[VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

15th February, 2024, 00:13

Post: #1

Valentin Albillo
Senior Member

Posts: 1,100
Joined: Feb 2015
Warning Level: 0%

[VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

Hi, **all**,

As it happens, I've been here for **9 years** to the day since I joined back in *2015*, and to commemorate that momentous event (plus today it's ***San Valentin**'s Day*, which also helps,) here you are, a brand new ***SRC #014*** dealing with the *accurate* computation of *NxN* real matrix *determinants* and in particular of those matrices whose elements are all *integer*, where the determinant's value should mandatorily be integer too but frequently *isn't* when computing it using our beloved *HP* calculators' **DET** functionality.

The Problem

HP vintage models such as the **HP-15C**, the **HP-71B** and the *RPL* models compute the determinant of a square matrix via its *LU* decomposition, which is a fast, efficient way to do it but involves *pivoting* and frequently incurs in *rounding errors* which at times can severely degrade the accuracy of the result, even to the point of rendering it *meaningless*, with no correct digits whatsoever, and even if the matrix is non-singular. At best, the determinant of *all-integer* matrices will very frequently be output as a *non-integer* value.

For example,using **MATRIX 9** *(HP-15C)* or **DET** *(HP-71B)* we typically get results like these:

```
      | 0   3   4 |   Det (15C): 1.999999998
 M1 = | 3   1   4 | , Det (71B): 1.99999999995
      | 1   1   2 |   Det Exact: 2

      | 29   18    9 |    Det (15C): -262.0000005
 M2 = | 32  -28  -22 |  , Det (71B): -262.000000023
      | 18   25   15 |    Det Exact: -262

      | -19   41   22    7 |    Det (15C): -2383.999891
 M3 = |   5   19  -14    0 |  , Det (71B): -2383.99999934
      | -36   16    9   26 |    Det Exact: -2384
      |  42  -38   14  -38 |
```

and worst cases include my own ***Albillo matrices*** featured in the *PDF* article HP Article VA016 - Mean Matrices, such as for instance:

```
        | 58   71   67   36   35   19   60 |
        | 50   71   71   56   45   20   52 |
        | 64   40   84   50   51   43   69 |   Det (15C): 1.080204421
 AM#1 = | 31   28   41   54   31   18   33 | , Det (71B): 0.97095056196
        | 45   23   46   38   50   43   50 |   Det Exact: 1
        | 41   10   28   17   33   41   46 |
        | 66   72   71   38   40   27   69 |

        | 13   72   57   94   90   92   35 |
        | 40   93   90   99   01   95   66 |
        | 48   91   71   48   93   32   67 |   Det (15C): -2.956799886
 AM#7 = | 07   93   29   02   24   24   07 | , Det (71B): 0.0699243217409
        | 41   84   44   40   82   27   49 |   Det Exact: 1
        | 03   72   06   33   97   34   04 |
        | 43   82   66   43   83   29   61 |
```

The Sleuthing

As stated above, the internal *LU decomposition* and subsequent processing to compute the determinant from it will usually incur in rounding errors, which can accumulate to the point that for difficult (but *non-singular*) matrices the result will be meaningless or having significantly degraded accuracy, frequently outputting integer determinants as non-integers, even for *2x2* matrices, let alone larger ones.

To attempt solving this annoying problem, some vintage *RPL* models had a flag setting that would check if all elements were integer and if so it would round the computed result to an integer value. However, this wasn't foolproof at all and at times this *ad-hoc* rounding would go awry and result in a **±1** error, usually much bigger than just leaving the non-integer value alone. Worse, the user wasn't informed that this rounding had taken place so in the end the cure was worse than the disease.

What to do ? Well, a possible solution would be to use a different algorithm, in particular one which doesn't involve internal arithmetic operations with *real* numbers, most likely to appear as the intermediate result of non-exact divisions (e.g. *1/3*, *293/177*, ...) and in *2005* (19 years ago as of *2024* !) I wrote a program to implement this idea, **MDET**, which you can find featured in this article:

> HP Program VA711 - HP-71B Exact Integer Determinants and Permanents
>
> *"**MDET** uses a recursive general expansion by minors procedure and works for any dimensions from 2x2 upwards, though it's only reasonably efficient for low-order **N** because computation time grows like N*N!. It produces exact integer results for integer matrices (provided there are no intermediate overflows) [...]"*

MDET uses just multiplications and additions/subtractions but *no* divisions in sight so it certainly delivers the goods and will compute *exactly* all determnants above. However, its recursive nature means that using it with matrices larger than, say, *7x7* or *10x10* (depending on the calc's speed) would *really* take too long because the execution time increases *super-exponentially*.

Thus, again, *what to do ?* As usual, the idea is correct but the chosen algorithm isn't optimal, we need to use a much faster one, in particular faster than *O(N*N!)*, and this is my implementation for the **HP-15C** of such an algorithm (the **HP-71B** version is quite straightforward so I'm not including it here.)

The HP-15C Implementation

This small ***50****-step (56 bytes = 8 registers)* program will accurately compute in *polynomial* time (not *super-exponential* time like *MDET*) the *determinant of* an *arbitrary* ***NxN*** real matrix **A** (not necessarily *all-integer*) for ***2*** ≤ ***N*** ≤ ***8*** (depending on available memory.)

It takes no inputs (but the user must have previously dimensioned and populated the real square matrix **A**) and it outputs the computed determinant to the display.

Program listing

```
►LBL A          001- 42,21,11      STO 0          027-    44  0
 RCL DIM A      002- 45,23,11      STO 1          028-    44  1
 STO I          003-    44 25      CLX            029-    43 35
 STO 2          004-    44  2      STO E          030-    44 15
 DSE 2          005- 42, 5, 2      RCL B          031-    45 12
 RCL MATRIX A   006- 45,16,11      CHS            032-       16
 STO MATRIX B   007- 44,16,12      DSE 0          033- 42, 5, 0
►LBL 2          008- 42,21, 2     ►LBL 3          034- 42,21, 3
 RCL MATRIX B   009- 45,16,12      RCL 0          035-    45  0
 STO MATRIX E   010- 44,16,15      STO 1          036-    44  1
 RCL I          011-    45 25      X<>Y           037-       34
 STO 0          012-    44  0      STO E          038-    44 15
►LBL 0          013- 42,21, 0      RCL- B         039- 45,30,12
 STO 1          014-    44  1      DSE 0          040- 42, 5, 0
 DSE 1          015- 42, 5, 1      GTO 3 ►        041-    22  3
 CLX            016-    43 35      RCL MATRIX E   042- 45,16,15
►LBL 1          017- 42,21, 1      RCL MATRIX A   043- 45,16,11
 STO E          018-    44 15      RESULT B       044- 42,26,12
 DSE 1          019- 42, 5, 1       x             045-       20
 GTO 1 ►        020-    22  1      CHS            046-       16
 DSE 0          021- 42, 5, 0      DSE 2          047- 42, 5, 2
  1             022-        1      GTO 2 ►        048-    22  2
 RCL 0          023-    45  0      MATRIX 1       049- 42,16, 1
 TEST 6         024- 43,30, 6      RCL B          050-    45 12
 GTO 0 ►        025-    22  0
 RCL I          026-    45 25
```

Notes:

- This stand-alone program works for *NxN* matrices (*2x2* ≤ *N* ≤ *8x8*, depending on the **HP-15** physical/virtual model's available memory,) and runs in polynomial time.
- It uses matrices **A** (the *input*) and auxiliary matrices **B** and **E**, all of them *NxN*, as well as registers $\mathbf{R_0}$-$\mathbf{R_2}$, $\mathbf{R_I}$ and labels **A**, **0-3**, but no subroutines or flags.
- The input matrix **A** is *not* affected by the computation and so remains unaltered and available for further use without having to re-input it or restore its elements back.

- The program uses the *end of program memory* to end execution. To call it instead from another program as a *subroutine*, add a **RTN** instruction at its very end (**RTN** *051- 43 32*). It will then return to the calling program with the determinant's value in the *X* stack register.

- The required available memory to compute an *NxN* determinant is ***3*N²+9*** registers (program itself included,) so a vintage physical **HP-15C/64** can do matrices up to *4x4*, the **CE/192** can do up to *7x7* and the **DM15/M1B** can do up to *8x8*.

Worked examples

Although all examples below deal with all-integer matrices, the program of course works for arbitrary matrices with *non-integer* elements and produces their determinants with improved accuracy as well. Assume we're using an **HP-15C CE** in *192-register* mode throughout.

Example 1. Accurately compute the determinant of the following ***4x4*** *all-integer* matrix and compare the result with the determinant computed using the buil-in instruction (**MATRIX 9**):

```
       | -19  41  22   7 |
M3 =   |   5  19 -14   0 |
       | -36  16   9  26 |
       |  42 -38  14 -38 |
```

- **Initialize**: allocate memory for register $\mathbf{R_2}$ and clear all matrices to *0x0*:

```
2, DIM (i), MATRIX 0
```
(MEM: 02 183 08-0)

- **Dimension** and **populate** the input matrix:

```
4, ENTER, DIM A, USER, MATRIX 1,

-19, STO A,  41, STO A,  22, STO A,   7, STO A,
  5, STO A,  19, STO A, -14, STO A,   0, STO A,
-36, STO A,  16, STO A,   9, STO A,  26, STO A,
 42, STO A, -38, STO A,  14, STO A, -38, STO A
```

- **Compute** its determinant:

```
FIX 6, GSB A  ->  -2,384.000000
```

- **Compare** the result with the one produced by the *built-in* function (**MATRIX 9**):
*(no need to re-input the matrix as it's left **unaltered** by the program)*

```
RCL MATRIX A, MATRIX 9  ->  -2383.999891
```

As you can see, the built-in instruction returns a *non-integer* value with degraded accuracy in the last *three* places, while the program returns the *exact integer* value.

Example 2. Ditto for my ***7x7*** matric **AM#1**:

```
        | 50  71  71  56  45  20  52 |
        | 64  40  84  50  51  43  69 |
AM#1 =  | 31  28  41  54  31  18  33 |
        | 45  23  46  38  50  43  50 |
        | 41  10  28  17  33  41  46 |
        | 66  72  71  38  40  27  69 |
```

- **Initialize**: allocate memory for register $\mathbf{R_2}$ and clear all matrices to *0x0*:

```
2, DIM (i), MATRIX 0
```
(MEM: 02 183 08-0)

- **Dimension** and **populate** the input matrix:

```
7, ENTER, DIM A, USER, MATRIX 1,

50, STO A, ...,  69, STO A
```

- **Compute** its determinant:

```
FIX 9, GSB A  ->  1.000000000
```

- **Compare** the result with the one produced by the *built-in* function (**MATRIX 9**):

```
RCL MATRIX A, MATRIX 9  ->  1.080204421
```

This time the built-in instruction returns a *non-integer* value with *very severely* degraded accuracy in the last *eight* places, while the program again returns the *exact integer* value.

Example 3. Last but not least, ditto for my ***7x7*** matric **AM#7**:

```
          | 13 72 57 94 90 92 35 |
          | 40 93 90 99 01 95 66 |
          | 48 91 71 48 93 32 67 |
AM #7 =   | 07 93 29 02 24 24 07 |
          | 41 84 44 40 82 27 49 |
          | 03 72 06 33 97 34 04 |
          | 43 82 66 43 83 29 61 |
```

- **Initialize**: allocate memory for register $\mathbf{R_2}$ and clear all matrices to *0x0*:

```
2, DIM (i), MATRIX 0        (MEM: 02 183 08-0)
```

- **Dimension** and **populate** the input matrix:

```
7, ENTER, DIM A, USER, MATRIX 1,

13, STO A, ...,  61, STO A
```

- **Compute** its determinant:

```
FIX 9, GSB A  ->  1.000000000
```

- **Compare** the result with the one produced by the *built-in* function (**MATRIX 9**):

```
RCL MATRIX A, MATRIX 9  ->  -2.956799886
```

And once more the built-in instruction returns a *non-integer* value with accuracy so degraded that *it loses all 10 digits* (and even the *sign* is wrong !) while the program returns the *exact integer* value once more.

That'll be all, *Over and Out.*

V.

WWW FIND EDIT QUOTE REPORT

15th February, 2024, 10:11 (This post was last modified: 15th February, 2024 10:11 by EdS2.) **Post: #2**

EdS2
Senior Member

Posts: 582
Joined: Apr 2014

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

Splendid! Thank you - and happy anniversary!

EMAIL PM FIND QUOTE REPORT

17th February, 2024, 11:43 (This post was last modified: 17th February, 2024 18:59 by J-F Garnier.) **Post: #3**

J-F Garnier
Senior Member

Posts: 940
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

Another great piece of 15c code from Valentin, and nice to come accross the famous AM matrices again !
Moreover, it's a new example of what can be done with the extended memory versions of the classic 15c.

Now, the challenge for us poor readers is to understand and possibly to identify the algorithm.
Well, I took it as my challenge...

> **Valentin Albillo Wrote:** (15th February, 2024 00:13)
>
> ... this is my implementation for the **HP-15C** of such an algorithm (the **HP-71B** version is quite straightforward so I'm not including it here.)

And here we come to the readability of RPN code...
I had to first translate the code to a high level language (actually the HP-71B BASIC) to be able to understand it.

BTW, would RPN/RPL fans be able to port Valentin's code to other Classic machines such as the 41C w/ Advantage pack or the 42S, or even the 28/48 Series, in a way to get a working *and readable* version, with just the 15c code as a guide? Just curious...

The algorithm (as far as I understand it) is really interesting, not only it provides an integer result for integer input matrices (within certain limits of course), but it doesn't use any division at all, and so doesn't have to choose a non-zero pivot. This simplifies the code a lot.
The drawback is that it uses 2 auxiliary matrices, which is a real limitation on small machines (such as the original 15C), contrary to other methods such as the Bareiss algorithm.

Thanks, Valentin, for this SRC !

J-F

EMAIL PM WWW FIND QUOTE REPORT

18th February, 2024, 02:41 **Post: #4**

Gene
Moderator

Posts: 1,370
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

Bravo, Valentin!

EMAIL PM FIND QUOTE REPORT

20th February, 2024, 17:43 (This post was last modified: 20th February, 2024 18:40 by J-F Garnier.) **Post: #5**

J-F Garnier
Senior Member

Posts: 940
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

I spent more time to play with this program and have a few more comments on Valentin's SRC #014:

This algorithm is fast, it is much faster than the expansion by minor (illustrated by the XDET program from Valentin), and its speed is actually of the same order of magnitude as the LU-decomposition method used by the HP standard DET functions. Of course, in this implementation, a part of the processing is done as user code so not as fast as it could be in microcode.

However, a characteristic of this algorithm and others is that the values of intermediate iterative calculations grow exponentially in magnitude.
We should remember that the exact representation of integer numbers is quite limited on our Classic machines: 1E10 for the 15c , 1E12 for the 71B (resp. 1E13 and 1E15 in internal calculations).

This SRC is a great illustration of a recent algorithm, but it works on the 15c (or the 71B) only if the input matrix elements are small enough to avoid any integer overflow in the intermediate calculations.
As a rule of the thumb, the elements of a 7x7 or 8x8 matrix should not have more than 2 digits. I found examples with numbers of 3 digits that still work well, but others don't.

For instance:

```
| 274 213 400 322 341 308 446 |
| 202 210 383 295 360 295 450 |
| 154 175 332 175 322 361 413 |
| 176 147 265 272 328 277 351 |
| 111 131 249 182 324 296 340 |
| 155 179 329 218 381 399 444 |
| 147 127 229 174 245 258 297 |
```

The exact determinant of this matrix is 1, as can be checked on a symbolic system, or just Free42.
Valentin's 15c program indeed gives exactly 1 (the 15c built-in DET gives -571)

On the contrary, with this matrix:

```
| 477 389 402 515 358 409 289 |
| 302 273 282 322 280 283 205 |
| 278 231 339 319 343 254 214 |
| 432 360 406 502 391 359 319 |
| 475 316 509 649 543 393 288 |
| 299 304 351 369 459 346 221 |
| 526 561 442 441 371 491 445 |
```

Valentin's 15c program gives 7269230 instead of 1 (the buit-in DET gives a hardly better -1337 value, with the wrong sign)

This SRC achieved its goal, as far as I'm concerned: learn new (even recent) methods for computing determinants, such as the Bareiss and Bird algorithms, and understand (a little bit) how they are working as well as their limitations on our machines.

J-F

EMAIL PM WWW FIND QUOTE REPORT

26th February, 2024, 12:27 **Post: #6**

59:59:59 **Werner**
Senior Member

Posts: 870
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

The Bird algorithm (thanks, J-F) implemented for the 42S.
I made two changes:
- negate the input matrix A once
- shrink the matrix B (remove a row) at every step, as the last row is always filled with zeroes.
O, and stack-only. Of course.

```
00 { 79-Byte Prgm }
01▸LBL "BIRD"
02 ENTER
03 +/-
@
04▸LBL 03
05 X<>Y
06 DIM?
07 X<> ST L @    [B]      I        -[A]
08 EDIT
09 CLX
10 SIGN
11 X=Y? @ quit when B is a single row
12 GTO 00
13 X<>Y
14 ENTER @      I        I        1        -[A]
15 STOIJ
16 R^
17 RCLEL
18 +/- @        SUM      -[A]
19 J-
20 1
21 ENTER
@               X        Y        Z        T        I,J
22▸LBL 02 @      1        N-I      SUM      -[A]     I,I-1
23 NEWMAT @     [0]      SUM      -[A]     -[A]
24 PUTM
25 I-
26 RCLEL
27 +/-
28 X<> ST Z
29 STOEL
30 STO+ ST Z
31 R↓
32 DIM?
33 STO+ ST Y
34 J-
35 FC? 76
36 GTO 02
@
37 DELR
38 EXITALL
39 R^
40 STO× ST Y
41 GTO 03
@
42▸LBL 00
43 RCLEL
44 ENTER
45 EXITALL
46 R↓
47 END
```

Cheers, Werner

27th February, 2024, 12:35 (This post was last modified: 27th February, 2024 16:16 by Werner.) **Post: #7**

59:59:59 **Werner**
Senior Member

Posts: 870
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

.. and, my 15C version.
- works for n=1, too
- twice as fast
- uses 2n fewer matrix elements

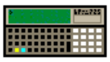
- uses R0 and R1 only (not I)
- 45 bytes instead of 57

```
001 LBL B
002 RCL MATRIX A
003 STO MATRIX B
004 RESULT B

005 LBL 3
006 RCL DIM B           -- B is ixn
007 X<>Y
008 STO 0
009 STO 1
010 RCL B               -- recall Bii
011 DSE 0               -- i := i-1;
012 ISG 1               -- skip
013 RTN                 -- if B is single row, we're done
014 CHS                 -- sum := -Bii;
015 RCL 0
016 R^
017 DIM B               -- remove last row, which will be zeroes anyway
018 RCL MATRIX B
019 STO MATRIX E
020 R^                  -- put sum back into stack reg. X
021 GTO 0

022 LBL 2
023 0
024 LBL 1               -- zero out row i of E, j=i-1..1
025 STO E
026 DSE 1
027 GTO 1
028 X<>Y
029 DSE 0

030 LBL 0
031 RCL 0
032 STO 1
033 X<>Y
034 STO E               -- Eii := sum;
035 RCL- B              -- sum := sum - Bii;
036 DSE 1
037 GTO 2
038 RCL MATRIX E
039 RCL MATRIX A
040 x
041 CHS
042 GTO 3
```

Cheers, Werner

EMAIL PM FIND QUOTE REPORT

28th February, 2024, 09:52 **Post: #8**



J-F Garnier
Senior Member

Posts: 940
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

> **Werner Wrote:** (27th February, 2024 12:35)
>
> .. and, my 15C version.
> - works for n=1, too
> - twice as fast
> - **uses 2n fewer matrix elements**
> - uses R0 and R1 only (not I)
> - 45 bytes instead of 57

Great, so we can now handle 8x8 matrices on the 15c CE/192 !

Let's try with this matrix I just created (determinant is exactly 1):

```
| 75 43 72 59 37 63 51 67 |
| 67 34 64 45 36 55 38 62 |
| 85 53 87 74 59 70 67 81 |
| 87 49 91 70 50 80 65 86 |
| 72 35 73 53 37 69 52 68 |
```

```
| 65 38 71 62 38 68 62 65 |
| 86 56 86 77 56 69 67 80 |
| 82 58 100 82 63 79 87 88 |
```

8 , ENTER , DIM A
enter elements into matrix A ... (a bit long)

Check the matrix with:
RCL MATRIX A , RESULT B , MATRIX 9 (det) -> .9999585581
(the RESULT B step is important, otherwise the matrix A is no more useable)

Now:
GSB B --> 1 exactly !

J-F

EMAIL PM WWW FIND QUOTE REPORT

1st March, 2024, 00:51 **Post: #9**

Valentin Albillo
Senior Member

Posts: 1,100
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

Hi, all,

Well, today it's ***February, 29th*** so 15 days have elapsed since I posted my *OP* and it's high time to add some final comments.

First of all, thanks to all of you for your interest and most especially to those of you who contributed with code and various improvements, namely **J-F Garnier** and **Werner**. Frankly, I expected to get more posts since the subject matter is both interesting, *instructive* and useful, and the **HP-15C CE** is still very much fashionable but alas, at a meager 7 posts and 1,000 views in 15 days it wasn't to be. Now I'll comment on some of your messages and various matters:

> **Gene Wrote:**
> Bravo, Valentin!

Thanks a lot for your kind appreciation, **Gene**.

> **J-F Garnier Wrote:**
> Another great piece of 15c code from Valentin, and nice to come accross the famous AM matrices again !

Thank you very much for your appreciation and kind words re my code and difficult matrices.

> **J-F Garnier Wrote:**
> And here we come to the readability of RPN code... I had to first translate the code to a high level language (actually the HP-71B BASIC) to be able to understand it.

You didn't post your **15C/RPN** to **71B/BASIC** program in this thread so I'm posting my original straightforward *HP-71B BASIC* version below.

> **J-F Garnier Wrote:**
> The algorithm (as far as I understand it) is really interesting, not only it provides an integer result for integer input matrices (within certain limits of course), but it doesn't use any division at all, and so doesn't have to choose a non-zero pivot. This simplifies the code a lot.

Indeed. *Bird's algorithm* doesn't use any divisions at all and that's a big plus, completely eliminating having to care about pivots, having to work with floating point values and having to deal with rounding errors.

> **J-F Garnier Wrote:**
> The drawback is that it uses 2 auxiliary matrices, which is a real limitation on small machines (such as the original 15C) [...]

Using two extra matrices is unavoidable, as *Bird's method* essentially relies on matrix multiplication and this usually requires three *different* matrices on the *HP-15C*: **A**=**B**x**C** with distinct matrices **A**, **B**, **C**.

> **J-F Garnier Wrote:**
> This SRC is a great illustration of a recent algorithm, but it works on the 15c (or the 71B) only if the input matrix elements are small enough to avoid any integer overflow in the intermediate calculations.

Yes, the size of intermediate results is a problem for many algorithms, including the naive one which in this case would entail adding 5040 terms (each of them a product of 7 elements,) which are sure to be more than 10-12 digits long if the elements are 3-digit long or more, so *Bird's algorithm* does no worse than others in this regard and usually it does better.

> **J-F Garnier Wrote:**
> This SRC achieved its goal, as far as I'm concerned: learn new (even recent) methods for computing determinants, such as the Bareiss and Bird algorithms, and understand (a little bit) how they are working as well as their limitations on our machines.

That's the idea ! 🙂

> **J-F Garnier Wrote:**
> Thanks, Valentin, for this SRC !

You're welcome, **J-F**, again thanks to you for your continued appreciation.

> **Werner Wrote:**
> The Bird algorithm (thanks, J-F) implemented for the 42S.
> I made two changes: - negate the input matrix A once, - shrink the matrix B (remove a row) at every step, as the last row is always filled with zeroes. O, and stack-only. Of course.

Excellent !

> **Werner Wrote:**
> .. and, my 15C version. - works for n=1, too, - twice as fast, - uses 2n fewer matrix elements, - uses R0 and R1 only (not I), - 45 bytes instead of 57

Most excellent ! The idea of removing a zero row at every step is really clever. Congratulations !

Now for my original *BASIC* program for the **HP-71B**. It is this *4-line*, *191-byte* suprogram called ***BIDET*** (***BI****rd's* ***DET****erminant, pun most definitely intended)*:

```
100  SUB BIDET(A(,),D) @ N=UBND(A,1) @ DIM B(N,N),E(N,N) @ MAT B=A @ FOR K=1 TO N-1 @ MAT E=B
110  FOR I=2 TO N @ FOR J=1 TO I-1 @ E(I,J)=0 @ NEXT J @ NEXT I
120  FOR I=1 TO N @ S=0 @ FOR J=I+1 TO N @ S=S-B(J,J) @ NEXT J @ E(I,I)=S @ NEXT I
130  MAT B=E*A @ MAT B=-B @ NEXT K @ D=B(1,1)
```

Let's try it with **J-F**'s second *7x7* matrix, namely:

```
| 477 389 402 515 358 409 289 |
| 302 273 282 322 280 283 205 |
| 278 231 339 319 343 254 214 |
| 432 360 406 502 391 359 319 |
| 475 316 509 649 543 393 288 |
| 299 304 351 369 459 346 221 |
| 526 561 442 441 371 491 445 |
```

Directly from the command line, first execute the following initialization:

```
>DESTROY ALL @ OPTION BASE 1 @ DIM A(7,7),D @ MAT INPUT A
```

(... enter all 49 elements ...)

and then let's compute the determinant while also gauging the *exactness* of the results:

```
>CFLAG INX @ CALL BIDET(A,D) @ D,FLAG(INX)

     1              0
```

so the determinant is computed as **1** and the ***IN****e****X****act* flag is ***0*** *(false)* so the value is ***exact***.

Now let's compare with the assembly-language **DET** function:

```
>CFLAG INX @ DET(A),FLAG(INX)

    -53.8832026193   1
```

and this time the ***IN****e****X****act* flag is ***1*** *(true)* so the computed value is truly ***inexact***. A whole lot !

As for my **HP-15C** program returning *7269230* instead of **1** (**J-F** dixit), it can possibly be made to return the exact value by combining it with *partitioned matrix* techniques, as decribed in my article.

We could try partitions into 4 blocks (from *1x1*... to *6x6*...) but perhaps the combo *4x4, 4x3, 3x4, 3x3* would possibly avoid *intermediate overflows* and result in the exact value being produced. This is an area worthy of research for those interested.

Last but not least, the same way that **J-F** translated my *15C/RPN* program to *71B/BASIC*, the reverse translation is easy to perform manually line by line and results in translating my *71/BASIC* subprogram into my exact *15C/RPN* original program, like this:

```
MAT B=A @ FOR K=1 TO N-1 @ MAT E=B

 ►LBL A
  RCL DIM A       N
  STO I           RI=N
  STO 2
  DSE 2           K=N-1
  RCL MATRIX A
  STO MATRIX B    MAT B=A
 ►LBL 2           FOR K loop
  RCL MATRIX B
  STO MATRIX E    MAT E=B

FOR I=2 TO N @ FOR J=1 TO I-1 @ E(I,J)=0 @ NEXT J @ NEXT I

  RCL I           N
  STO 0           I=N
 ►LBL 0           FOR I loop
  STO 1
  DSE 1           J=I-1
  CLX             0
 ►LBL 1           FOR J loop
  STO E           E(I,J)=0
  DSE 1
  GTO 1►          NEXT J
  DSE 0
   1
  RCL 0
  TEST 6 (X#Y?)
  GTO 0►          NEXT I

S=-B(N,N) @ E(N,N)=0 @ FOR I=N-1 TO 1 STEP -1 @ E(I,I)=S @ S=S-B(I,I) @ NEXT I

  RCL I           N
  STO 0           I=N
  STO 1           J=N
  CLX             0
  STO E           E(N,N)=0
  RCL B           B(N,N)
  CHS             S=-B(N,N)
  DSE 0           I=N-1
 ►LBL 3           FOR I loop
  RCL 0           I
  STO 1           J=I
  X<>Y            S
  STO E           E(I,I)=S
  RCL- B          S=S-B(I,I)
  DSE 0
  GTO 3►          NEXT I

MAT B=E*A @ MAT B=-B @ NEXT K @ D=B(1,1)

  RCL MATRIX E
  RCL MATRIX A
  RESULT B
   x              MAT B=E*A
  CHS             MAT B=-B
  DSE 2
  GTO 2►          NEXT K
  MATRIX 1        I=1, J=1
  RCL B           B(1,1)
```

Easy, right ?

That's all for now, best regards.

V.

1st March, 2024, 21:57 (This post was last modified: 4th March, 2024 09:18 by Werner.) **Post: #10**

Werner
Senior Member

Posts: 870
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

I wasn't done yet ;-)
Here is a low memory version of the Bird algorithm.
For a problem of order n it needs only **n^2 + 3*n** matrix elements.
It uses RI, and R0-R2 so requires at least 2 DIM (i).
Drawbacks:
- it is slower than my previous version (yet, for AM7, still faster than Valentin's original).
- it needs yet another matrix, D
- it's a bit larger (80 bytes = 12 registers)
- the code is somewhat more involved ;-)

```
001 LBL D
002 CLX
003 STO 0
004 DIM D       -- erase D and E
005 DIM E
006 RCL DIM A
007 STO 1
008 STO I
009 1
010 X<>Y
011 DIM D       -- D and E 1xn
012 DIM E
013 RCL DIM A
014 RCL g A
015 DSE I       -- i := n - 1;
016 ISG 0       -- R0=1 and skip
017 RTN         -- quick return when n=1
018 STO- D      -- d(n) := -a(n,n)
019 RESULT B

020 LBL 1       -- i=n-1 to 1
021 RCL DIM A
022 STO 1
023 RCL D       -- x := d(n);
024 RCL I
025 ENTER
026 RCL g A
027 STO- D      -- d(n) := d(n) - a(i,i);
028 X<>Y
029 RCL I
030 STO 1       -- k := i;
031 X<>Y
032uSTO E       -- e(k) := x; k := k + 1;
033 LBL 8       -- e(k) := a(i,k); k=i+1..n
034 RCL I
035 RCL 1
036 RCL g A
037uSTO E
038 GTO 8

039 RCL DIM A
040 RCL- I
041 STO 2

042 LBL 2       -- for nj=n-i to i step -1
043 RCL MATRIX E
044 RCL MATRIX A
045 x
046 CHS
047 RCL I
048 RCL+ 2
049 STO 1
050 DSE 1       -- j := nj + i - 1;
051 RCL D       -- d(j)
052 1
053 RCL I
```

```
054 RCL g B     -- b(i)
055 STO- D      -- d(j) := d(j) - b(i);
056 X<>Y
057 RCL I
058 STO 1       -- k := i;
059 X<>Y
060uSTO E       -- e(k) := d(j); k := k + 1;
061 LBL 9       -- e(k) := b(k); k=i+1..n
062 RCL B
063uSTO E
064 GTO 9
065 DSE 2
066 GTO 2

067 DSE I
068 GTO 1

069 RCL B
070 RTN
```

Hope you like it,
Werner

EMAIL PM FIND QUOTE REPORT

6th March, 2024, 20:18 (This post was last modified: 6th March, 2024 20:22 by Werner.) **Post: #11**

Werner
Senior Member

Posts: 870
Joined: Dec 2013

RE: [VA] SRC #014 - HP-15C & clones: Accurate NxN Determinants

Shorter, and a bit slower (still marginally faster than Valentin's original)

- 50 lines, 56 bytes (8 registers) (including RTN ;-)
- a 10x10 matrix takes only about 5 seconds on the 15CE
- registers used I012, so needs 2 DIM (i) setting at least
- order n needs n^2+3*n matrix elements
- total memory needed, not counting I01: **n^2+3*n+9**

max. orders possible:

- 15C/64: 6x6
- 15CE/192: 12x12
- DM15L/229: 13x13

```
#  Program produced by JRPN 15C. (https://jrpn.jovial.com/run15/index.html)
#  Generated 2024-3-6 11:35 Central European Standard Time.

   000 {          }
   001 { 42 21 14 } f LBL D
   002 {        1 } 1
   003 {    44  0 } STO 0
   004 { 45 23 11 } RCL DIM A
   005 {    44 25 } STO I
   006 {    43 35 } g CLx
   007 { 42 23 14 } f DIM D     -- erase D
   008 {       33 } Rv
   009 { 42 23 14 } f DIM D     -- D,E 1xn
   010 { 42 23 15 } f DIM E
   011 { 42 26 12 } f RESULT B
   012 { 42 21  1 } f LBL 1     -- for i=n to 1 step -1
   013 { 45 23 11 } RCL DIM A
   014 {       16 } CHS
   015 {    45 25 } RCL I
   016 {    44  1 } STO 1
   017 {       40 } +
   018 {    44  2 } STO 2       -- nj := -(n-i);
   019 {    43 35 } g CLx
   020 { 44 16 15 } STO MATRIX E
   021 { 42  5 15 } f DSE E     -- E := -ei;
   022 { 42 21  2 } f LBL 2     -- for nj=i-n..0
   023 { 45 16 15 } RCL MATRIX E
   024 { 45 16 11 } RCL MATRIX A
   025 {       20 } *
   026 {       16 } CHS         -- B := -E*A;
   027 {    45 25 } RCL I
   028 { 45 30  2 } RCL - 2
```

```
029 {     44  1 } STO 1       -- j := i - nj;
030 {     45 14 } RCL D       -- t := D(j);
031 {         1 } 1
032 {     45 25 } RCL I
033 {  45 43 12 } RCL g B
034 {  44 30 14 } STO - D     -- D(j) := D(j) - B(i);
035 {     43 35 } g CLx
036 {         1 } 1
037 {     45 25 } RCL I
038 {     44  1 } STO 1
039 {  44 43 12 } STO g B     -- B(i) := t;
040 {  42 21  9 } f LBL 9     -- E(k) := B(k); k=i..n
041 {     45 12 } RCL B
042 {   44 15 u } u STO E
043 {     22  9 } GTO 9
044 {  42  6  2 } f ISG 2     -- next nj;
045 {     22  2 } GTO 2
046 {  42  5 25 } f DSE I     -- next i;
047 {     22  1 } GTO 1
048 {     45 14 } RCL D
049 {        16 } CHS
050 {     43 32 } g RTN
```

End.

Cheers, Werner
(that's it, I think. I see no immediate further improvements)
(incidentally, while I used JRPN to produce the above listing, the program does not run on the simulator as-is, as it contains a bug (the simulator, not the program) - insert ENTER just after RCL D in line 030 to make it work on JRPN)