52-NOTES

Volume 3 Number 9 48/39/38 September 1978

Newsletter of the SR-52 Users Club
published at
9459 Taylorsville Road
Dayton, OH 45424

ROOTS OF POLYNOMIALS

For some months now, Bill Skillman (710) has been trying various approaches to developing a general purpose root-finder for second, third, and fourth degree polynomials (quadratics, cubics, and quartics, respectively). Successive new approaches have been motivated by either Bill or my finding special cases for which earlier approaches wouldn't work. Bill's latest version (July 16th) appears to work for all of our test cases, and is listed below as a "strawman" for members to test and/or improve upon.

Root-finding is an important topic in mathematics and engineering, but one loaded with pitfalls which an understanding of some of the key mathematical concepts can help to avoid. The tutorial which follows addresses the topic of quadratic roots, and is an attempt to air some of the more important concepts vis-a-vis PPC application. It is intended to be a compromise between pedagogical rigor and grade school basics, so as to be useful to most members. I invite members with equation theory expertise to correct errors, share comments, and/or contribute sequels which extend the discussion to higher order polynomials.

The roots of any function in one or more variables are the values given the variable(s) for which evaluation of the function produces zero. ML-08 is a sort of general purpose, brute-force approach for functions in one real variable, requiring either a good initial guess bracketting each root, or lots of iterating to get some modicum of accuracy. Only real roots are found, and there is no guarantee that none is missed.

Finding the roots of polynomials is a special case, for which more efficient methods can be used, provided the user is aware of the potentially large field of pitfalls. Generally, the lower the degree of the polynomial, the easier it is to devise a closed (non-iterative), unconditional solution. An nth degree polynomial has exactly n roots, but may appear to have fewer when some are identical. All or some of the roots may be complex (grammatically "compound", since each complex number is composed of 2 parts: real and imaginary, the latter being the coefficient of $\sqrt{-1}$, called i by mathematicians, and j by electrical engineers).

Zero and first degree polynomial roots are sort of trivial special cases: Ax^0 is just the constant A, and has a "root" only if A=0; the first degree: $Ax + B$ has the single root $x=-B\div A$, $A\neq 0$. (If A=0, the given polynomial reduces to degree zero).

The SR-52 Users Club is a non-profit loosely organised group of TI PPC owners/users who wish to get more out of their machines by exchanging ideas. Activity centers on a monthly newsletter, 52-NOTES edited and published by Richard C Vanderburgh in Dayton, Ohio. The SR-52 Users Club is neither sponsored nor officially sanctioned by Texas Instruments, Inc. Membership is open to any interested person: $6.00 includes six future issues of 52-NOTES; back issues start June 1976 @ $1.00 each.

Efficient ways of finding the roots of higher order polynomials generally involve clever rearrangement of the given terms, and the addition of new ones, to reduce the degree, step by step. The familiar quadratic formula derives from first dividing both sides of: $Ax^2 + Bx + C = 0$ by A, and rearranging to $x^2 + Bx/A = -C/A$, then adding $B^2/4A^2$ to both sides (to complete the square on the left) producing $(x+B/2A)^2 = B^2/4A^2-C/A$, which rearranges to $x=(-B\pm(B^2-4AC)^{\frac{1}{2}})/2A$ after taking the square root of both sides. The roots of a quadratic are either both real or both complex, depending on the sign of the B^2-4AC term, known as the discriminant (d). When d is negative, $x1=-B/2A+i(Abs(B^2-4AC))^{\frac{1}{2}})/2A$; $x2=-B/2A-i(Abs(B^2-4AC))^{\frac{1}{2}})/2A$, or in simplified complex number notation: $x=a\pm ib$, known as a complex conjugate pair. A straightforward approach to mechanization of the quadratic formula with PPCs would be to use one type of processing when d is negative, and another when it is not. At best, this wastes time testing, and code for separate paths, compared with a single method which handles both cases, if such can be found. It turns out that the polar/rectangular functions are the key to a single method, as the V2N2p2 routines show. Here with x (Reg0 or t) set to d and y (the display) set to zero, the R-P function yields r (Reg0 or t)= Abs d and theta (display) = 0 if d is positive; -180 or 180 if d is negative (degree mode). Dividing theta by 2, taking $\sqrt{\text{Abs d}}$, and converting back via P-R produces $y=\sqrt{\text{Abs d}}$ with x=0 if d is negative, and y=0 with $x=\sqrt{\text{Abs d}}$ for d not negative, providing a handy way to present either 2 real roots, or a complex conjugate pair with the same processing.

But just having an efficient routine won't always guarantee good results. When working with real numbers it is important to keep an eye out for critical data dependencies. For quadratics, when the 4AC term is small (but not zero) compared to B^2, $\sqrt{d}$ can easily be indistinguishable from B, even though 4AC is measurable, resulting in the smaller root miscalculated to be zero. The best approach to detecting this problem depends on whether the user expects to catch troublesome inputs by eye, or expects the machine to flag potential trouble. Solving the problem depends on by how much B^2 and 4AC can be expected to differ in relation to machine precision, among other things, and members are invited to suggest viable approaches.

TI-59/(PC-100A) Program: Quadratic, Cubic, Quartic Root-Finder

User Instructions: Bill Skillman (710)

For roots of $Ax^2+Bx+C=0$; Key i, press i; repeat for i=A,B,C; key A'; for roots of $Ax^3+Bx^2+Cx+D=0$; Key i, press i; repeat for i=A,B,C,D;press B'; for roots of $Ax^4+Bx^3+Cx^2+Dx+E=0$: Key i, press i; repeat for i=A, B,C,D,E; press C'. Without printer connection: Press R/S following each output root; imaginary parts of complex roots are flashed. With printer connection: Real roots and real parts of complex roots are printed and tagged with R; imaginary parts are tagged with I.

Program Listing:

```
000: GTO 620 LE' SBR623 rtn LD' 1 Excl 1/x Prd2 Prd3 Prd4 rtn LB' D'
025: 35 Op4 R2 ÷ 3 = S8 x² ± + R3 ÷ xXt 3 = S10 R4 + R8 X (x² X 2 - 0
059: xXt = S11 x² + R10 X x² X 4 = INV xGEt 147 √x + xXt R11 = ÷ 2 ±
085: + SBR126 xXt = SBR126 + xXt - R8 + iflg2 142 E' R8 = ÷ 2 ± +
112: xXt = X 3 √x = xXt - R8 = GT0615 (S13 Op10 Excl3 Absx INV yˣ 3 X
138: R13) rtn 0 = S21 rtn R10 ± √x X xXt 2 = S12 R11 ÷ 2 ÷ xXt yˣ 3 =
166: ± Rad INV Cos ÷ 3 = S9 SBR196 S21 SBR187 S22 2 X 2 X π ÷ 3 + R9
195: = cos X R12 - R8 = S23 iflg2 639 GTO E' LA xXt 4 Op17 CMs xXt
```

```
220:  S1 xXt 13 RST LB S2 xXt 14 RST LC S3 xXt 15 RST LD S4 xXt 16 RST
250:  LE S5 xXt 17 RST LA' D' 35 Op4 R2 ÷ 2 = ± S6 S7 x² - R3 = CP xGEt
282:  296 ± √x xXt R6 Iff1g4 516 GTO 615 √x SUM6 INV SUM7 R6 Iff1g4 491
308:  E' R7 GTO E' LC' R2 CP x=t 469 D' Prd5 R2 X R4 - 4 X R5 = Exc3
337:  S15 X ± Exc2 S14 4 - R14 x² = X R5 - R4 x² = Exc4 S16 Stf1g2 20
367:  S0 SBR025 INV Stf1g2 CP CLR Op20 R*0 + R14 x² ÷ 4 - R15 = SBR581
394:  INV xGEt 376 √x S19 ± + R14 ÷ 2 INV Prd*0 = S2 R14 X R*0 - R16 =
422:  Op10 X (R*0 x² - R5) SBR581 INV xGEt 376 √x = S20 ± + R*0 = S3
451:  SBR261 R19 SUM2 SUM2 R20 SUM3 SUM3 GTO 261 R4 INV x=t 321 R3 x=t
478:  540 R5 Exc3 S2 Stf1g4 GTO 260 SBR496 R7 CP xGEt 511 ± √x xXt SBR
504:  616 ± xXt GTO 616 √x E' ± GTO E' xXt INV P/R ÷ 2 = xXt √x xXt P/R
526:  xXt S6 SBR609 xXt 1 ± Prd6 GTO 609 R5 ÷ R1 = xGEt 569 ± √x √x xXt
553:  35 SBR620 ± E' ± xXt SBR616 ± GTO E' Nop ÷ 4 = √x √x X xXt 1 =
578:  GTO 527 Fix4 EE INV EE INV Fix rtn
609:  35 Op4 R6 E' Stf1g3 24 Op4 xXt Op6 Op8 INV Iff1g3 637 Op69 INV
635:  Stf1g3 R/S CE rtn      (Write partition: 639.39; record: 479.59)
```

- - - - - - - - - - - - -

CALENDAR PRINTER COMPETITION (continued)

As reported last month, Panos has pulled ahead of Lou (whose V3N7p4 program is his best so far), averaging 2 min 38.6 sec per year over the 1972-1976 5-year span with the program listed below. Panos takes advantage of Lou's "2-trick" (using 2 to both supply the integer part of HIR-formatted printcode, and a required pointer increment), and worked out a clever way to combine calculations for the number of Saturdays in a month with determining the start day of the next month: With Sun=0, Mon=1, ...Sat=6 convention (calling Pgm 20 with Reg2 set to zero does the trick), the sum of the start weekday and the number of days in the month, divided by 7 gives the number of Saturdays as the quotient, and the start day of the next month as the remainder. He also speeds up last-line processing by reverse-order printcode packing, proceeding backwards from the next month start day.

My only contribution to Panos' program was to add the Lbl C entry point to provide the option of beginning with a specified month. I expect that any attempt to provide the more restrictive option of printing only one specified month, without duplicating a lot of the processing code would slow down multiple month execution unacceptably. The only labels used are B and C, and since neither is referenced by the program, they can be easily changed to suit the user's preference.

TI-59/PC Program: Calendar Printer — Panos Galidas (207)/Ed

User Instructions: For 1 whole year: Key year, press B; for 1-9 years: key yyyyy.n, press B (n=number of years); for any number of additional years (N), key N, press R/S after initial printing has halted; to start at a specified month, key year, press C, key month, press R/S, key number of years (optional), press R/S (not optional).

Program Listing:

```
000:  SUM0 R*0 H5 SUM0 R*0 H6 R22 H36 Op20 R*0 SUM0 INV Int H7 R*0 H8
030:  Op5 Dsz4 000 R2 CP x=t 121 SUM0 Op0 xXt 2 x=t 114 5 x=t 095 Op20
057:  6 x=t 087 1 x=t 081 3 x=t 075 R*0 EE GTO 097 R*0 EE GTO 105 R*0
083:  EE GTO 116 R*0 INV SUM0 EE H8 R*0 INV Int H7 Op30 R*0 INV SUM0 +
109:  R22 = H6 R*0 H5 Op5 CLR Adv R1 xXt 19 x=t 309 Op21 29 S0 R*1
138:  Int S7 + R2 - (CE ÷ 7 ) Int S4 X 7 = xXt R*1 H8 R3 Op6 R24 H5 R25
171:  H6 R26 H7 R27 H8 Op5 Op0 xXt Exc2 xXt 5 x=t 026 3 x=t 018 2 x=t
```

```
199:  008 0 x=t 002 Op30 6 x=t 026 4 x=t 018 GTO 002 LB S5 Int S3 INV
226:  SUM5 10 Prd05 1 S01 0 S2 Pgm20 SBR 086 - (CE ÷ 7) Int X 7 = S2
255:  Op23 8 S1 R20 S9 R3 ÷ 4 = S4 INV Int CP INV x=t 132 Op29 R4 ÷ 25
285:  = S4 INV Int INV x=t 132 R4 ÷ 4 = INV Int x=t 132 Op39 GTO 132
309:  Dsz5 255 Op23 R3 R/S S5 GTO 257 LC S6 S3 R/S S1 R/S . S05 0 S2
339:  Pgm 20 SBR 086 - (CE ÷ 7) Int X 7 = S2 R6 S03 7 SUM1 GTO 260
```

Prestored Data:

```
08:  31.00000251331 29.00000211714 31.00000301335 30.00000133335
12:  31.00000301345 30.00025413117 31.00025412745 31.00000134122
16:  30.00036173337 31.00000321537 30.00000313242 31.00000161715
20:  28.00000211714 0 98.99 0 1.00003600003 1.00000037 1.0043000037
27:  1.000021000036 2.010000000002 2.010002000003 2.010003000004
31:  2.010004000005 2.010005000006 2.010006000007 2.01000700001
35:  2.010010000011 2.010011000012 2.010012000201 2.010201000202
39:  2.010202000203 2.010203000204 2.010204000205 2.010205000206
43:  2.010206000207 2.01020700021 2.010210000211 2.010211000212
47:  2.010212000301 2.010301000302 2.010302000303 2.010303000304
51:  2.010304000305 2.010305000306 2.010306000307 2.01030700031
55:  2.010310000311 2.010311000312 2.010312000401 2.010401000402
59:  2.010402
```

- - - - - - - - - -

ROMAN NUMERAL PROGRAMS (58/59/PC)

Here is another programming topic which seems to be gathering growing interest, probably not so much because users find many practical applications, but because of the programming challenge. Dix Fulton (83) has a PPX-59 program (which I haven't yet seen) which converts "Arabic" to Roman and vice versa, and Bob Petrie (632) has a program which converts up to fairly large (1-3999999) "Arabic" integers to Roman numerals. I put the word Arabic in quotes since the numerals 0-9 are really of European origin. Bob's program lists below, and follows the modern Roman Numeral conventions: I=1, V=5, X=10, L=50, C=100, D=500, M=1000, $\overline{V}$=5000, $\overline{X}$=10000, $\overline{L}$=50000, $\overline{C}$=100000, $\overline{D}$=500000,and $\overline{M}$=1000000, printing 2 lines in cases where the number of characters exceeds 20. There is quite a bit of "bookkeeping" involved, and Bob's mechanization is worth examining in detail to see how it works. Try bettering Bob's memory requirements, or execution speed, and/or try adding a Roman to Arabic conversion. Perhaps some members will want to consider one or more earlier "Roman" conventions, such as a string of C's followed by a vertical bar and an equal number of backward C's to represent a large power of ten. For example, with a typewriter/PC character approximation: (/) =1000, ((/))=10000, (((/)))=100000, etc. The conventions for forming intermediate numbers have also evolved over the years: IX for 9 succeeded VIIII, and sometimes 19 was written IXX and sometimes XIX. So before you start writing a Roman Numeral program, decide just which applicable conventions you are going to use, and what range of numerical values to allow.

TI-59/PC Program: Roman Numerals — R G Petrie (632)

User Instructions:

Key an integer in the 1-3999999 range, and press A. In from 15 seconds to 2 minutes see the equivalent Roman Numeral representation.

Prestored Data:
```
31:  1 100 10000 1000000 100000000 1.1 11.2 111.3 12.2 2.1 21.2 211.3
43:  2111.4 13.2 24 42 44 27 15 16 30 42.2 44.2 27.2 15.2 16.2 30.2
58:  3732320014 2422000000 4131161721 2431171600
```
Program Listing:
```
000:  LA S30 3 Op17 CMs 6 Op17 CP R30 xXt xGEt 250 4 EE 6 INV EE xXt
024:  xGEt 217 Prt Adv Log Int + 1 = S09 18 S01 24 S02 31 S03 30 S7
052:  SBR230 CP x=t 142 + 35 = S4 R*4 INV Int X 10 = S5 R*4 Int S06 6
081:  S7 SBR230 + 44 + R0 X 2 = S8 R*8 Int X R*3 = SUM*2 R*8 INV Int X
111:  R32 X R*3 = SUM*1 Op23 36 xXt R3 INV x=t 138 31 S3 Op21 Op22 Dsz5
140:  083 Op20 R9 xXt R0 INV x=t 048 Op0 R23 Op3 R22 Op4 Op5 R29 Op3
169:  R28 Op4 Op5 R21 Op1 R20 Op2 R19 Op3 R18 Op4 Op5 R27 Op1 R26 Op2
201:  R25 Op3 R24 Op4 Op5 Adv Adv Adv Adv CLR R/S R58 Op0 Op1 R59 Op2
227:  GTO 209 R*7 ÷ 10 CP + xXt = Int S*7 xXt INV Int X 10 = rtn 7 Op17
253:  Op0 R60 Op1 R61 Op2 GTO 209   (record 4 banks 479.59 partition)
```
- - - - - - - - - - -

TIPS AND MISCELLANY

An Effective Op3mn on 2-Digit Registers (58/59): Maurice Swinnen (779) passes along a tip from the Jan-Apr 78 issue of DISPLAY (in German, due to S Seitz) suggesting a sequence of the form: Dszmn ab where Reg mn is to be decremented, and ab is the code for one of the Ins, Del, or BST pseudos. These pseudos are always skipped during program execution, effectively turning the Dsz mn into Op 3mn, except that zero is the lower decrement limit.

Efficient Number Base Conversions (52,57,58,59): Edward Nilges (972) notes that the usual methods, which examine digit strings digit by digit are unacceptably slow. One way to speed execution of the conversion of base b integers to base ten is to make use of some or all of the user-defined keys, each one corresponding to a number place. The sequence: LA X R1 + R/S LB X R2 + R/S LC X R3 + R/S LD X R4 + R/S LE = R/S will convert up to 5-place integers in base b to base ten, provided b^4,b^3,b^2 and b are stored in Reg 1,2,3,4 respectively. Incidently, in designing such a (or any other) routine for maximum speed, make it as special-purpose as you can. For example, for b=2, don't bother to store 16,8,4 and 2 since the routine will be shorter, and run faster with the in-line: LA X 16 + R/S LB X 8 + ...; for b=8, the cutoff is between 8^3 and 8^2 for choosing between storing and creating the multiplier. And don't design the routine to handle numbers larger or smaller than will ever be input. Fraction conversion can be added by: ...LE + R/S LA' X 1/b + R/S LB' X $1/b^2$ ... etc. Converting from base ten to base b is not so easy to speed up. Ideas, anyone?

Clearing Program Registers (52): Dick Blayney (610) notes that in cases where program memory needs to be cleared, but data preserved, one can write: 000: *S00 Dsz 00, and in RUN mode key: 97 S00 0 rset RUN. Execution halts at step 223 of cleared memory with a flashing zero displayed.

Programming a Variable EE Function (52): Dick wonders if anyone has been able to devise a program sequence which would effectively execute as a manual EE mn. The obvious EE Rab unfortunately doesn't work, and the INV log and y^x functions often produce inexact results. Here is a somewhat clumsy way, using dynamic code generation: 000: LA R97 S96 R02 SUM96 R01 E rtn ... 216: stflg 0 LE EE 0 0 rtn. Run with a call to A, with the desired mantissa in Reg 01 and decapower in Reg 02

in a nOm (for mn) format. For example, 1.2 in Reg01 and 403 in Reg02 produce 1.2 D34. This approach could be used with the 58/59 machines, but with the additional partition/repartition hassle. Anyone have a better idea?

CROM Availability Abroad: Karl Meusch (924) raises the question, and according to TI, as of the end of August, the 9 modules mentioned in V3N8p5 are all being distributed throughout the world, but some to more places than others. If one dealer doesn't have what you want, try another, or write TI.

PC-100B: Karl also notes that a printer labeled PC-100B, interfacing with the 58/59 only is being marketed in Europe. TI confirms this, but had no further details.

Club Program Exchange: Dave reports that he is back in business again (V3N8p6), but that copying now costs 10¢ per page.

58/59 Service Manuals: Indications are that early versions have found their way into the hands of a few members. TI reports that it is now (Sept 7th) accepting orders for separate (one for each machine) manuals at $11.95 each, plus $1.50 handling and postage (Box 53 Lubbock TX 79408 Attn: Parts Order). Topics covered include: circuit diagrams for each of 7 versions (59), clock rates, a memory test program, and card read/write adjustments, according to Bob Petrie (632). However, Bob notes that "...details of internal timing, protocol, etc are expressly not discussed".

More On Printer Spacing (V3N4p6) and Other Behavior: Bob has found that both the vertical and horizontal space buffers around each printed character, as well as the Op0 Op5 and Adv line spaces extend an integral number of dot positions from the 5 X 7 dot character matrix. Horizontal spacing is 2 dots to the right; vertical, 3 dots down. An Op0 Op5 line space spans 13 dots (3 from the bottom of the previous line plus ten), while an Adv line space spans 17. Bob finds each dot to measure .012" on a side, with .0173" vertical and .0157" horizontal average separation between dots. Bob notes an idiosyncrasy which may be common to most printers: feed tends to shift the paper left until it is restrained by the plastic guide. He also notes that a manually keyed Adv on the 59 advances paper continuously so long as the key is depressed. This behavior is common to the 56 and 58, but not the 52, which moves the paper just 17 dot positions, no matter how long the pap key is depressed.

Mag Card Read/Write Error Minimization (59): G Higa (879) reports that initializing the program counter (IAR) to an address outside a bank being recorded minimizes read/write errors. Anyone else?

Blank Display (V3N7p6): Sven Nilsson (1048) reports having measured battery drain on his 58 as 127 ma with 1 displayed, but saving only 1 ma with the blank display (126 ma). But Bob Petrie (632) gets 175 ma with the displayed 1, so we seem to need more measurements to arrive at valid conclusions (comparing the same machines).

Membership Address Changes: 246: 69 Palm Club Pompano Bch, FL 33062; 334: 3383 Knollwood Ct Las Vegas, NV 89121; 981: 5426 Mitchell Dr Dayton, OH 45431.

Correction (V3N7p5): Joel Pitcairn (514) has kindly pointed out that the minus sign (near the end) in the integer extraction routine should be removed.