Boldly Going – HP-16C Scientific Functions Part 2

Welcome to a new article in my *"Boldly Going"* series, this time featuring the second part of the mini-series of articles dedicated to provide the *HP-16C Computer Scientist* calculator with a full set of the main scientific functions usually available in most *HP* scientific models, and some of them even in financial models such as the *HP-12C.* Here we'll address the implementation of the 7 inverse functions corresponding to the direct ones already discussed and implemented in *Part 1.*

Introduction

For completeness' sake, I'll re-state the goals introduced in the first part.

The *HP-16C* is a fantastic programmer's calculator belonging to the legendary *Voyager Series*, which includes such landmark models as the financial *HP-12C* and the *"Swiss-knife" HP-15C.* Indeed, the *HP-16C* excels at its design goals, with a most comprehensive instruction set covering all the needs of professionals and students working in the field, plus decent programmability and all the many exceptional traits germane to the *Voyager Series*, such as *Continuous Memory*, incredibly long battery life, rock-solid ergonomic hardware, the works.

However, when it comes to floating-point calculations, the *HP-16C* falls surprisingly short of the mark, even when compared to the *financial* (not *scientific) HP-12C.* Regrettably, its floating-points capabilities are restricted to basic arithmetic (+, −, x, ÷), reciprocal (*1/x*) and square root ($\sqrt{x}$) and that's all. It lacks the logarithm (*LN*), exponential (e^x) and power (y^x) functions, which even the *HP-12C* includes, let alone trigonometrics, to say nothing of hyperbolics or advanced functions such as *Gamma*. It also lacks any sort of statistical functions such as the various summations, mean, standard deviation, linear regression, and even the factorial (*n!*).

That being so, this threesome mini-series of articles strives to overcome those limitations by providing a full set of scientific functions, discussed and implemented as follows:

- *Part 1 – Direct Functions*: trigonometric (*sin*, *cos*, *tan*), hyperbolic (*sinh*, *cosh*, *tanh*), exponential (e^x).
- *Part 2 – Inverse Functions*: trigonometric (*arcsin*, *arccos*, *arctan*), hyperbolic (*arcsinh*, *arccosh*, *arctanh*), natural logarithm (*ln*).
- *Part 3 – Extra Functions:* powers (y^x), factorial (*n!*), *Gamma* function, etc.

Boldly going ...

As stated in the ***Introduction*** above, the purpose of this *Part 2* is to implement the 7 inverse functions *arcsin(x)*, *arccos(x)*, *arctan(x)*, *arcsinh(x)*, *arccosh(x)*, *arctanh(x)* and *ln(x)*, subject to these five desirable requirements:

1) All seven inverse functions must fit into a single, relatively short *(100-step)* program.
2) Full or very extended ranges, up to $-10^{50} < x < 10^{50} - 10^{100}$, depending on the particular function.
3) Fast execution speed over the whole argument range, a few seconds at most.
4) About ***5-6*** correct digits or better over the whole argument range for all functions.
5) Convenience of use, each function will be callable by pressing just two *easy-to-remember* keys.

Two questions come to mind:

Why settle for just *5-6* correct digits instead of *9-10* ? Because that would conflict with *three* of the other requirements: the program would be significantly *longer*, would have a *restricted* argument range for some of the functions, and would run noticeably *slower*. It seems better to go and achieve those three worthwhile goals even if this means some small sacrifice in accuracy, which nevertheless is rarely justified in real-world applications.

Why not have *all* direct and inverse functions included in a *single* program ? Again, several reasons:

i. Having all 14 functions in a single program would need the *whole* of program memory, leaving *no* program memory for anything else and also *no* storage registers for manual use, except the index ***R_I***.

ii. Having to delete and then key in anew 200+ steps every time you need to run other programs or have some storage registers available to perform other calculations, would be a tiresome, error-prone chore.

iii. There aren't enough labels to define and use all 14 functions. Calling the functions themselves needs 14 labels, one for each function (***GSB A-F***, ***GSB 0-7***) and as there are only 16 possible labels in all, that would leave just two labels for internal use (mostly calling internal subroutines), which aren't enough.

iv. As it's next to impossible to fit all 14 functions at once, if desired the users can go and bundle the functions they want by including the relevant code sections into a single program. For instance, a user might want to include the trig functions, the exponential function and the logarithmic function, forfeiting hyperbolics altogether. That would be just 8 functions, it's doable, and it's left as an exercise for the reader, though some useful relevant details are given in the ***Notes on function extraction*** below.

To achieve all five goals above, we'll have to overcome the following *HP-16C* shortcomings:

a) No *storage* (let alone *recall*) *arithmetic* and no summations either, so all operations have to be performed on the stack, which wastes both program steps and time, and loses stack registers. Even the *HP-12C* allows for storage arithmetic using register *R_0* to *R_4*, but not the *HP-16C.*

b) no ***x^2*** or ***y^x*** functions, no ***INT***, ***FRAC***, ***ABS***, ***SGN*** or ***n!*** , which might come handy. Most functions available for integer bases (e.g.: multiply/divide by *2^n*, remainder) don't work in *floating-point* mode.

Program Listing for the HP-16C

01	**LBL A**	*26*	CF 1	*51*	5	*76*	1
	SF 1		CF 0		EEX		-
	GSB 9		GTO 8		CHS		F? 1
	-		**LBL F**		6		CHS
05	CHS	*30*	CF 1	*55*	F? 1	*80*	ENTER
	$\sqrt{x}$		GSB 9		CHS		+
	GTO 8		-		+		$\sqrt{x}$
	LBL B		CHS		X<>Y		DSZ
	SF 1		$\sqrt{x}$		F? 1		DSZ
10	CF 0	*35*	1/X	*60*	X<>Y	*85*	**LBL 6**
	GTO 8		GTO 8		**LBL 7**		ENTER
	LBL C		**LBL 0**		F? 1		+
	SF 1		CF 1		X<>Y		DSZ
	GSB 9		CF 0		STO 0		GTO 6
15	+	*40*	ENTER	*65*	1	*90*	F? 0
	$\sqrt{x}$		1/X		+		CHS
	1/X		X>Y?		2		RTN
	GTO 8		SF 0		÷		**LBL 9**
	LBL D		+		$\sqrt{x}$		CF 0
20	CF 1	*45*	2	*70*	ISZ	*95*	ENTER
	GSB 9		÷		F? 1		X<0?
	+		**LBL 8**		X<>Y		SF 0
	$\sqrt{x}$		1		X>Y?		x
	GTO 8		STO I		GTO 7		1
25	**LBL E**	*50*	1	*75*	RCL 0	*100*	RTN

Resources used:

- *100 steps (but 100* RTN *isn't needed if that's the end of program memory)*

- *flags 0, 1*
- *labels A-F, 0, 6-9*
- *registers 0, I*

FLOAT mode required

103 bytes left for other uses such as additional routines or storage registers, i.e.: up to 14 floating-point registers, plus register $\boldsymbol{R_I}$

Registers used:

0:	*scratch*
I:	*loop control*

Function details:

Function	*Call sequence*	*Input Range*	*Program steps used*
arcsin(x)	GSB **A**	$-1 \leq x \leq 1$	*01-07 + 47-100*
arccos(x)	GSB **B**	$-1 \leq x \leq 1$	*08-11 + 47-92*
arctan(x)	GSB **C**	$-10^{50} < x < 10^{50}$	*12-18 + 47-100*
arcsinh(x)	GSB **D**	$-10^{50} < x < 10^{50}$	*19-24 + 47-100*
arccosh(x)	GSB **E**	$1 \leq x < 10^{50}$	*25-28 + 47-92*
arctanh(x)	GSB **F**	$-1 < x < 1$	*29-36 + 47-100*
ln(x)[1]	GSB **0**	$10^{-99} \leq x < 10^{100}$	*37-92*

Notes on function extraction:

- For the purpose of creating a mix of choice functions into a single program, the user must extract the program steps specified above for each function. All functions require the *47* **LBL 8** .. *92* **RTN** subroutine.
- ***Group 1***: the hyperbolic functions *arcsinh(x)*, *arccosh(x)* and *arctanh(x)*, as well as the logarithm *ln(x)*, all clear flag **1**, so **CF 1** itself and all pairs of steps **F? 1** *(some step)* needn't be extracted, saving up to 14 bytes.
- ***Group 2***: the trigonometric functions *arcsin(x)*, *arccos(x)* and *arctan(x)* all set flag **1**, so **SF 1** itself and all the single instructions **F? 1** need not be extracted, saving up to 8 bytes.
- If extracting functions from <u>*both*</u> ***Group 1*** *and* ***Group 2*** at once, the above instructions do <u>*not*</u> apply.

[1] The ***ln(x)*** function returns the natural (base ***e***) logarithm of *x*. To obtain logarithms to any other base *b>1*, simply divide the result of ***ln(x)*** by the constant *ln(b)*, e.g.; $\boldsymbol{log_{10}(x)}$ = ***ln(x)*** */ ln(10)* = ***ln(x)*** */ 2.302585093*, and $\boldsymbol{log_2(x)}$ = ***ln(x)*** */ 0.6931471806*.

Usage Instructions

To evaluate any of the seven available inverse functions for a given argument ***x***, proceed as follows: first of all place the *HP-16C* in *floating-point mode* by executing `FLOAT 0..9` or `FLOAT .` (eq. to **`SCI 6`**) and then:

y = arcsin(x)	*Input Range:* $-1 \leq x \leq 1$	*Output Range:* $-\pi/2 \leq y \leq \pi/2$

x $\boxed{\texttt{GSB A}}$ → *arcsin(x)*

y = arccos(x)	*Input Range:* $-1 \leq x \leq 1$	*Output Range:* $0 \leq y \leq \pi$

x $\boxed{\texttt{GSB B}}$ → *arccos(x)*

y = arctan(x)	*Input Range:* $-10^{50} < x < 10^{50}$	*Output Range:* $-\pi/2 < y < \pi/2$

x $\boxed{\texttt{GSB C}}$ → *arctan(x)*

y = arcsinh(x)	*Input Range:* $-10^{50} < x < 10^{50}$	*Output Range:* $-115.823 < y < 115.823$

x $\boxed{\texttt{GSB D}}$ → *arcsinh(x)*

y = arccosh(x)	*Input Range:* $1 \leq x < 10^{50}$	*Output Range:* $0 \leq y < 115.823$

x $\boxed{\texttt{GSB E}}$ → *arccosh(x)*

y = arctanh(x)	*Input Range:* $-1 < x < 1$	*Output Range:* $-11.8597 < y < 11.8597$

x $\boxed{\texttt{GSB F}}$ → *arctanh(x)*

y = ln(x)	*Input Range:* $10^{-99} \leq x < 10^{100}$	*Output Range:* $-227.958 < y < 230.2591$

x $\boxed{\texttt{GSB 0}}$ → *ln(x)*

Notes:

- The computation time is mostly independent of the specific function being evaluated (just a few seconds) and varies by no more than a factor of about 2 between the extremes of the particular range.
- The accuracy is typically **5-6** correct *digits* (not *decimals*) or better for all functions and ranges. See the **Notes** section at the end of this article for a detailed discussion and the **Appendix** for comprehensive sample results.
- Input values outside the *Input Range* will result in an error condititition. See **Note 2** at the end of this article.
- The initial stack contents aren't preserved, so previously store in some registers (except $\boldsymbol{R_0}$ or $\boldsymbol{R_I}$) any contents you may want to keep between calculations. Also, ***x*** isn't automatically saved in **`LAST X`**.
- Results for *arcsin*, *arccos* and *arctan* are in *radians*. For degrees, multiply them by 180/π = *57.29577951*.

Examples

Compute *arcsin(x)* for *x=(−0.7, −0.1, 1)*, *arccos(x)* for *x=(−1, −0.1, 0.7)*, *arctan(x)* for *x=(−0.25, 20, ~10^{50})*, *arcsinh(x)* for *x=(−1.5, 50, ~10^{50})*, *arccosh(x)* for *x=(1.5, 50, ~10^{50})*, *arctanh(x)* for *x=(-0.4, 0.9, 0.995, 0.99999, ~1)*, and *ln(x)* for *x=(6.4. 10^{-23}, 0.15, 1.5, 50, 10^{50}, ~10^{100}).*

$\boxed{\texttt{FLOAT 6}}$

arcsin(x):

−0.7	$\boxed{\texttt{GSB A}}$	⟶	−0.775397	{ all 6 digits are correct }
−0.1	$\boxed{\texttt{GSB A}}$	⟶	−0.100167	{ all 6 digits are correct }
1	$\boxed{\texttt{GSB A}}$	⟶	1.57079**3**	{ last digit should be a **6** }

arccos(x):

−1	$\boxed{\texttt{GSB B}}$	⟶	3.1415**86**	{ last digits should be **93** }
−0.1	$\boxed{\texttt{GSB B}}$	⟶	1.670964	{ all 6 digits are correct }
0.7	$\boxed{\texttt{GSB B}}$	⟶	0.79539**8**	{ last digit should be a **9** }

arctan(x):

−0.25	$\boxed{\texttt{GSB C}}$	⟶	−0.24497**8**	{ last digit should be a **9** }
20	$\boxed{\texttt{GSB C}}$	⟶	1.520838	{ all 7 digits are correct }
~1E50	$\boxed{\texttt{GSB C}}$	⟶	1.57079**3**	{ last digit should be a **6**; key in ~1E50 as 9.999999999 $\boxed{\texttt{EEX}}$ 49 }

arcsinh(x):

−1.5	$\boxed{\texttt{GSB D}}$	⟶	−1.19476**9**	{ last digit should be a **3** }
50	$\boxed{\texttt{GSB D}}$	⟶	4.6052**69**	{ last digits should be **70** }
~1E50	$\boxed{\texttt{GSB D}}$	⟶	115.822**249**	{ last digits should be **402**; key in ~1E50 as 9.999999999 $\boxed{\texttt{EEX}}$ 49 }

arccosh(x):

1.5	$\boxed{\texttt{GSB E}}$	⟶	0.9624**37**	{ last digits should be **24** }
50	$\boxed{\texttt{GSB E}}$	⟶	4.6050**98**	{ last digits should be **70** }
~1E50	$\boxed{\texttt{GSB E}}$	⟶	115.822**249**	{ last digits should be **402**; key in ~1E50 as 9.999999999 $\boxed{\texttt{EEX}}$ 49 }

arctanh(x):

−0.4	$\boxed{\texttt{GSB F}}$	⟶	−0.423649	{ all 6 digits are correct }
0.9	$\boxed{\texttt{GSB F}}$	⟶	1.47221**1**	{ last digit should be a **9** }
0.995	$\boxed{\texttt{GSB F}}$	⟶	2.9944**78**	{ last digits should be **81** }
0.99999	$\boxed{\texttt{GSB F}}$	⟶	6.1030**74**	{ last digits should be **34** }
~1	$\boxed{\texttt{GSB F}}$	⟶	11.859**671**	{ last digits should be **499**; key in ~1 as **.**9999999999 }

ln(x):

6.4E-23	$\boxed{\texttt{GSB 0}}$	⟶	−51.103**900**	{ last digits should be **159** }
0.15	$\boxed{\texttt{GSB 0}}$	⟶	−1.8971**31**	{ last digits should be **20** }
1.5	$\boxed{\texttt{GSB 0}}$	⟶	0.40546**9**	{ last digit should be a **5** }
50	$\boxed{\texttt{GSB 0}}$	⟶	3.91202**8**	{ last digit should be a **3** }
1E50	$\boxed{\texttt{GSB 0}}$	⟶	115.129**518**	{ last digits should be **255** }
~1E100	$\boxed{\texttt{GSB 0}}$	⟶	230.25**9036**	{ last digits should be **8509**; key in ~1E100 as 9.999999999 $\boxed{\texttt{EEX}}$ 99 }

Appendix: Sample results

Upper value = theoretical result, *Lower value = computed result* ** ~1E50 = 9.999999999 E49*

x	arcsin(x)	arccos(x)	x	arcsin(x)	arccos(x)
-1	-1.5707 96 -1.5707 93	3.1415 93 3.1415 86	*0.1*	0.1001 67 0.1001 67	1.4706 29 1.4706 26
-0.98	-1.3704 61 -1.3704 54	2.9412 58 2.9412 52	*0.2*	0.2013 58 0.2013 57	1.3694 38 1.3694 31
-0.95	-1.2532 36 -1.2532 29	2.8240 32 2.8240 32	*0.3*	1.2661 04 1.2660 99	0.3046 93 0.3046 91
-0.9	-1.1197 70 -1.1197 65	2.6905 66 2.6905 54	*0.4*	0.4115 17 0.4115 16	1.1592 79 1.1592 74
-0.8	-0.9272 95 -0.9272 91	2.4980 92 2.4980 83	*0.5*	0.5235 99 0.5235 97	1.0471 98 1.0471 95
-0.7	-0.7753 97 -0.7753 97	2.3461 94 2.3461 84	*0.6*	0.6435 01 0.6434 98	0.9272 95 0.9272 91
-0.6	-0.6435 01 -0.6434 98	2.2142 97 2.2142 86	*0.7*	0.7753 97 0.7753 97	0.7953 99 0.7953 98
-0.5	-0.5235 99 -0.5235 97	2.0943 95 2.0943 90	*0.8*	0.9272 95 0.9272 91	0.6435 01 0.6434 98
-0.4	-0.4115 17 -0.4115 16	1.9823 13 1.9823 03	*0.9*	1.1197 70 1.1197 65	0.4510 27 0.4510 26
-0.3	-0.3046 93 -0.3046 91	1.8754 89 1.8754 83	*0.95*	1.2532 36 1.2532 29	0.3175 60 0.3175 59
-0.2	-0.2013 58 -0.2013 57	1.7721 54 1.7721 52	*0.98*	1.3704 61 1.3704 54	0.2003 35 0.2003 35
-0.1	-0.1001 67 -0.1001 67	1.6709 64 1.6709 64	*1*	1.5707 96 1.5707 93	0 0
0	0 0	1.5707 96 1.5707 93			

x	arctan(x)	x	arctan(x)
0	0 0	*10*	1.4711 28E00 1.4711 25E00
0.1	9.9668 65E-02 9.9668 31E-02	*20*	1.5208 38E00 1.5208 38E00
0.25	2.4497 87E-01 2.4497 83E-01	*50*	1.5507 99E00 1.5507 94E00
0.5	4.6364 76E-01 4.6364 54E-01	*100*	1.5607 97E00 1.5607 90E00
0.75	6.4350 11E-01 6.4349 76E-01	*1E3*	1.5697 96E00 1.5697 91E00
1	7.8539 82E-01 7.8539 66E-01	*1E4*	1.5706 96E00 1.5706 97E00
1.5	9.8279 37E-01 9.8279 00E-01	*1E5*	1.5707 86E00 1.5707 81E00
2	1.1071 49E00 1.1071 43E00	*1E10*	1.5707 96E00 1.5707 93E00
3	1.2490 46E00 1.2490 41E00	*1E20*	1.5707 96E00 1.5707 93E00
5	1.3734 01E00 1.3733 95E00	*~1E50**	1.5707 96E00 1.5707 93E00

Note: *for negative x, results are the same but also negative*

*** ~1E100 = 9.999999999 E99*

x	arcsinh(x)	arccosh(x)	ln(x)	x	arcsinh(x)	arccosh(x)	ln(x)
0	0 0	-	-	*20*	3.6895 04E00 3.6895 28E00	3.6882 54E00 3.6882 48E00	2.9957 32E00 2.9957 03E00
0.25	2.4746 65E-01 2.4746 99E-01	-	-1.3862 94E00 -1.3863 06E00	*50*	4.6052 70E00 4.6052 69E00	4.6050 70E00 4.6050 98E00	3.9120 23E00 3.9120 28E00
0.5	4.8121 18E-01 4.8121 87E-01	-	-6.9314 72E-01 -6.9315 28E-01	*100*	5.2983 42E00 5.2983 96E00	5.2982 92E00 5.2983 47E00	4.6051 70E00 4.6052 12E00
1	8.8137 36E-01 8.8138 02E-01	0 0	*0* *0*	*1E3*	7.6009 03E00 7.6009 50E00	7.6009 02E00 7.6009 50E00	6.9077 55E00 6.9078 65E00
1.25	1.0475 93E00 1.0476 00E00	6.9314 72E-01 6.9315 28E-01	2.2314 36E-01 2.2314 35E-01	*1E4*	9.9034 88E00 9.9035 15E00	9.9034 88E00 9.9035 15E00	9.2103 40E00 9.2104 23E00
1.5	1.1947 63E00 1.1947 69E00	9.6242 37E-01 9.6243 74E-01	4.0546 51E-01 4.0546 92E-01	*1E5*	1.2206 07E01 1.2206 15E01	1.2206 07E01 1.2206 15E01	1.1512 93E01 1.1512 97E01
2	1.4436 35E00 1.4436 22E00	1.3169 58E00 1.3169 70E00	6.9314 72E-01 6.9315 28E-01	*1E10*	2.3719 00E01 2.3719 34E01	2.3719 00E01 2.3719 34E01	2.3025 85E01 2.3025 93E01
3	1.8184 46E00 1.8184 65E00	1.7627 47E00 1.7627 60E00	1.0986 12E00 1.0986 25E00	*1E20*	4.6744 85E01 4.6744 67E01	4.6744 85E01 4.6744 67E01	4.6051 70E01 4.6051 87E01
5	2.3124 38E00 2.3124 61E00	2.2924 32E00 2.2924 51E00	1.6094 38E00 1.6094 64E00	*1E50*	1.1582 24E02 1.1582 22E02	1.1582 24E02 1.1582 22E02	1.1512 93E02 1.1512 95E02
10	2.9982 23E00 2.9982 40E00	2.9932 23E00 2.9932 52E00	2.3025 85E00 2.3026 06E00	*~1E100***	-	-	2.3025 85E02 2.3025 90E02

Note: *for **arcsinh(x)** with negative x, results are the same but also negative*

**** ~1 = 0.9999999999*

x	arctanh(x)	x	arctanh(x)	x	arctanh(x)
0	0 0	*0.6*	0.6931 47 0.6931 53	*0.99*	2.6466 52 2.6466 74
0.1	0.1003 35 0.1003 37	*0.7*	0.8673 01 0.8673 07	*0.995*	2.9944 81 2.9944 78
0.2	0.2027 33 0.2027 35	*0.8*	1.0986 12 1.0986 25	*0.999*	3.8002 01 3.8001 99
0.3	0.3095 20 0.3095 21	*0.9*	1.4722 19 1.4722 11	*0.9999*	4.9517 19 4.9517 57
0.4	0.4236 49 0.4236 49	*0.95*	1.8317 81 1.8317 87	*0.99999*	6.1030 34 6.1030 74
0.5	0.5493 06 0.5493 12	*0.98*	2.2975 60 2.2975 63	*~1****	11.8594 99 11.8596 71

Note: *for negative x, results are the same but also negative*

Notes

1. The maximum accuracy obtained ultimately depends on the number of digits the calculator can carry (the *HP-16C* is a 10-digit calculator while the *HP-42S*, for instance, is a 12-digit calculator) and it also depends on the value of the constant *(K)* defined at steps *50-54*, which can be empirically fine-tuned to optimize the accuracy, and also depends itself on the number of digits and, to a lesser degree, on the particular function being evaluated.

   As using a different, specifically fine-tuned constant for each of the 7 functions would take some 35-40 additional program steps or more, the program slightly compromises by using the same constant *K* for all functions, which I carefully fine-tuned with the help of an *ad-hoc* routine I wrote to, given a candidate value for *K*, first compute the results for a selection of arguments covering the whole range, and then to accumulate the absolute value of the relative errors for each evaluation as compared to the theoretical value. The optimum value of *K* resulting in the smallest sum of relative errors is the one finally featured in the program.

   For 10-digit calculators such as the *HP-16C*, I found that *K = 0.000015* gives about **5-6** correct digits or better, while for 12-digit calculators a smaller *K = 0.000002* gives an extra correct digit, i.e.: about **6-7** correct digits or better. At any rate, the accuracy obtained is highly consistent throughout all seven functions and the whole extended ranges, as is the running time, which makes for a reliable, no-surprises experience. Also, for actual, real-world applications, which usually deal with experimentally obtained data, 5-6 digits should be adequate enough.

2. Input values outside the *Input Range* will result in an error condition. In the case of *Overflow* errors, when some intermediate computation results in a number with a magnitude greater than ±*9.999999999E99*, it will be automatically replaced with this value, flag 5 (the *out-of-range* flag) will be set and the **G** annunciator will be displayed, but the program will **not** stop, though the final result will most likely be invalid.

   On the other hand, if the input value causes an illegal mathematical operation (such as attempting division by zero or the square root of a negative number) ***Error 0*** (*Improper Mathematical Operation*) will be displayed and the program will immediately **stop**. In both cases, simply input a valid value and call the function again.

References

Valentín Albillo (2005) *HP Article VA018 - Long Live the HP-16C*

Valentín Albillo (2002) *HP Article VA004 - HP-12C Tried and Tricky Trigonometrics*

Copyrights