Long Live the HP-25 !

Valentín Albillo (HPCC #1075)

The HP-25 was my very first HP calculator. Back then in 1975 I was about to enter University and getting a powerful calculator to help me through was at the top of my To-Do list. I already had a simple (if quite expensive) four-function **Sears** calculator with one memory register and no square root, but after a colleague of mine showed me his recently acquired **HP-21** and introduced me to that most misterious yet unbelievably powerful RPN way, I found myself utterly hooked to the point were I couldn't contemplate getting or using an ordinary algebraic calculator anymore. It had to be an RPN one, nothing less would do for me.

Yet the problem was, even the HP-21 was very expensive, much less than the HP-65, say, but still much more than any normal algebraic model. As a 'poor' student, I could hardly afford such an expense but never mind, I got the money after a whole year of savings and deprivation. With the money in hand I was ready to go and order my very own HP-21 when something happened. A new HP model, the **HP-25**, had just been released ! It was visually quite similar to the HP-21 in size and shape, but it seemed to be much more powerful, with many additional functions, eight (!) storage registers and, lo and behold, programmable (!!).

Now, this was a mixed blessing to me. On the one hand, this new machine was way beyond my wildest dreams in power and functionality. I was pretty satisfied with what the HP-21 had to offer, so obviously it goes without saying that the HP-25 far exceeded any wish list I could have. It was the machine I wanted, and I was sure it would serve me exceedingly well in my incoming universitary endeavours and way beyond that, even for my personal, mathematical hobbies. On the other hand, it was significantly more expensive than the HP-21, so the money I had painfully managed to save for its acquisition was plainly insufficient. What to do ?

Well, I simply couldn't resist. After tremendous struggles and efforts, I did manage to get the additional money and ordered my HP-25 at once. After a week or so (an specialized order, very few people could afford or were interested in one in my neighbourhood) I finally got it in my hands, and was instantly marveled at the feeling of incredible quality it oozed. Here it was, a small yet very solid device, with a wonderful red-LED display, extremely pleasant keyboard with two (!!) shift keys, golden and blue, a nice carrying case, and two awesome books to accompany it, the exceedingly interesting and well-written Owner's Handbook (full of wonderful photographs which depicted the small HP-25 in a variety of scientific and engineering environments, dwarfed by the huge measuring or computing devices nearby), and the unbelievable Applications Manual, which provided tons of useful programs with full explanations, relevant equations, and joyful examples.

I then spent the next few weeks dedicating nearly 100% of my free time to absorb all the information and try out every single program and example, being constantly astonished by its capabilities, which were much greater than I expected. I knew that it was 'programmable', but I wasn't sure at all what that exactly meant, this is, what it could do. I thought that it would simply remember keystrokes and would be able to compute a complicated expression repeatedly at the touch of a key, but that would be all. I hadn't expected logical tests, branching, single-step capability, etc., so I was amazed no end when I first saw that Newton's method for solving f(x)=0 was featured as one of the application programs in the book.

That this small calculator would be capable of storing the equation definition, then unattendedly compute both its value for some argument and its derivative as well, then perform the necessary test to decide whether convergence was achieved and if so stop and display the root, or else update the approximation and loop back for yet another iteration, all by itself, I could hardly believe. Now, in 2006, this may sound trivial but back then in 1975 it was nothing sort of miraculous, you'd need a room-sized computer or an incredibly expensive, large programmable desktop calculator to even attempt it. Yet this thing was reliably and quietly solving non-linear equations while sitting in the palm of my hand. A miracle.

Up to then, I had written 'programs' for my four-function calculator, this is, sequences of keystrokes that, when repeated mechanically, without thinking much if at all, would produce the answer to some complicated calculation. I had such 'programs' for computing square roots, trigonometric and exponential functions, and even solve some simple equations, such as cubics and certain quintics. But of course they were absolutely manual and tedious procedures As soon as I learned to program my HP-25, these procedures were the first thing of my own I tried on it, and to my utter delight, they worked flawlessly!

I then took it to my University, where as it happened we students had access to a computer, suitably advanced for its time yet extremely primitive by today standards. It was a 3-terminal device, where each terminal had a teletype-style console with a keyboard and a line printer (but no display or video screen), with just 1.3 Kb of RAM for each terminal and a rudimentary version of BASIC , with just one statement per line, dreadful 6-digit accuracy, no string manipulation, non-optional LET statement, rudimentary and unreliable matrix operations, and paper punch I/O.

The first day I got there, I promptly showed my brand-new, just gathered-from-the-shop HP-25 to my teacher before even unwrapping its manuals, and asked him what it could do. "Everything this computer does" he said. Indeed. Most of the programming exercises I was asked to do in the computer I could do, and much more conveniently, in my HP-25C. I was extremely proud with my acquisition and had thus become an RPN and HP calc lover overnight.

Later, I created much more complicated programs specifically to help me at exams, such as computing elliptic functions. Indeed, the teachers would let me use it at exams with the one condition that I wrote the programs myself, which I did, and they were amazed with the results. While everyone was wasting time looking for some elliptic function in cumbersome tables, my HP-25 would simply compute it, to 8-10 digit accuracy, in mere seconds. No searching, no interpolations, no inaccurate results. My exams were not only correct, but much more accurate than anyone else's. I was often out of the examination hall more than half an hour before the allotted time elapsed. I still remember one of the problems: given an arc of a 3-D helix between two points, to find the coordinates of the points which trisected its arc length. This required computing some elliptic functions and their inverses, and the HP-25 did the drudgery for me with unbelievable precision.

What then had the HP-25 which was so revolutionary for its time ? A lot. Namely:

- 10-digit accuracy in the range from 1E-100 to 1E+100, where most calculators just had 8 digits, from 1E-8 to 1E+8.
- 8 storage registers, with full storage arithmetic, where most calculators just had one memory register, with M+ and M-
- A vast range of math functions, including reciprocal, square root, all trigonometric and their inverses in radians, gradians and degrees, exponentials, logarithms, exponentiation, rectangular to polar conversions and vice versa, etc., where most calculators just had the usual arithmetic functions plus square root, and the few that had some transcendental functions usually implemented them with dreadful accuracy.
- Full RPN, with 4 stack registers plus LASTX, where most other calculators of the time were algebraic, with no parentheses or just one or two levels of pending operations and inconsistent hierarchy of operations.
- Small size, bright display, solid, reliable, extremely good keyboard and rechargeable batteries, where most other calculators felt feeble and had horrible, unreliable keyboards and displays.
- Exceedingly good, enjoyable, comprehensive documentation, with an Owner's Handbook which was a work of art and an Applications Manual way beyond belief, when most calculators would have an small pamphlet full of trivial examples or a back label with common operations, if at all.
- Programmability. 49 fully-merged (up to 3 keystrokes) steps, logical tests, branching and looping, single-step execution, simple program entering and edition. No non-HP calculators at the time could compare, only the HP-65 could (at 5-6 times the price and without fully merged program steps at that)
- And of course, classic HP outmost quality and service.

You really need to have lived those times to fully understand what the HP-25 meant to many of us, fledgling engineers. Now, let's show what the HP-25[1] can do.

[1] No HP-25 ? No problem. You can either use the nice HP-25 Java applet at the Museum of HP or else google for and download the free **Nonpareil** emulator which includes the HP-25 as well.

Sample programs: Solving Differential Equations

Numerically solving equations such as **f(x)=0** is one thing. But solving 1st-order differential equations **y'=f(x,y)** is quite another. Back then in 1975 you really weren't expected to be capable to do either, much less automatically. As the Applications Manual convincingly showed, the HP-25 could do the former, which required the numerical computation of the derivative of f(x). This needed just two evaluations of f(x), which was somewhat tricky as, unfortunately, the HP-25 capabilities did not include subroutine calling, **GSB** (Go-Sub), as the HP-65 did. Yet the Applications Manual demonstrated a neat trick you could use to distinguish whether you were evaluating f(x) or f(x+inc), which essentially amounted to using a simulated flag, as flags, though also available in the HP-65, were not part of the HP-25 programming paradigm either.

Now, to numerically solve a 1st-order differential equation **y'=f(x,y)**, there are a number of algorithms we can use. The Applications Maual included one, namely a modified version of Euler's method. This is as simple as it gets, but it is a low-accuracy method, which requires a very small step size to achieve passable results and that implies long execution times and rounding error accumulation.

A much better, widely used method, is **Runge-Kutta's 4th-order algorithm** (the error is proportional to h^5, which is very small for small h). It goes like this: given **y'=f(x,y)** and the initial condition **$y(x_0)=y_0$**, we must find $y(x_0+h)$ for some suitable step h. Using Runge-Kutta's 4th-order method, we must compute:

$$k_1 = h * f(x_0, y_0)$$
$$k_2 = h * f(x_0 + h/2, y_0 + k_1/2)$$
$$k_3 = h * f(x_0 + h/2, y_0 + k_2/2)$$
$$k_4 = h * f(x_0 + h, y_0 + k_3)$$

and then we finally have:

$$y_1 = y(x_1) = y(x_0 + h) = y_0 + (k_1 + 2 * k_2 + 2 * k_3 + k_4)/6$$

and iterating this we can compute y(x) for any value of x. But the problems are:

- Can we fit all of this in the 49 steps of program memory provided by the HP-25, including both the program itself *plus* sufficient space to define some reasonable, non-trivial f(x,y) ?
- Computing the k-values requires ***four*** calls to the f(x,y) definition. How can we make these four calls *and* return to the proper sequence in the algorithm having no subroutine capabilities and no flags ? A simulated flag will allow your program to easily return to two different places, but *four* ?

Despite the apparently insurmountable difficulties, most specially the 49-step limit, I was so dissatisfied with Euler's method's abysmal performance that I tried very hard to implement Runge-Kutta's 4th order method. But after several unsuccessful attempts and under the pressure of incoming exams, I finally gave up for the moment and, not wishing to go back to Euler's 1st-order method, I decided that if 4th-order wasn't within reach, perhaps 3rd-order would, so I worked the theory out and came up with this 3rd-order Runge-Kutta method instead:

$$k_1 = h * f(x_0, y_0)$$
$$k_2 = h * f(x_0 + 2/3 * h, y_0 +2/3 * k_1)$$
$$k_3 = h * f(x_0 + 2/3 * h, y_0 +2/3 * k_2)$$

$$y_1 = y(x_1) = y(x_0 + h) = y_0 + (2 * k_1 + 3 * k_2 + 3 * k_3)/8$$

which requires just ***three*** evaluations of f(x,y) instead of four, so it seemed more amenable to fit within the strict limitations, as well as faster if slightly less accurate (error proportional to h^4 instead of h^5). Taking advantage of the repetitive patterns I eventually came up with this HP-25 program:

```
01  CLX      11   ...   21  X<0     31   +      41  RCL 5
02  STO 3    12   ...   22  GTO 38  32  RCL 0   42  STO+3
03  RCL 2    13   ...   23  Rdown   33  RCL 4   43    8
04  RCL 1    14   ...   24  STO 5   34   *      44  STO/3
05  f(x,y)   15   ...   25  STO+3   35  RCL 1   45  RCL 0
06   ...     16   ...   26  STO+3   36   +      46  STO+1
07  GTO 18   17   ...   27  RCL 4   37  GTO 05  47  RCL 3
08   ...     18  RCL 0  28  STO-7   38  STO/ 7  48  STO+2
09   ...     19   *     29   *      39   /      49  RCL 2
10   ...     20  RCL 7  30  RCL 2   40  STO-3
```

which does the job and still lets you with 13 free steps to define your f(x,y); it computes the next datapoint and stops with y_n in the display (and in R2, while x_n is stored in R1). To use it, you must:

- previously store a couple of constants, namely 2/3 in R4 and 1 in R7. This is done *only once* no matter how many f(x,y) you may define and solve.
- switch to PRGM mode and define your f(x,y) from step 05 up to and including step 17, assuming x is in X and y is in Y. If f(x,y) doesn't use all available steps, you must include a GTO 18 at its end.
- switch back to RUN mode and store initial values: h in R0, x_0 in R1, y_0 in R2, then f PRGM and R/S

you'll get $y_1 = y(x_1) = y(x_0 + h)$. Press R/S again to get y_2, y_3, and so on. You can also change the step value h at any moment by storing its new value in R0, which may be useful to improve accuracy near singularities or extrema.

Let's see an example: Given **y' = x^3y^3 –xy**, with y(0)=1/Sqrt(2), find y(1).

We do the following, using h = 0.1 as our initial step size:

- define f(x,y):

  GTO 04, PRGM mode, *, X^2, LASTX, *, LASTX, -, GTO 18, RUN mode
- store once-only constants:

  2 , ENTER, 3, /, STO 4, 1, STO 7
- store initial values and step size:

  0 (x_0), STO 1, 2, SQRT, 1/X (y_0), STO 2, 0.1 (h), STO 0, FIX 5
- run it: **f PRGM, R/S**; by repeatedly pressing **R/S** you'll eventually get:

X	0.1	.02	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y	.70359	.69324	.67663	.65462	.62819	.59832	.56592	.53175	.49642	.46037

which are indeed correct to 5 decimal places.

<u>You are hereby encouraged to single-step through the program </u>to see how neatly it does manage to call f(x,y) three times with different arguments, yet it always returns to the proper sequence by a most clever use of the constants at R4 and R7.

Upon seeing and analyzing this program, my friend Fernando del Rey (former PPC member #4995 and nowadays a proud HPCC member as well) was, in his own words, "enthralled and amazed" at how cunningly it managed to get the job done, and felt inspired enough that he decided to try his hand at the elusive 4th-order method, using similar tricks as the ones he had seen in my 3rd-order attempt.

To our utter astonishment and delight, he succeeded as well, and produced the following marvel, that I still consider the very best program for the HP-25 I've ever seen, and one of the most clever pieces of code for *any* machine, period:

```
01  RCL 2    11   ...    21  GTO 26  31   *      41  STO/ 7
02  RCL 1    12   ...    22   +      32  RCL 2   42  RCL 3
03  f(x,y)   13  RCL 0   23  1/X     33   +      43  STO-3
04   ...     14   *      24  STO 4   34  RCL 4   44   6
05  GTO 13   15  STO+3   25  GTO 31  35  STO-7   45   /
06   ...     16  RCL 7   26   -      36  RCL 0   46  STO+2
07   ...     17  X<0     27  CHS     37   *      47  RCL 0
08   ...     18  GTO 41  28  STO 4   38  RCL 1   48  STO+1
09   ...     19   1      29  X<>Y    39   +      49  RCL 2
10   ...     20  X#Y     30  STO+3   40  GTO 03
```

It works very much like mine, except it leaves 10 program steps free to define f(x,y) (which must end with a GTO 13 if it doesn't use all of them), and the initial, once-only stored constants are 0 in R3 and 1 in R7. The usage instructions are the same, except you must begin your f(x,y) definition at step 03.

Let's see it in action: Given **$y' = (x^2 + \tan^2 y)/(1 + \tan^2 y)$** with y(0)=0, find y(1).

We do the following, using a 0.2 step size, for faster results:

- define f(x,y):

  GTO 02, PRGM, X^2, X<>Y, TAN, X^2, +, 1, LASTX, +, /, NOP, RUN mode
- store constants:

  0, STO 3, 1, STO 7
- store initial values and step size, and set RAD mode:

  0 (x_0), STO 1, 0 (y_0), STO 2, 0.2 (h), STO 0, RAD, FIX 6
- run it: **f PRGM, R/S**; by repeatedly pressing R/S you'll eventually get:

X	0.2	0.4	0.6	0.8	1
Y	0.002667	0.021357	0.072323	0.172355	0.336878

which are correct to 6 digits within a few ulps, despite using a larger h=0.2

Final remarks

I think the above examples truly demonstrate just how wonderful and capable the HP-25 was at its time, and the sheer enjoyment of owning it and making the most out of it. It made me an HP fan for life and did wonders to my then incipient programming abilities, thus probably changing my future life for the best.