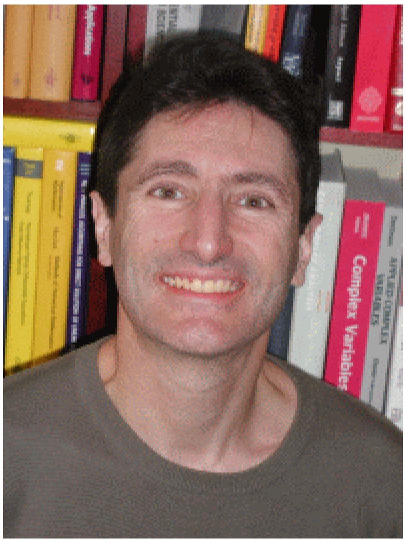
Best Black-Scholes in the HP-12C Galaxy!

Tony Hutchins, #1049

In DataFile V22N3 I attempted a history of HP calculator Black-Scholes programs, but I missed the *best* one, an absolute classic, from 1988, which can be found here: http://www.math.nyu.edu/research/carrp/papers/pdf/hp12cpgm.pdf



This was written by Dr. Peter Carr when he was a doctoral student at UCLA. Peter still uses an HP-12C with his program on it. He now heads up Quantitative Research at Bloomberg in New York and in 2003 was selected as Risk Magazine's prestigious "Quant of the Year." He also directs the Masters in Mathematical Finance program at NYU's Courant Institute. Hedging is his major focus, as you can tell from his published papers on his website. Peter sent me the photo, entitled "Big Smile".

Timothy Crack sent me a copy of this program in late January and I was astounded at the ingenuity of the code and the sheer scope of the functionality (it even *re-solves* for the implied volatility!). Since then my awe has only increased! What a gem this is. How did it remain hidden for so long? It struck me as a black hole might, suddenly hitting me from nowhere, and totally consuming all available resources on the HP-12C, not to mention all my attention.

I was determined to polish this gem, if I could. A.N. Whitehead's writings first made me see that limitations are required for any attainment, and the theoretical limitations of this program are fascinating. They are not really practical limitations, but I couldn't resist the challenge. In the appendix I list:

(1) The original program, which of course works fine as it is. This is reproduced from Peter's *hp12cpgm.pdf* - but the whole paper is essential reading.

(2) My modified version (same functionality and output as (1) but the output is rearranged).

HP-12C Hall of Fame

Peter's program has to be the first entrant. Look at the elegance of lines 9-32 which prepare d_1 and d_2! What about the ingenuity of lines 1-8 which use .4 as an initial guess for the isd, only if FV>=1? Also .4 is used to seed the vega - it exceeds $1/\sqrt{2\pi}$ by less than .3%. The distinction between big/small FV gives the program real character, as well as a seamless user interface. Lines 35-59 compute UTPN(|d|) (where "d" here stands for "d_1 or d_2") and the next 6 are sheer brilliance. Finally, note the cunning simplification of Stephen Derenzo's formula (refer to the paper for this). We have to be very grateful that Peter took the time to *publish* his work.

The five page paper explains clearly, in five steps, how to use the program. The input paradigm is very elegant: four observable variables (n, i, PV and PMT), and a fifth or final value (FV).

Example

This is taken from Peter's documentation. First we find the isd (implied standard deviation) for a call option with price $3.54 and strike $40:

.33[n] .12[i] 40[PV][ENTER][PMT] 3.54[FV][R/S], see 0.3003, [R/S], see 0.2999. We take .3 as the isd and now calculate the price for a call with strike $35:

35[PMT] .3[FV][g][GTO]00[R/S], see 6.88=Call, [RCL]3, see 0.52=Put, [RCL]0, see 5.07=Vega (slightly high, by .01, due to the .4 vega seed mentioned above).

The first part had FV>1, and the second FV<1. The first FV was interpreted by this magic program as a Call value, and the second as an isd.

The [g][GTO]00 above is only needed when *switching* from a big FV to a small one. In the above case if we stored the *calculated* isd *directly* in FV, the [g][GTO]00 is not actually necessary. But in general, before starting an FV<1 *just after* an FV>=1 calculation the program pointer should be reset.

Suppose you want to try a very large isd in the above case: .99[FV][R/S], see 11.79. If you want larger then [RCL][n][g][12x][g][12x], [RCL][i][g][12÷][g][12÷] increases the isd by a factor of 12, even though FV stays at .99. [R/S], see 39.98, showing we are approaching the limit of 40. You can return to the original n & i by doing [RCL][g][12x][n] twice and [RCL][g][12÷][i] twice. I admit the re-scaling by 10 is probably easier, but it is fun to use 12!

Polishing

1. Re-solving for volatility >1

4[n] .05[i] 37.5[PV] 100[PMT] 20[FV][R/S], see 1.043, [R/S], see 1.043, ad infinitum. The correct results are: 1.043, 0.974, 0.976, ... This was fixed by replacing line 97 with [CLx]. Peter described this situation clearly in an e-mail:

As you know, my program uses the magnitude of the value put in FV to decide whether the user wants an implied vol or an option price. If the number put in FV is <1, then the program assumes the user wants an option price and if the number put in FV is > 1 then the program assumes the user wants an implied vol.

If the market call price is less than $1 and the user wants an implied, then I suggested scaling up the spot and strike by say $100, so that the call price would also scale up to over $1.

If I understand you correctly, you are saying that if the user puts a number >1 in FV indicating he wants an implied vol and if that implied vol is actually over 100%, then there is a bug in the code and the proper implied vol is not eventually calculated.

You also suggest a fix. Is that correct?

2. FV=1 equivalent to FV=.4

6[n] .05[i] 2[PV]10[PMT] 1[FV][R/S], see 0.15, which is equivalent to .4[FV][R/S]. This would be fine if the X<=Y at line 87 were X<Y. As it is it takes 2 extra lines to fix. The correct result here is an isd guess (1.03 then .85, .87, ...) to give Call=1.

3. Error 0 when d=0

16[n] .08[i] 100[PV][ENTER][PMT] .4[FV][R/S] gives "error 0" at line 47. One extra line will fix this.

4. Overrun error for |d|>15.17

This can happen at line 54 as EXP(d^2) then exceeds E100. It can be fixed with a CHS but then the main loop needs to actually cope with cases where both $N(d_1)$ and $N(d_2)=0$. As it is the overrun actually prevents looping. The total cost is about 3 extra lines. The loop itself would be fine if the X<=Y at line 69 were X<Y. But, without X<Y we basically need 2 extra X<>Y to fix it. N(d)=0 (on our HP-12C) only happens theoretically when d < −21.2 (where N(d)<E−99) - to replicate that we would need to compute EXP(.5 x d^2) instead of SQRT (EXP(d^2))... which costs a line and turns out to be pointless anyway (vide infra).

6[n] .05[i] 50[PV] 100[PMT] .02[FV][R/S] gives 1.6E−16, .01[FV][R/S] gives the "9.999999 99" overrun: d_1 and d_2 are now less than −15.17 ($N(d_1)$ and $N(d_2)=0$). Another example, where d_1 and d_2 exceed 15.17, is given in the next section.

5. Looping for d_1 and d_2 between ~6.3 and 15.17

This is similar to item 4 and happens when $N(d_1)=N(d_2)=1$, or slightly less - both can evaluate as 0.999999999 for example ([f][PREFIX] is required to see this), on our 10 digit machine. For both d>6.48 (UTPN(6.48)≈5E−11) both N(d) evaluate as 1 exactly. In fixing this I did manage to *save one* line, but I couldn't decide what to do with it.

6[n] .05[i] 100[PV][ENTER][PMT] .02[FV][R/S], see 25.92, but .019[FV][R/S] never stops unless a key is pressed. Do that, then reset the program pointer to line 00 and then .001[FV][R/S] gives the overrun error (d_1 and d_2 are over 15.17).

5a. Progress report

Well, one line doesn't help much. I didn't want to remove any features - for example the Call delta can be removed and saves 2 lines.

I kept looking a different parts of the code but whole sections are already polished obsidian!! But then I played with the Normal approximation and found I could save 2 lines and only have slightly impaired accuracy. No good enough. Then about a week later I looked at it again and found I could save another line *and* slightly improve the accuracy!! However I was still short of 2 lines. So I tried all possible ways of doing the d-loop and didn't really get anywhere - almost every permutation left me 2 lines over. Then by a sheer stroke of luck I tried a mixture of methods - doing $N(d_1)$ before $N(d_2)$ and using $N(d_1)$ −1 as the criterion for exit. It was as if the Put delta wanted to *contribute* rather than just be done as a separate calculation. This gave a really nice short loop test and end game where the Call and Put are calculated and I couldn't believe it when the line count showed I had the 2 lines!! The output is now arranged differently. I need to store the $N(d_1)$ in R_5 as at that point, right between the two loops, all other registers are busy, carrying important data. So, that meant I really had to put the Call in R_4, and then I thought the best place for the Put and Put delta is right underneath the Call data, in R_1 and R_2. R_3 just holds d_2 by default - quite handy to review though, as d_1 and d_2 are not displayed. d_1 is easily recovered with RCL n g √x RCL FV × RCL 3 +.

Using $N(d_1)-1$ *in* the option value calculations, rather than $N(d_1)$ as in (1), immediately limits transmission of a small $N(d_1)$ (under 5E−11) into the option calculations and effectively means we now use a normal distribution truncated at not only +6.48 (as before) , but also at −6.48. The program does at least *handle* any normal variate (d). And the original $N(d_1)$ is stored in R_5, after all. As an example the modified version gives 0 exactly for the example in section 4 above, where the original gives 1.6E−16. So, in finding a way to free up a limitation I have introduced a new one, but at least it roughly harmonises with the range published for the Derenzo approximation (|d|<5.5).

6. Accuracy

In my previous Black Scholes program I used an approximation with an absolute error range [-1.63E-5, 1.01E-5]. Peter uses a simplification of the Derenzo formula. In units of E-5, the Derenzo has error range [-4.78, 7.18]. Peter's simplification has error range [-12.37, 9.14]. In version (2) I use this simplification of the Derenzo (saving 3 keystrokes) which has a tighter error range [-9.14,5.62]:

$$Q(x) = \tfrac{1}{2}\cdot EXP(-\tfrac{1}{2}\cdot x^2 - 4\cdot x/(x/.85 + 5))$$

where Q(x) gives UTPN(x) (upper tail normal probability), where x >=0. The formula is written with an extra 'x' to avoid an error when x=0.

Epilogue

I am so pleased I was able to produce (2) which is closer to (1) than I expected - only the output is shifted - the 5 main outputs of vega, Call/Put values and deltas are all there. And I fixed everything I could find. A lot of fun!

What a piece of living history! It's as if a secret tradition in 12C programming has been uncovered. I always felt that one day I'd see some special magic in the 12C. Back in 1988, Peter wrote:

Although handheld computers are fast approaching calculators in portability and price, many calculators will continue to be used. Hopefully this program will allow the venerable HP12C to compete with these new computers.

And, recently Peter wrote:

I'm especially pleased you were able to polish the program. If it means anything to you, I plan to put your program on my HP12C.

Thanks Peter. Indeed the venerable 12C is still alive!

Appendix, part (1), the original program.

n	i	PV	PMT	FV	R_4: PutΔ	R_5: used	R_6: isd
Option term	Interest rate	Asset spot price	Opt. strike price	<1: isd >=1:call	R_1: Call	R_2: CallΔ	R_3: Put
					R_0: Vega		

(1)Press	Display	(1)Press	Display	(1)Press	Display
RCL FV	01- 45 15	RCL 1	34- 45 1	x≷y	67- 34
STO 6	02- 44 6	STO 4	35- 44 4	STO 2	68- 44 2
g INTG	03- 43 25	STO × 4	36-44 20 4	g x≤y	69- 43 34
•	04- 48	2	37- 2	g GTO 34	70-43,33 34
4	05- 4	8	38- 8	STO 4	71- 44 4
STO 0	06- 44 0	1	39- 1	RCL PV	72- 45 13
g x≤y	07- 43 34	CHS	40- 16	×	73- 20
STO 6	08- 44 6	ENTER	41- 36	x≷y	74- 34
RCL PV	09- 45 13	3	42- 3	RCL 3	75- 45 3
STO × 0	10-44 20 0	5	43- 5	×	76- 20
RCL PMT	11- 45 14	1	44- 1	−	77- 30
RCL n	12- 45 11	RCL 4	45- 45 4	STO 1	78- 44 1
RCL i	13- 45 12	g √x	46- 43 21	RCL PV	79- 45 13
×	14- 20	÷	47- 10	−	80- 30
g e^x	15- 43 22	8	48- 8	STO + 3	81-44 40 3
÷	16- 10	3	49- 3	RCL 5	82- 45 5
STO 3	17- 44 3	+	50- 40	STO ÷ 0	83-44 10 0
÷	18- 10	÷	51- 10	1	84- 1
g LN	19- 43 23	g e^x	52- 43 22	STO − 4	85-44 30 4
RCL 6	20- 45 6	RCL 4	53- 45 4	RCL FV	86- 45 15
RCL n	21- 45 11	g e^x	54- 43 22	g x≤y	87- 43 34
g √x	22- 43 21	g √x	55- 43 21	g GTO 99	88-43,33 99
STO × 0	23-44 20 0	STO 5	56- 44 5	RCL 1	89- 45 1
×	24- 20	÷	57- 10	−	90- 30
STO 1	25- 44 1	2	58- 2	RCL 0	91- 45 0
÷	26- 10	÷	59- 10	÷	92- 10
g LSTx	27- 43 36	g INTG	60- 43 25	RCL 6	93- 45 6
2	28- 2	g x≤y	61- 43 34	+	94- 40
STO 2	29- 44 2	1	62- 1	STO 6	95- 44 6
÷	30- 10	g LSTx	63- 43 36	R/S	96- 31
−	31- 30	g x≤y	64- 43 34	RCL 6	97- 45 6
STO + 1	32-44 40 1	−	65- 30	g GTO 03	98-43,33 03
g GTO 35	33-43,33 35	RCL 2	66- 45 2	RCL 1	99- 45 1

Appendix, part (2), the modified program.

n	i	PV	PMT	FV	R_4: Call	R_5: CallΔ	R_6: isd
Option term	Interest rate	Asset spot price	Opt. strike price	<1: isd >=1:call	R_1: Put	R_2: PutΔ	R_3: d_2
					R_0: Vega		

(2)Press	Display	(2)Press	Display	(2)Press	Display
RCL FV	01- 45 15	STO X 4	34-44 20 4	STO 5	67- 44 5
STO 6	02- 44 6	4	35- 4	1	68- 1
g INTG	03- 43 25	RCL 4	36- 45 4	−	69- 30
•	04- 48	g √x	37- 43 21	STO 2	70- 44 2
4	05- 4	X	38- 20	RCL 4	71- 45 4
STO 0	06- 44 0	g LSTx	39- 43 36	STO X 0	72-44 20 0
g x≤y	07- 43 34	•	40- 48	RCL 3	73- 45 3
STO 6	08- 44 6	8	41- 8	g GTO 33	74-43,33 33
RCL PV	09- 45 13	5	42- 5	RCL PV	75- 45 13
STO X 0	10-44 20 0	÷	43- 10	STO 4	76- 44 4
RCL PMT	11- 45 14	5	44- 5	X	77- 20
RCL n	12- 45 11	+	45- 40	x≷y	78- 34
RCL i	13- 45 12	÷	46- 10	RCL 1	79- 45 1
X	14- 20	CHS	47- 16	X	80- 20
g e^x	15- 43 22	g e^x	48- 43 22	−	81- 30
÷	16- 10	RCL 4	49- 45 4	STO + 1	82-44 40 1
STO 1	17- 44 1	CHS	50- 16	STO + 4	83-44 40 4
÷	18- 10	g e^x	51- 43 22	RCL FV	84- 45 15
g LN	19- 43 23	g √x	52- 43 21	g INTG	85- 43 25
RCL 6	20- 45 6	STO 4	53- 44 4	g x=0	86- 43 35
RCL n	21- 45 11	X	54- 20	g GTO 99	87-43,33 99
g √x	22- 43 21	2	55- 2	g LSTx	88- 43 36
STO X 0	23-44 20 0	÷	56- 10	RCL 4	89- 45 4
X	24- 20	g INTG	57- 43 25	−	90- 30
÷	25- 10	g x≤y	58- 43 34	RCL 0	91- 45 0
STO 3	26- 44 3	1	59- 1	÷	92- 10
g LSTx	27- 43 36	g LSTx	60- 43 36	RCL 6	93- 45 6
2	28- 2	g x≤y	61- 43 34	+	94- 40
STO 2	29- 44 2	−	62- 30	STO 6	95- 44 6
÷	30- 10	RCL 2	63- 45 2	R/S	96- 31
STO − 3	31-44 30 3	g x≤y	64- 43 34	CLx	97- 35
+	32- 40	g GTO 75	65-43,33 75	g GTO 03	98-43,33 03
STO 4	33- 44 4	x≷y	66- 34	RCL 4	99- 45 4

Appendix, part (2), keystrokes for the sensitivities.

Vega: [RCL]0
Vega is the sensitivity of the option price to the volatility. Also, our solar system is speeding through space in the direction of Vega, the fifth brightest star in the sky, of magnitude 0.0, in the constellation Lyra (called *Vultur cadens* or Swooping Vulture two centuries ago). The Arabs' title for the constellation was *Al Nasr al Waki* (referring to the swooping Stone Eagle of the desert). Anyway Vega derives from the Arabic *Waki* and is definitely not Greek, but the sensitivities are collectively called "greeks". Having vega we can calculate the gamma and thetas - another stellar achievement of Peter's 12C program!
Gamma: [RCL]0[RCL][FV][÷][RCL][n][÷][RCL][PV][ENTER][X][÷]
The next 4 are different for call and put options. The call sensitivities are listed. To obtain the corresponding values for the put just replace [RCL]**5** and [RCL]**4** (shown bold below) by [RCL]2 and [RCL]1 respectively.
Call Delta: [RCL]**5**
Call Lambda: [RCL]**5**[RCL][PV][X][RCL]**4**[÷]
Lambda is the option *leverage*, the ratio of the percentage change in the option price to the percentage change in [PV], the underlying price. The delta gives the sensitivity to the underlying, and the gamma is the sensitivity of the delta to the underlying.
Call Rho: [RCL]**5**[RCL][PV][X][RCL]**4**[−][RCL][n][X]
Call Theta: [RCL]**5**[RCL][PV][X][RCL]**4**[−][CHS][RCL][i][X]
[RCL]0[RCL][FV][X][RCL][n][÷]2[÷][−]
Rho and theta are the sensitivities to [i] and the elapse of time (reduction in [n]) respectively. Theta is commonly divided by 365 or 252 (trading days in a year). Vega and rho are commonly divided by 100. Taking an example from Peter's paper: .33[n] .12[i] 40[PV][ENTER][PMT] .3[FV][R/S], see 3.54. We find:

<table>
<tr><th></th><th>Value</th><th>Vega</th><th>Gamma</th><th>Delta</th><th>Lambda</th><th>Rho</th><th>Theta</th></tr>
<tr><td>Call</td><td>3.54</td><td rowspan="2">8.74</td><td rowspan="2">0.0552</td><td>0.624</td><td>7.05</td><td>7.07</td><td>-6.55</td></tr>
<tr><td>Put</td><td>1.99</td><td>-0.376</td><td>-7.57</td><td>-5.62</td><td>-1.92</td></tr>
</table>

The following examples of small sensitivity tests illustrate usage of the above.
Vega: If [FV] increases by .01, the call and put values increase by ~$0.09.
Gamma: If [PV] increases by $1.00, the call and put *deltas* increase by ~0.06
Call delta: If [PV] increases by $0.10, the call value increases by ~$0.06.
Call lambda: If [PV] increases by 1% ($40 to $40.40) the call value increases by ~7.07%, to ~$3.79.
Call rho: If [i] increases by .01, the call value increases by ~$0.07.
Call theta: If [n] n decreases by 1/52 (i.e. in a week) the call value *reduces* by ~6.55/52 or $0.13, to ~$3.41. Let's test this: [RCL][n]52[1/x][−][n][R/S], see 3.41.
Note that vega, gamma and the call theta above are slightly affected in the 3rd significant digit by the .4 used in vega (correct values are 8.72, .0551 and -6.53 respectively). All other values are correct as shown.

Appendix, part (2a), accuracy.

It is possible to extend the accuracy of (2) by one significant digit (cost=17 lines), but only at the expense of *removing* the built-in Newton-Raphson iteration for the isd, which frees up 20 lines, but of course removes the principal feature (FV is now just the isd). In version (2a), the 3 bonus lines were used as follows:

-	$N(d_2)$ is stored in R_6 (the only difference in output).
-	d_1 and d_2 are displayed, during execution.
-	[x≷y] shows the Put value, at the end.

(2a) uses the normal approximation from V22N3. The vega seed is .399 instead of .4 (error reduced from +.265% to +.0145%) - this ensures more accurate vega, gamma and thetas, commensurate with the new normal approximation. The following table shows an error comparison. The values are obtained thus:

4[n] .05[i] 37.5[PV] 100[PMT] .5[FV][R/S]

[RCL]2 to see Call delta in version (1), [RCL]5 in (2) and (2a).

Version	Call Value	Error	Call Delta	Error
(1)	6.3948	-.0108	.389301	-.000120
(2)	6.3979	-.0077	.389334	-.000097
Derenzo	6.4014	-.0042	.389394	-.000027
(2a)	6.4067	+.0011	.389416	-.000005
Exact	6.4056	-	.389421	-

Rounding the Call value to the nearest cent we see that (2a) agrees with the exact figure of 6.41. The Derenzo, and (2) round to 6.40, whereas (1) rounds to 6.39, even though the underlying figure is hardly more than 1 cent out.

The *drawback* of (2a) is of course that isd iteration is only possible manually. First *key in an isd guess*, press [FV][R/S] then repeat the following until the target is achieved: choose whether the call or put is targeted (use [x≷y] or [RCL]4 or [RCL]1) then *key in the target value* and press: [−][CHS][RCL]0[÷][RCL][FV][+][FV][R/S].

Version (2a) does calculate vega, labelled in textbooks as ν, which looks remarkably like the Greek letter "Nu". The vega does almost *sound* Greek though, and possibly rhymes with omega. Vega was not in the old V22N3 version, but thanks to Peter it does fit into version 2(a). The sensitivity keystrokes are the same as for (2), but the rho and theta keystrokes can be shortened considerably in (2a) by first setting up R_3 and R_6: [RCL]1[RCL]4[−][RCL][PV][+][STO]3[RCL]6[×][STO]6.

Then the call rho is: [RCL]6[RCL][n][×] and the put rho: [RCL]6[RCL]3[−][RCL][n][×]. The call theta is: [RCL]6[CHS][RCL][i][×][RCL]0[RCL][FV][×][RCL][n][÷]2[÷][−] and the put theta is a continuation: [RCL]3[RCL][i][×][+]. If desired (2a) can be changed to do the above setup of R_3 and R_6 by using the 3 bonus lines differently, for example lines 92-99 could be: [RCL]1[STO]3[×][STO]6[−][STO][+]1[STO][+]4[RCL]4. Also, [−][CHS][RCL]6[÷][RCL][n][÷][RCL][i][+][i][R/S] can then be used to re-solve for [i], given a *target Call value*, using a method similar to that above for the isd.

Appendix, version (2a).

n	i	PV	PMT	FV	R_4: Call	R_5: CallΔ	R_6: N(d_2)
Option term	Interest rate	Asset price	Opt. price	isd	R_1: Put	R_2: PutΔ	R_3: d_2
					R_0: Vega		

(2a)Press	Display	(2a)Press	Display	(2a)Press	Display
RCL PV	01- 45 13	STO 4	34- 44 4	1	67- 1
STO 0	02- 44 0	x≷y	35- 34	g LSTx	68- 43 36
RCL PMT	03- 45 14	3	36- 3	g x≤y	69- 43 34
RCL n	04- 45 11	•	37- 48	−	70- 30
RCL i	05- 45 12	0	38- 0	RCL 2	71- 45 2
×	06- 20	0	39- 0	g x≤y	72- 43 34
g e^x	07- 43 22	6	40- 6	g GTO 88	73-43,33 88
÷	08- 10	÷	41- 10	x≷y	74- 34
STO 1	09- 44 1	1	42- 1	STO 5	75- 44 5
÷	10- 10	+	43- 40	1	76- 1
g LN	11- 43 23	1/x	44- 22	−	77- 30
RCL FV	12- 45 15	×	45- 20	STO 2	78- 44 2
RCL n	13- 45 11	g LSTx	46- 43 36	RCL 4	79- 45 4
g √x	14- 43 21	g LSTx	47- 43 36	•	80- 48
STO × 0	15-44 20 0	1	48- 1	3	81- 3
×	16- 20	8	49- 8	9	82- 9
÷	17- 10	7	50- 7	9	83- 9
STO 3	18- 44 3	×	51- 20	×	84- 20
g LSTx	19- 43 36	2	52- 2	STO × 0	85-44 20 0
2	20- 2	4	53- 4	RCL 3	86- 45 3
STO 2	21- 44 2	−	54- 30	g GTO 25	87-43,33 25
÷	22- 10	×	55- 20	RCL PV	88- 45 13
STO − 3	23-44 30 3	8	56- 8	STO 4	89- 44 4
+	24- 40	7	57- 7	×	90- 20
g PSE	25- 43 31	+	58- 40	x≷y	91- 34
STO 6	26- 44 6	×	59- 20	STO 6	92- 44 6
ENTER	27- 36	•	60- 48	RCL 1	93- 45 1
×	28- 20	2	61- 2	×	94- 20
g √x	29- 43 21	%	62- 25	−	95- 30
g LSTx	30- 43 36	RCL 6	63- 45 6	STO + 1	96-44 40 1
CHS	31- 16	x≷y	64- 34	STO + 4	97-44 40 4
g e^x	32- 43 22	g INTG	65- 43 25	RCL 1	98- 45 1
g √x	33- 43 21	g x≤y	66- 43 34	RCL 4	99- 45 4