking can be done by mere comparison with boundary addresses.

More features of muLisp for compactness and speed:

- If necessary, region sizes are reallocated.
- Atoms and Cons cells are aligned on even addresses so that the least significant bit of addresses can be used as a temporary marker bit.
- Function definition are Cdr-coded: Lists in function bodies are arrays of pointers rather than linked Cons cells. The least significant bit of a pointer is 1 iff it is the last pointer in an array.
- If two or more pointers in a function definition point to syntactically identical forms, only one copy of that form is stored.
- This **condensing** process can optionally be done **between** functions too, and such functions are excluded from garbage collection, saving time.

Still more features of muLisp for compactness and speed:

- (Zap-string *symbol*) replaces the *symbol*'s print-string with the null string.
- Zapping non-public function and variable names as the last step in a program definition saves space.
- The assembly-language implementation of the muLisp interpreter is exceptionally fast and space efficient:
- The compiler was used only for a few bottom-level computer algebra functions that were invoked very frequently and didn't invoke other time-consuming functions.
- (Compilation increased code size by about a factor of 2 to 3, while increasing speed by at most a similar factor.)

Features of **muMath** for compact implementation and speed:

- Many algorithms were implemented incrementally by putting on the property list of the function or operator name several small functions for handling different kinds of operands.
- For example, there were separate functions for differentiating numbers, variables, sums, etc., making it easy for users to add differentiation of Bessel functions etc. without doing surgery on a large monolithic function.
- The assembly-language Assoc function did the dispatching, so it was faster than interpreting conditional branches.
- Each muLisp arithmetic function invoked an associated trap function to optionally catch error throws for non-numeric arguments. Each muMath trap function dispatched to separate functions for adding a number to a variable, a sum to a product etc.
- Thus the most common case of combining numbers is tried first in machine language rather than after an interpreted test.

muMath was bundled with the first commerical luggable computer in 1981

muLisp for the Intel 8086 and 8088

- The first mass-market PC that could address more than 64 kilobytes was the IBM-PC, introduced in **1981**, which used the Intel 8088.
- MS-DOS left 640 kilobytes for implementing muLisp and muMath.
- The memory was segmented into 64-kilobyte segments: 16-bit offsets were automatically added to the contents of an appropriate 20-bit segment register to determine addresses.
- We used the data segment for 16-bit Cars, the extra segment for 16-bit Cdrs, and the stack segment for return and argument offsets.
- There could be up to 6 segments for Cdr-coded function definitions.
- When added to the code-segment register, the first 16-bit offset in each function definition was a pointer to the symbol Lambda.
- Gaps were inserted if necessary so that this happened in only one code segment. If it didn't happen with the current segment, successive code segments were tried until it did happen.

Derive for MS-DOS

- muMath had a teletype-style interface, like MS-DOS.
- Most potential users also wanted expression plots, 2D display of results and a windowing menu-driven interface.
- So we introduced *Derive* with its own windowing system in 1988 as replacement for muMath.
- We needed approximate arithmetic not only for plots, but also for approximate integration, equation solving etc.
- We already had exact rational arithmetic.
- Therefore **rounded rational arithmetic** was the most compact way to introduce approximate arithmetic, and it was easy to make it have adjustable precision.
- At the default 6 digits of precision, which was sufficient for most plots, it was about the same speed as the then-prevalent 8 to 16 digit software floating point, so plot speed was about the same.

There was a ROM version of *Derive* for DOS for the HP 95LX

HEWLETT PACKARD 95LX
DERIVE integrates symbolically & plots
3D-plot
Center x:0
For the Hewlett Packard 95LX
DERIVE
A Mathematical Assistant
Copyright © 1991 Soft Warehouse, Inc. All rights reserved worldwide.
Soft Warehouse
HONOLULU • HAWAII
This side up. Insert carefully in direction of arrow.
Parts made in Japan.

Derive for Windows

- The Intel 386 was the first x86 processor to support an unsegmented 32-bit address space and to make Microsoft Windows acceptable.
- So we implemented a 386 assembly-language version of muLisp that employed full 32-bit addresses.
- Using this together with C++, we wrote a Windows interface for the *Derive* engine.
- By this time floating-point processors were common, and plotting speed with floating-point hardware was dramatically faster than software rounded rational arithmetic.
- Therefore we implemented a muLisp function that created a contiguous reverse Polish version of a Derive expression, together with a C function that could evaluate the reverse Polish using IEEE floating point arithmetic.

Summary

There are many ways to implement computer algebra compactly:

1. Special-purpose methods:
    a) Univariate polynomials and series using dense coefficient arrays.
    b) Finessing computer algebra via evaluation and interpolation.
2. General-purpose methods:
    a) String-based computer algebra.
    b) Contiguous stack-based computer algebra.
    c) Compact linked-data computer algebra.

Final Exam:

What is wrong

with this picture?