Long Live The HP-34C !

Valentín Albillo (HPCC #1075)

Back then in 1979 Fernando del Rey and I were already seasoned HP calc users, what with me owning two excellent HP machines (an **HP-25** first, shortly followed by an extremely-hard-to-get **HP-67**!) while Fernando bypassed the **HP-25** and went straight for the **HP-67** after his **HP-55**'s "baptism" to RPN and the HP ways (awesome machine, that **HP-55**; a real pity its programmability is second only to the **HP-10C**'s). Then, almost unexpectedly (but not quite) we were blessed with HP releasing our *Dream Machine*, with capitals and all, the **HP-41C**, which kept us with our little hands fully full for very many years to come.

But 1979 also saw another wonderful new machine, part of a brand new series. Lost amid the **HP-41C** brightness it could have easily passed by, largely unnoticed, but it had so incredible a features/price ratio that, far from it, it *was* hugely noticed, to become a large sales success among students and other wannabe HP lovers who could hardly afford the cost of owning an **HP-41C** however much they might want it. A bare-bones **HP-41C** was already far too expensive, at least in our Spanish homeland (twice as pricey as in the US), and you could do very little with it unless you also got an equally expensive card reader and at least one memory module or two. Otherwise you'd find yourself with an awesome instruction set and features but too little RAM to even barely scratch its potential, similarly to what later happened to the **HP-28C**. Unlike that crippled machine, the **HP-41C** could be expanded enormously, but at a hefty price. Too hefty. Enter the **HP-34C**.

A landmark model

At first glance, the **HP-34C** could look disappointing. No LCD display but the old fashioned, battery-hog red LED variety. No alphanumerics. No mass storage. No expandability. It had an interesting, novel design, but in the hand it felt rather flimsy, like you could crush it by merely closing your fist, unlike the rock-solid previous models. But, for the first time, it was a relatively cheap model to the point of being affordable by students and other low-income people which couldn't justify shelling the cash for an **HP-41C**, say. For the first time you could own a real, genuine RPN HP programmable calculator without filing for bankruptcy.

And what a calculator it was ! Despite its looks, you could fathom that it was something by the mere fact that, like the **HP-67**, it had *three* prefix keys, **f**, **g**, and **h** !. Why so many ? Well, the answer was to be found in the huge *293-page* ***Owner's Handbook and Programming Guide***, a wonderful 4-colour affair in the style of the celebrated **HP-67** manual. A cursory reading would inform you that the little, *"fragile"* HP calculator right in front of you could do all the following, and more:

- *Automatic memory allocation*, from 70 steps of program memory and 21 data registers to 210 steps and one register, so that you could optimize for data or program memory usage as needed.
- *Full programmability*, including 6 levels of subroutine nesting, full indirect addressing of data registers, program locations, and modes, plus labels, insertion and deletion of program steps, flags, conditionals, **ISG**/**DSE** loop constructions, etc.
- *Ground-breaking new computational functionalities,* such as numeric **SOLVE** and **INTEGRATE**, plus **Gamma** function, linear regression, and the usual set of trigonometric, logarithmic, and exponential functions, squares, square root, reciprocals, and powers, plus basic statistics and accumulations.
- *Convenience additional functionality*, such as mantissa, indirect exchange of registers with the display contents, exchange of the display contents with the indirect register, plus basic conversions such as polar/rectangular, degrees/radians, and decimal hours/degrees to hours/degrees-min-sec, not to forget the invaluable number-alteration functions like **INT**, **FRAC**, and **ABS**.

To people used to the **HP-25** or even the **HP-67** such as myself, this much power was uncanny, indeed. Most of us felt that it was indeed a pity that the **HP-41C**, despite being vastly superior in most aspects, still sorely lacked the awesome **SOLVE** and **INTEGRATE** functionality, something that wouldn't be truly alleviated till the release of the Advantage ROM, still in the distant future. That being so, it became common to own *both* calculators, the **HP-41C** for sheer power and *élan*, the **HP-34C** as a relatively inexpensive, useful model having highly valuable extra functionality not available in its costlier sibling.

Though numerical root finding and integration were the most noticeable extras, the **HP-34C** did include a number of important additional functions that the **HP-41C** didn't, such as **x!** (Factorials and Gamma function, while **HP-41C**'s **FACT** would only work with positive integer arguments being just a Factorial implementation), and linear regression as well, including the correlation coefficient and estimates. It also featured mantissa viewing, plus an unexpected novelty, a user-inititated self-test that would check that the internal ROM and RAM were in peak shape whenever the user executed the special key sequence **STO ENTER**. As for synthetics, even the modest **HP-34C** did have its share of techniques for synthetic exploration of its innards: by interrupting the **STO ENTER** self-test midway it was possible to leave the program pointer where it usually wouldn't go, then a little ingenuity would allow its mischievous user to discover all sorts of interesting facts.

That said, no matter how powerful the instruction set there were limitations to endure, but the excellent programmability meant that many of them you could 'program away', so to say. **INTEGRATE**, for instance, could call **SOLVE**, and vice versa, which would allow you to compute the integral of an implicit function or solve an integral equation, terrific stuff indeed. But recursivity wasn't allowed so **SOLVE** couldn't call **SOLVE** (thus precluding easy solving of a system of 2 possibly non-linear equations in 2 unknowns), and **INTEGRATE** couldn't call **INTEGRATE**, thus precluding the numerical computation of double or multi-dimensional integrals. But the very availability of **INTEGRATE**, plus the excellent programmability means that we can overcome this limitation with extreme ease, as the following short sample program fully demonstrates.

Sample program: Two-dimensional Integration

Program listing

```
01 25.13.11   LBL A   24         2    2     47 23.71. 2   STO/2
02     23. 3  STO 3   25 23.71. 0   STO/0   48     24. 2  RCL 2
03    15.22    Rv     26        71    /     49 23.51. 8   STO+8
04     23. 4  STO 4   27     23. 1  STO 1   50    15.23   DSE
05    15.22    Rv     28     13. 0  GSB 0   51     22. 2  GTO 2
06       21   X<>Y    29        51    +     52     24. 8  RCL 8
07     23. 6  STO 6   30     13. 1  GSB 1   53    25.12   RTN
08       41    -      31     23. 2  STO 2   54 25.13. 0   LBL 0
09 24.14.23   RCL I   32     24. 1  RCL 1   55     24. 0  RCL 0
10       71    /      33     13. 0  GSB 0   56       73    .
11     23. 7  STO 7   34        41    -     57        6    6
12     24. 8  RCL 8   35     13. 1  GSB 1   58     14. 3  SQRT
13 23.41. 8   STO-8   36 23.51. 2   STO+2   59       61    x
14 25.13. 2   LBL 2   37         5    5     60    25.12   RTN
15     24. 6  RCL 6   38 23.61. 2   STOx2   61 25.13. 1   LBL 1
16     24. 6  RCL 6   39     24. 1  RCL 1   62     23. 5  STO 5
17     24. 7  RCL 7   40     13. 1  GSB 1   63     24. 4  RCL 4
18 23.51. 6   STO+6   41         8    8     64     24. 3  RCL 3
19       51    +      42        61    x     65 14.72.12   INTG B
20     23. 0  STO 0   43 23.51. 2   STO+2   66    25.12   RTN
21       21   X<>Y    44     24. 0  RCL 0   67 25.13.12   LBL B
22 23.41. 0   STO-0   45 23.61. 2   STOx2
23       51    +      46         9    9
```

Notes

- All in all, 66 program steps + f(x,y) definition (**LBL B**, ..., **RTN**)
- No flags used, labels A-B, 0-2, registers 0-8, I
- Also runs *as is* in an **HP-15C**, except **50 DSE** must be **50 DSE I**

Program description

HP-34C's powerful built-in **INTEGRATE** function allows the user to numerically compute integrals of the generic form:

$$I = \int_{a}^{b} f(x).dx$$

Alas, unlike the **HP-71B**, **INTEGRATE** can't be used recursively, i.e.: you can't use it to compute the definite integral of a function which itself is expressed as another integral. So, you can't include **INTEGRATE** as a program step within a subroutine which is to be called by **INTEGRATE**, either from a program or right from the keyboard.

This restriction is mainly due to the fact that **INTEGRATE** requires a lot of registers to do its work, and calling it recursively would mean having to use one extra bank of registers for each level of recursion implemented. There simply aren't that many registers available so recursivity is out of the question, which is a pity because a number of important technical applications do require computing definite 2-dimensional integrals routinely, of the generic form:

$$I = \int_{x_0}^{x_M} \int_{y_0}^{y_N} f(x,y)\ .dy.dx$$

However, the **HP-34C**'s full programmability comes to the rescue allowing us to easily implement this simple program (just 66 steps) which does use the built-in microcode **INTEGRATE** function together with a 5th-order Gaussian method to compute 2-dimensional integrals accurately and relatively fast. The present program I wrote uses **INTEGRATE** to compute the *inner* integral, namely:

$$y(x) = \int_{y_0}^{y_N} f(x,y)\ .dy$$

while the *outer* integral is computed using the following 3-point, 5th order Gaussian method for numerical quadrature, as follows: let's have the definite integral:

$$I = \int_{a}^{b} y(x)\ .dx$$

First of all, we transform the *arbitrary* integration interval **[a,b]** into the *fixed* interval **[-1,1]** by using a suitable change of variables, namely:

$$x = (b + a)/2 + t * (b - a)/2 \text{ , so we have } dx = (b - a)/2 * dt$$

Now, the resulting integral is computed like this (where $k = 0.6^{1/2}$):

$$\int_{-1}^{1} h(t)\ .dt = (8*h(0) + 5*(h(k) + h(-k)))/9$$

which is a 3-point Gaussian method, thus requiring just 3 evaluations of the function and returning *exact* results if **h(t)** is a polynomial of *degree 5* or less. In my opinion, this is by far the best existing method available for classical HP calculators, which were extremely RAM-challenged and somewhat slow, as it requires jut 3 evaluations per subinterval to achieve 5th-order accuracy and further, its coefficients are extremely simple numerics (5, 8, 9) or expressions (square root of 0.6), so they'll take a minimum number of program steps to store or compute them and thus no data registers are wasted for this purpose.

This compares favourably with higher-order Gaussian methods, which require more function evaluations (thus more time) and their coefficients are not so easy to describe, normally requiring two full registers to store their precomputed values for each point used (one for the **x** *ordinate*, the other for the *weight*). It also wins hands down against such simpler methods as Simpson's rule, which also requires 3 evaluations and has simple coefficients but provides only 3rd-order accuracy versus 5th-order for our present method, which is thus far more accurate.

Of course, using just *one* subinterval might result insufficiently accurate, so the program allows the user to specify the number of subintervals for the computation, and will apply the Gaussian method to each subinterval in turn, returning the grand total as the computed value of the integral. This way, the user can freely compromise between calculation time and accuracy.

A good strategy would be to start out with a modest number of subintervals, say 2, then keep on *doubling* the number of subintervals until the results coincide to the desired number of places. This is essentially what the built-in **`INTEGRATE`** function does, and for the purposes of this simple demo program we'll let the user decide whether to use this doubling strategy or chose a suitable number of subintervals right from the start (say 10), based on past experiences with similar functions and integration intervals, and accept the result as suitably accurate. If you're sophisticated enough, using some convergence acceleration technique is a definite possibility, such as *extrapolation to the limit* every couple iterations or so.

Also, do not forget to take into account that the inner integral is computed using the built-in **`INTEGRATE`** function, so the *display mode* will control overall accuracy and has to be set properly, as described in the **HP-34C *Owner's Handbook and Programming Guide***. As a general rule, selecting **`SCI 4`** is generally adequate to get 4-digit accuracy in the inner integral for modest integration intervals and, with a suitable choice for the number of subintervals, 4-digit overall accuracy for the outer integral as well. Going from 1 subinterval to 2 will usually double the running time, and going from **`SCI 4`** to **`SCI 6`** will take 2 to 3 times as long. These aren't hard rules, you'll need to experiment a little.

Usage instructions

Once you've keyed the program in, you must do the following:

1. Define the function to be integrated, **f(x,y)**. To that effect:
   - switch to **RUN** mode, if not already in it.
   - press **GTO B**
   - switch to **PRGM** mode
   - enter the necessary steps to compute the value of f(x,y), where you must assume that at the beginning the value of **X** is in **R5** and the value of **Y** is in the display (**X**-register). End your sequence with a **RTN** instruction.
   - switch to **RUN** mode

2. Specify the number of subintervals to use, M:
   - M, **STO I**

3. Enter the limits of integration and compute the double integral:
   - x_0, **ENTER**, x_M, y_0, **ENTER**, y_N, **A ?** Integral's value

4. For another computation using the same f(x,y) go to step 2 above.

5. For a different f(x,y) go to step 1 above, but don't forget to previously *erase* all program steps defining the former f(x,y) (except the **LBL B**, of course) before entering the new one.

Note:

The accuracy obtained depends on two factors, mainly: the chosen *display setting* **FIX N** or **SCI N**, and the number M of *subintervals* to use. In order to avoid excessive computing times, it is highly recommended to select **FIX 4** or **SCI 4** at most, and M=2 or M=4. This will get you 4 correct digits in reasonable times for most well-behaved f(x,y). If you aren't getting 4 correct digits or more, start by increasing the number of subintervals M to M=8, say, and only increase the display setting to **FIX 6** or **SCI 6**, say, as a last resort. See the examples to get a feeling of the obtainable accuracy for a number of cases and settings.

Examples

1. **Compute the following integral:**

$$I = \int_{0}^{1} \int_{1}^{2} (x^2 + y^2)\ .dy.dx$$

- First of all, let's define f(x,y). In RUN mode:

  GTO B, switch to PRGM, X^2, RCL 5, X^2, +, RTN

- As f(x,y) is a 2nd-degree polynomial in X,Y, our inner result will be *exact* using just 1 subinterval and a modest display setting, say FIX 4:

  1, STO I, FIX 4

- Now we enter the limits and compute the integral:

  0, ENTER, 1, ENTER, 1, ENTER, 2, A **?** 2.6667

- Changing to FIX 7, the result is, after some 2 min.:

  FIX 7 **?** **<u>2.6666666</u>**

  which is correct to all digits shown, as the exact result is I = 8/3

2. **Compute the following integral:**

$$I = \int_{3}^{4} \int_{1}^{2} 1/(x + y)^2\ .dy.dx$$

- After erasing the previous f(x,y), if any, we'll enter the definition for this new f(x,y) by following the same procedure. The definition is:

  RCL 5, +, x^2, 1/x, RTN

- Our new f(x,y) is not a polynomial in X,Y, so we can't expect an exact result, but let's use the same settings anyway:

  1, STO I, FIX 4

- As before, we now enter the limits and compute the integral:

  3, ENTER, 4, ENTER, 1, ENTER, 2, A **?** 0.0408

- Changing to FIX 6, the result is:

  FIX 6 **?** **<u>0.040821</u>**

  where the exact result is I = Ln(25/24) = 0.040822

3. Compute the following integral :

$$I=\int_{-2.3}^{1.6}\int_{3.9}^{6.1}(e^{-x*x}+x^3-y^3*x^2+7)*\tan^{-1}(x-2)*\sin(y+3).dy.dx$$

- First erase the previous f(x,y), if any, then enter the definition for this new f(x,y), as before:

  STO 9, 3, y^X, RCL 5, - RCL 5, x^2, *, LASTX, CHS, e^X, X<>Y,

  -, 7, +, RCL 5, 2, -, TAN^{-1}, *, RCL 9, 3, +, SIN, *, RTN

- This is a pretty *complicated* f(x,y) and both intervals of integration are somewhat *large*, so let's try 2 subintervals and SCI 4:

  2, STO I, SCI 4, RAD

- Let's enter the limits and compute the integral:

  -2.3, ENTER, 1.6, ENTER, 3.9, ENTER, 6.1, A **?** 1.3213E3

- changing to FIX 2, the result is:

  FIX 2 **?** **<u>1321.27</u>**, where the exact result is I = 1321.275779

  Despite each evaluation of f(x,y) taking 6 seconds, which increases the computation time, and both inner and outer intervals of integration being quite wide, which affects the accuracy, we've nevertheless got 6 correct digits in reasonable time (some 17 min.). You might want to try your modern HP models and see how they compare !

4. Computing the following improper integral:

$$I = \int_{0}^{Inf}\int_{0}^{Inf} e^{-x^2-y^2}\,.dy.dx$$

- As always, do erase the previous f(x,y), if any, then enter the definition for this new f(x,y), which is

  x^2, RCL 5, x^2, +, CHS, e^X, RTN

- This *improper* double integral does appear in Probability. As the limits are both Infinity (actually, 4 will have to do) we'll use 3 subintervals:

  3, STO I, FIX 4

- Now, as we can't enter infinite limits, we'll stick to 4 for both upper limits, as the resulting functional value (e^{-32}), is suitably small:

  0, ENTER, 4, ENTER, 0, ENTER, 4, A **?** **<u>0.7853</u>**

  where the exact result is I = Pi/4 =0.7854

Final remarks

The **HP-34C** was indeed a landmark machine. On the one side, despite being an HP calculator (i.e., a luxury item) it was meant to be really *affordable*, even if build quality suffered a little in the end, most specially fragile battery contacts and not feeling nearly as rock-solid as earlier models did. On the other hand, it was a *very powerful*, revolutionary programmable with a comprehensive function set, full programmability, automatically partitionable continuous memory, and those wonderful, *state-of-the-art* numeric **`SOLVE`** and **`INTEGRATE`** capabilities right at your fingertips, a true *world-first*, adequately served by a superb manual. It all utterly more than made for the slight quality problems, which never were that bad to begin with. People would get to know it better and *loved* it, in spades.

Certainly we did, which is why my friend Fernando del Rey and I set up to the HP-commissioned task of writing a full **HP Solutions Book** for it, namely ***"HP-34C Advanced Math"***, featuring 20+ excellent programs, a labour of love, which made the most of showing off the powerful **HP-34C**-specific capabilities. It was primarily intended to help boost **HP-34C** sales among students. Not that it made any difference, actually: the **HP-34C** sold like *mad* ! **Long Live the HP-34C !**