The New SopLog Function

By
Namir C. Shammas

Contents

Introduction

This study presents a new mathematical function, the *SopLog*. This function is defined as a special power summation logarithm:

$$\mathrm{SopLog}_N(S) = x \tag{1}$$

Where the integer N is the base of the function and S is a summation defined as:

$$S = \sum_{i=1}^{N} i^x \qquad (2)$$

Where x is a power than can be a non-integer. The above summation was inspired by the integer summation problem the young Gauss solved at school using the following famous equation:

$$S = \sum_{i=1}^{N} i = (N+1)N/2 \qquad (3)$$

Equation (3) yields the sum of the first N integers, each raised to the power 1. By contrast, equation (2) asks the question, "What power x should each of the first N integers be raised to and then added to yield the sum of S?" In other words, equation (2) has N and S as the input, and x is the output calculated by iterations using equation (2) in a root-seeking algorithm. Plotting the values of x vs S (for a fixed N) yields a semi-log plot, as shown by the following sample graph:

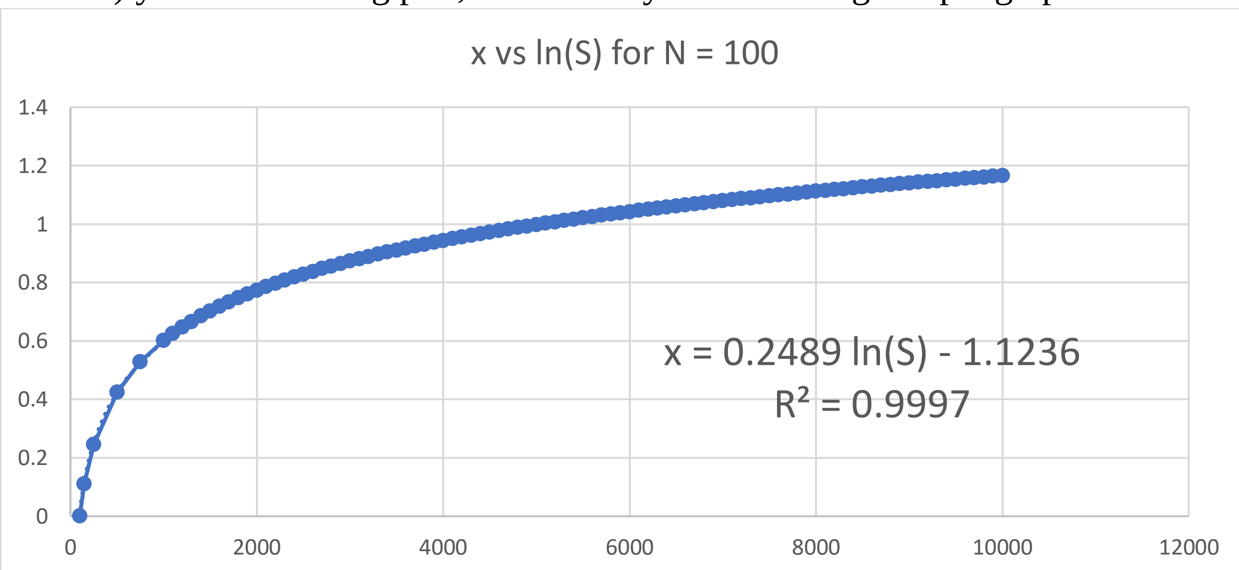

Figure 1. Sample plot of x vs ln(S).

The above graph has led me to name the function SopLog (short for *Sum of Powers Logarithm*, since the names *Summation Logarithm* and *Slog* were already used in other web-published math studies).

The SopLog function has the following special cases:

1. $\text{SopLog}_N(S) = 0$ for N = S
2. $\text{SopLog}_N(S) = 1$ for $S = \sum_{i=1}^{N} i$

Unlike with popular logarithms, the base N for the SopLog function must always be an integer since it represents the number of terms in a series.

The SopLog function is not a complicated function to calculate but it can require a lot of CPU time for large values of N on slow machines, such as vintage programmable calculators. To evaluate it and obtain the power x (as in equation (2)) you need to solve for the root x of the following equation:

$$f(x) = S - \sum_{i=1}^{N} i^x \tag{4}$$

Using an efficient root-seeking method, such as Newton's method. Keep in mind that as N grows bigger, more summation terms need to be calculated and added. If you choose another root-seeking algorithm, such as Halley' method keep in mind that the number of function calls per iteration influences the computational time.

The Excel VBA Code

I present the code to calculate the summation for SopLog and the implementation of equation (4) to calculate x for the given values of N and S. Here is the VBA code (placed in the workbook's module to make it accessible to all cells in every workbook):

```
Option Explicit

Function SumFx(ByVal N As Integer, ByVal S As Double ByVal Pwr
As Double,) As Double
  Dim I As Integer
  Dim Sum As Double

  Sum = 1
  For I = 2 To N
    Sum = Sum + I ^ Pwr
  Next I
  SumFx = S - Sum

End Function

Function SopLog(ByVal N As Integer, ByVal S As Double, _
    Optional ByVal Toler = 0.00000001) As Double
  Dim X As Double, r As Integer
  Dim h As Double, Diff As Double, F0 As Double


  If N = S Then
    SopLog = 0
    Exit Function
  End If
```

```
  If S = Fix(N * (N + 1) / 2) Then
    SopLog = 1
    Exit Function
  End If

  r = 1
  X = 1
  Do
    h = 0.0001 * (1 + Abs(X))
    F0 = SumFx(N, S, X)
    Diff = h * F0 / (SumFx(N, S, X + h) - F0)
    X = X - Diff
    r = r + 1
  Loop Until Abs(Diff) < Toler Or r > 1000
  SopLog = X
End Function
```

The function SumFx calculates the difference between the targeted sum S and the summation of the N terms of i raised to the power x. The function SopLog calculates x for the given values of N and S (and a small tolerance). The function SopLog implements Newton's method to calculate the value of x that yields a very small value of f(x) in equation (4). The function SopLog handles the two cases where N and S are equal and where S is the sum of N integers. If either condition is true, the function returns the appropriate results without going through Newton's method. The function performs up to 1000 iterations in Newton's method. This limit is a safeguard against infinite looping. You can edit the listing and alter this limit to a lower value.

Throughout this study I initialize x with 1 in Newton's method. An alternative initialization scheme is to set x to $\log_{10}(S/N)$. This scheme seems to work better in vintage calculators since it gives an initial guess that is closer to the converged value of x.

The various Excel worksheets (for difference values of N) that obtain the results use the following code to calculate the values of x for rows of N and S values:

```
Sub go()
  Dim r As Integer, N As Integer, S As Double, Toler As Double

  r = 2
  Toler = 0.0000000001
```



```
  Do Until IsEmpty(Cells(r, 1))
    N = Cells(r, 1)
    S = Cells(r, 2)
    Cells(r, 3) = SopLog(N, S, Toler)
    r = r + 1
  Loop
End Sub
```

In each worksheet, column A has the values of N. Column B has the values of S. Column C has the calculated values of x. Here is a sample partial view of a typical worksheet:

	A	B	C
1	N	Sum	x
2	100	100	0
3	100	150	0.110121
4	100	250	0.245589
5	100	500	0.424944
6	100	750	0.527995
6	100	1000	0.600423
7	100	1100	0.624305
8	100	1200	0.646061
9	100	1300	0.666037

Table 1. Sample partial Excel worksheet view.

You can also place the implementations for functions sumFx and SopLog in a module used by the workbook. This approach allows you to invoke the function SopLog like it was a built-in Excel function. For example if cells A2 and B2 contain the values for N and S, respectively, you can type in "=SopLog(A2,B2)" in cell C2 to obtain the corresponding value of x.

The Results

This section presents the summary of results of the calculations. If you are interested in looking at the source data, then you can browse through section "The Detailed Data" with its multiple tables.

Earlier I mentioned that plotting x vs S (for a fixed N value) shows a logarithmic plot in the form $x = b \ln(S) - a$. The next subsections look at the relation between the slope, b, and N, and between the intercept, a, and N.

The Slope b vs N

The following table shows the results of N vs the slope b:

N	Slope
10	0.4821
50	0.2916
100	0.2489
250	0.2086
500	0.1859
750	0.1748
1000	0.1677
2500	0.1486
5000	0.1368
7500	0.1308
10000	0.1268

Table 2. N vs Slope b.

The plot of y (=$\log_{10}$(Slope b)) vs x (=$\log_{10}(\log_{10}(N))$) is:

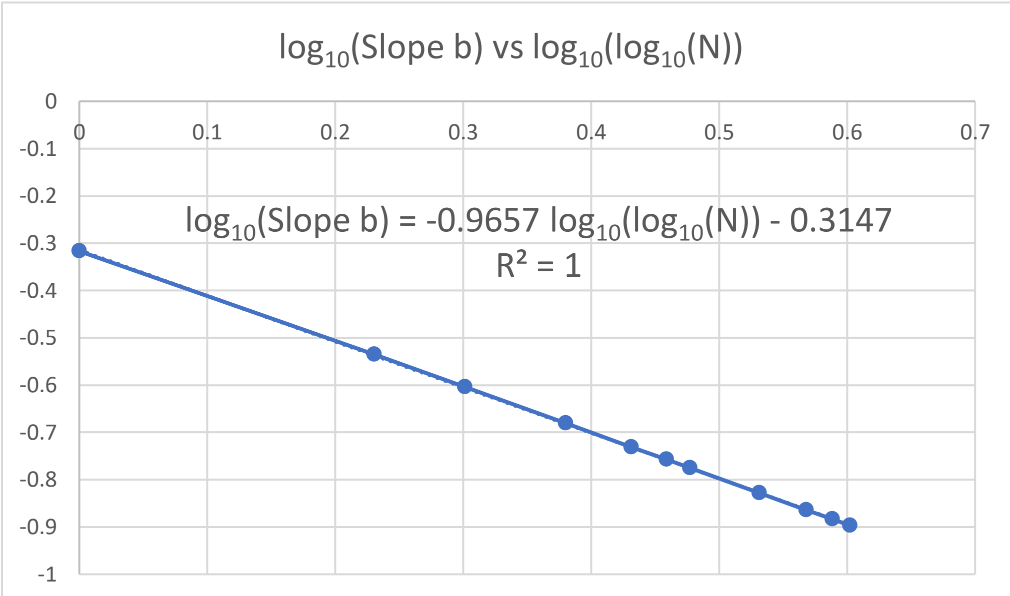

Figure 2. log10(Slope b) vs log10(log10(N)).

I used Excel's regression utility to perform a linear regression between $\log_{10}(S)$ and $\log_{10}(\log_{10}(N))$. Here is the resulting regression ANOVA table:

SUMMARY OUTPUT						
Regression Statistics						
Multiple R	0.999980185					
R Square	0.99996037					
Adjusted R Square	0.999955966					
Standard Error	0.001161581					
Observations	11					
ANOVA						
	df	SS	MS	F	Significance F	
Regression	1	0.306406177	0.306406177	227090.2976	4.01705E-21	
Residual	9	1.21434E-05	1.34927E-06			
Total	10	0.30641832				
	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%

Intercept	-0.314706469	0.000911418	-345.2932238	7.29532E-20	-0.31676824	-0.312644698
Log(log(N))	-0.965654959	0.002026388	-476.5399224	4.01705E-21	-0.970238968	-0.96107095

The well fitted relationship is:

$$\log_{10}(\text{Slope b}) = -0.314706469 - 0.965654959 * \log_{10}(\log_{10}(N)) \qquad (5)$$

The Absolute Intercept a vs N

Here is the table showing the absolute values of the intercept vs N:

N	Abs Intercept
10	0.8687
50	1.0903
100	1.1236
250	1.1472
500	1.1567
750	1.1602
1000	1.1621
2500	1.1652
5000	1.1657
7500	1.1655
10000	1.1652

Table 3. N vs absolute intercept.

Plotting the absolute intercept vs $\log_{10}(N)$ we get the plot:

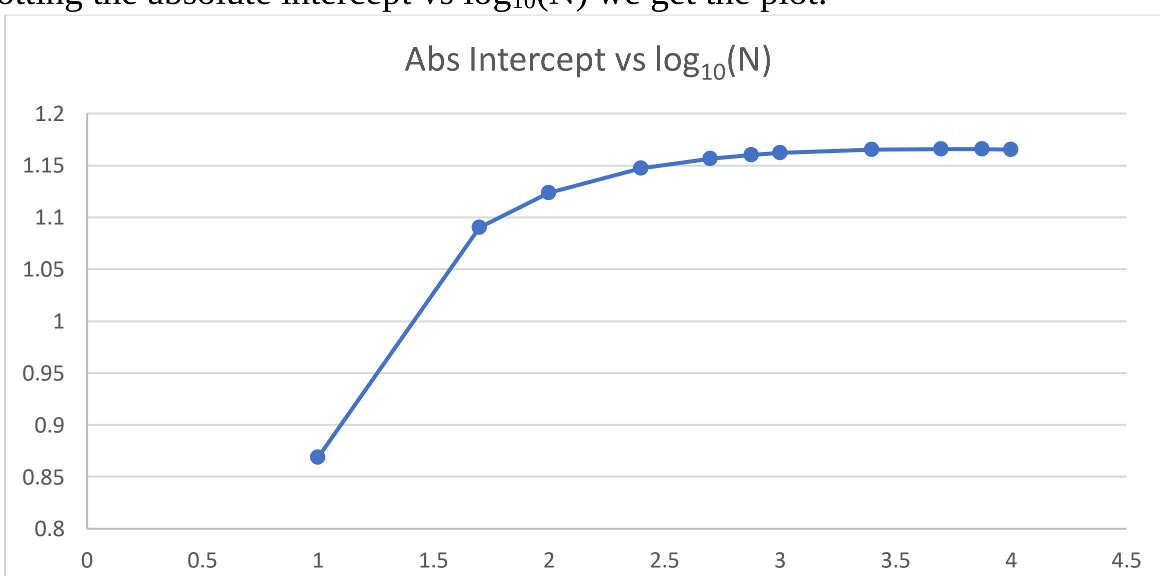

Figure 3 absolute intercept vs log10(N).

The above graph suggests a crescent shape curve exists with the following model:

Abs Intercept a = A * (1 – exp(–k * $\log_{10}$(N)) (6)

I used the Excel Solver to calculate the values for A and k by minimizing the sum of the differences squared between the observed intercept values and the ones calculated using equation (6). The results are:

Abs Intercept a = 1.1689028 * (1 – exp(–1.6467435 * $\log_{10}$(N)) (6b)

The asymptotic value for the absolute intercept is 7/6. Thus, you can approximate x for given S and N values using:

x = b ln(S) – a (7)
b = 10^(–0.314706469 – 0.965654959 * $\log_{10}$($\log_{10}$(N)))
a = 1.1689028 * (1 – exp (–1.6467435 * $\log_{10}$(N))

How do the approximations of equation (7) perform? The next table shows sample values for N and S. The table shows the values of x calculated using the SopLog function and the approximation of equation (7). The last column shows the percentage error:

N	S	x	x est	% Err
100	5000	0.997579	0.987489	1.01145
100	7500	1.095956	1.088079	0.718692
100	10000	1.16543	1.159449	0.513228
500	5000	0.42757	0.426805	0.178824
500	7500	0.500854	0.502116	-0.25199
500	10000	0.552589	0.55555	-0.53591
750	5000	0.329462	0.329684	-0.06741
750	7500	0.398329	0.400536	-0.55429
750	10000	0.44694	0.450807	-0.8651
1000	5000	0.267188	0.267877	-0.25801
1000	7500	0.333231	0.335878	-0.79418
1000	10000	0.379848	0.384125	-1.12602

Table 4. Comparing calculate x vs estimate x values.

The above table shows that the absolute percentage error stays below 1.15%, which reflects well on equation (7). This is a good tradeoff between speed and accuracy.

Sample Values for Testing

Before I present implementations in various programming languages and for various HP calculators, I present a table for sample values (calculated using the MATLAB implementation) that you can use in testing the various implementations of SopLog:

N	S	x = $SopLog_N(S)$
10	150	1.525648647035608
10	500	2.129188323991269
10	1000	2.467997575374322
50	150	0.355611870132926
50	500	0.723423668944274
50	1000	0.928822384903918
100	500	0.424943969992081
100	1000	0.600422799908712
500	1000	0.131262567839523
500	5000	0.427569851940243
500	10000	0.552588884953760
1000	5000	0.267187615565215
1000	10000	0.379847559482939

Table 5. Sample values for N, S, and x.

Implementation in MATLAB

This section presents a MATLAB implementation for the function SopLog:

```
function x = soplog(N, S)
%SOPLOG implements the SopLog function.

  toler=1e-8; % default tolerance
  x = 1; % initial guess for x
  maxiter = 1000;
  for iter =1:maxiter
    h = 0.001 * (1 + abs(x));
    f0 = sumFx(N, S, x);
    fp = sumFx(N, S, x + h);
    fm = sumFx(N, S, x - h);
    diff = 2 * h * f0 / (fp - fm);
```



```
    x = x - diff;
    if abs(diff) < toler, break; end
  end
end

function y =sumFx(N,S,x)
  sumx = 1;
  for i=2:N
    sumx = sumx + i^x;
  end
  y = S - sumx;
end
```

Implementation in Python

This section presents a Python implementation for the function SopLog:

```
# soplog implementation

def sumFx(N, S, x):
  sumx = 1
  for i in range(2, N+1):
    sumx += i ** x
  return S - sumx

def soplog(N, S):
  toler=1e-8
  x = 1
  maxiter = 1000
  for iter in range(1, maxiter+1):
    h = 0.001 * (1 + abs(x))
    f0 = sumFx(N, S, x)
    fp = sumFx(N, S, x + h)
    fm = sumFx(N, S, x - h)
    diff = 2 * h * f0 / (fp - fm)
    x -= diff
    # print(x)
    if abs(diff) < toler:
      break
  return x
```

Implementation in C++

This section implements the SopLog function in C++:

```
#include <iostream>
#include <cmath>
```



```
using namespace std;

double sumFx(long N, long S, double x)
{
    double sum = 1;

    for (long i = 2; i <= N; i++) {
        sum += pow(i, x);
    }
    return S - sum;
}

double soplog(long N, long S)
{
    double toler = 1e-8;
    double x = 1;
    double h, diff, f0, fp, fm;
    int iter, maxiter = 1000;

    for (iter = 1; iter <= maxiter; iter++)
    {
        h = 0.001 * (1 + fabs(x));
        f0 = sumFx(N, S, x);
        fp = sumFx(N, S, x + h);
        fm = sumFx(N, S, x - h);
        diff = 2 * h * f0 / (fp - fm);
        x -= diff;
        if (fabs(diff) < toler)
          break;
    }
    return x;
}
```

Implementation for HP-71B

This section presents an implementation of the SopLog function for the HP-71B.

The Listing

```
10 REM SopLog Function
20 DEF FNS(N, S, X)
30 S0 = 1
40 FOR I=2 TO N
50 S0=S0+I^X
60 NEXT I
70 FNS = S - S0
90 END DEF
90 INPUT "ENTER N? ";N
100 INPUT "ENTER S? ";S
```



```
110 R = 1
120 X = 1
130 T = 0.00000001
140 BEEP 500,0.1
150 DISP "X=";X @ WAIT 1
160 H = 0.001*(1 + ABS(X))
170 F0 = FNS(N, S, X)
180 D = H * F0 /(FNS(S, N, X+H) - F0)
190 X = X - D
200 R = R + 1
210 ID ABS(D)>T AND R<1000 THEN 140
220 BEEP 1000,2
230 DISP "SOPLOG=";X
240 END
```

Usage

1. Press the RUN key.
2. The program prompts you with "N? ".
3. Enter the value for N and press the ENDLINE key.
4. The program prompts you with "S? ".
5. Enter the value for S and press the ENDLINE key.
6. The program sounds a short beep and displays intermediate results as "x = #.#######". This step is repeated several times. You need not take any action.
7. When the program is done, it sounds a higher pitch and longer beep, then displays "SOPLOG= #.######".
8. To rerun the program, go to step 1.

If you want to run the program in full or partial quiet mode, you can insert a REM keyword before either or both BEEP commands in lines 140 and/or 210.

Implementation for General Sharp and Casio BASIC Pocket Computers

This section presents an implementation of the SopLog function BASIC that runs on most Sharp, Casio, and TI BASIC pocket computers.

The Listing

```
10 REM SopLog Function
20 INPUT "ENTER N? ";N
30 INPUT "ENTER S? ";S
40 R = 1
50 X = 1
60 T = 0.00000001
70 H = 0.001*(1 + ABS(X))
80 Z = X
```

```
90 GOSUB 1000
100 F = Y
110 Z = X + H
120 GOSUB 1000
130 D = H * F / (Y - F)
140 X = X - D
150 R = R + 1
160 ID ABS(D)>T AND R<1000 THEN 70
170 PRINT "SOPLOG=";X
180 END
1000 REM F(X)=0
1010 Y = 1
1020 FOR I=2 TO N
1030 Y = Y + I^Z
1040 NEXT I
1050 Y = S - Y
1060 RETURN
```

Most of the Sharp, Casio, and TI BASIC pocket computers do not support multiline functions or even single-line function definitions. This missing feature forces me to use GOSUBs, common in all of these machines, to emulate the function that calculate the SopLog summation. In addition, the above listing uses single-letter variable to make sure that the listing works for a maximum number of BASIC pocket computers. A fair number of more *mature* BASIC pocket computers support variable named using either two-letters or a letter and a digit (like with the HP-71B).

The subroutine, staring at line 1000, calculates the function fx(x) for given values of N and S. Its virtual argument is Z (which stores the value of x OR x+h) and returns the result using variable Y. The first call to GOSUB 1000, in line 90, stores the value of X in Z and then stores the result of Y in F. The second call to GOSUB 1000, in line 120, stores the value of X+H in Z and uses the returned value in Y to represent fx(x+h). There is no need to copy the value of Y to another variable after the second GOSUB call.

Usage

1. Press the RUN key.
2. The program prompts you with "N? ".
3. Enter the value for N and press the ENTER key.
4. The program prompts you with "S? ".
5. Enter the value for S and press the ENTER key.
6. When the program is done, displays "SOPLOG= #.######".

7. To rerun the program, go to step 1.

Implementation for the HP-41C Calculator

This section shows you the listing for the HP-41C calculator (and the HP-42S). The program is not long but can require significant time for high values of N. You can use the HP-41CX emulator by Warren Furlow (see www.hp41.org) and then set the Turbo speed to maximum by pressing the TAB key on your PC's keyboard, while the emulator is running, to activate Turbo speed.

The Registers Map

The registers used in the listings are:

Registers	***Contents***
00	S
01	N
02	x
03	toler=1e-8
04	h
05	f0
06	N loop counter
07	Sum of i^x'
08	x' used in subroutine E

Table 6. Registers' map.

The Listing

Here is the listing for the HP-41C implementation.

Step	***Mnemonic***	***Comment***
01	LBL "SOPLOG"	
02	LBL A	
03	"S?"	
04	PROMPT	
05	STO 00	store S
06	"N?"	

Step	Mnemonic	Comment
07	PROMPT	
08	STO 01	store N
09	1	
10	STO 02	x=1
11	1E–08	
12	STO 03	store tolerance
13	LBL 01	start of loop
14	RCL 02	
15	PSE	
16	1	
17	+	
18	0.001	
19	*	
20	STO 04	h
21	RCL 02	
22	XEQ E	calculate f(x)
23	STO 05	store fx(x)
24	RCL 02	
25	RCL 04	
26	+	
27	XEQ E	calculate fx(x+h)
28	RCL 05	
29	-	
30	1/X	
31	RCL 04	
32	*	
33	RCL 05	
34	*	diff=h*fx(x)/(fx(x+h)–fx(x))
35	STO– 02	x = x – diff
36	ABS	
37	RCL 03	

Step	Mnemonic	Comment
38	X<=Y?	still not converged?
39	GTO 01	end of loop
40	BEEP	
41	RCL 02	
42	RTN	
43	LBL E	Calculate f(x)
44	STO 08	store x'
45	0	
46	STO 07	sum=0
47	RCL 01	
48	STO 06	reset loop counter = N
49	LBL 02	start of loop
50	RCL 06	
51	RCL 08	
52	Y^X	calculate i^x
53	STO+ 07	sum=sum+i^x
54	DSE 06	
55	GTO 02	end of loop
56	RCL 00	
57	RCL 07	
58	–	S – sum of i^x
59	RTN	

Table 7. HP-41C listing for SopLog.

Program Usage

1. Go to the program by executing GTO [ALPHA] SOPLOG [ALPHA].
2. Turn on the USER mode by pressing the [USER] button.
3. Press A (the Σ+ key). The program prompts you with “S?”
4. Enter the value of S and press the R/S key.
5. The program prompts you with “N?”
6. Enter the value of N and press the R/S key.
7. The program will pause to display intermediate values for x.

8. When the program converges, it will beep and then display the value for x. The beep allows you to be notified at the end of typically time-consuming calculations—allowing you to perform other tasks while the calculations are in progress.
9. To calculate x for different values of N and/or S, go to step 3.

Implementation for the HP-41CX Calculator

This section presents an enhanced HP-41CX listing that uses the stopwatch of the HP-41CX and some other program enhancements.

The Registers Map

The registers used in the listings are:

Registers	Contents
00	S
01	N
02	x
03	toler=1e-8
04	h
05	f0
06	N loop counter
07	Sum of i^x’
08	x' used in subroutine E

Table 8. Registers’ map.

The Listing

Step	Mnemonic	Comment
01	LBL "SOPLOG"	
02	LBL A	
03	"S?"	
04	PROMPT	
05	STO 00	store S
06	"N?"	

Step	Mnemonic	Comment
07	PROMPT	
08	STO 01	store N
09	CF 22	
10	"X?"	
11	PROMPT	
12	FC? 22	
13	1	
14	STO 02	store x
15	0	
16	SETSW	Reset stopwatch
17	RUNSW	start stopwatch
18	1E-08	
19	STO 03	store toler
20	LBL 01	start of loop
21	TONE 9	
22	"x="	
23	ARCL 02	
24	AVIEW	display intermediate value
25	RCL 02	
26	0.001	
27	*	
28	STO 04	calculate and store h
29	RCL 02	
30	XEQ E	calculate f(x)
31	STO 05	store fx(x)
32	RCL 02	
33	RCL 04	
34	+	
35	XEQ E	calculate f(x+h)
36	RCL 05	
37	-	

Step	Mnemonic	Comment
38	1/X	
39	RCL 04	
40	*	
41	RCL 05	
42	*	diff=h*fx(x)/(fx(x+h)–fx(x))
43	STO- 02	x = x -diff
44	ABS	
45	RCL 03	
46	X<=Y?	still not converged?
47	GTO 01	end of loop
48	STOPSW	stop stopwatch
49	RCLSW	read time elapsed
50	CLD	clear alpha display
51	RCL 02	
52	RTN	
53	LBL E	Calculate f(x)
54	STO 08	store x'
55	0	
56	STO 07	sum=0
57	RCL 01	
58	STO 06	reset loop counter = N
59	LBL 02	start of loop
60	RCL 06	
61	RCL 08	
62	Y^X	calculate i^x
63	STO+ 07	sum=sum+i^x
64	DSE 06	
65	GTO 02	end of loop
66	RCL 00	
67	RCL 07	
68	-	S – sum of i^x

Step	Mnemonic	Comment
69	RTN	

Table 9. HP-41CX listing for SopLog.

I ran the above program on a new Swiss Micros DM41X (a modern day clone of the HP-41CX). To calculate $\text{SopLog}_{1000}(10000)$ it took 14 minutes and 2 seconds. So, you can imagine how long it would take on an original HP-41CX.

You can adapt the HP-41CX listing for the 41C or the 41CV models by eliminating the stopwatch-related commands and keeping the other enhancements.

Program Usage

1. Go to the program by executing GTO [ALPHA] SOPLOG [ALPHA].
2. Turn on the USER mode by pressing the [USER] button.
3. Press A (the Σ+ key). The program prompts you with "S?"
4. Enter the value of S and press the R/S key.
5. The program prompts you with "N?"
6. Enter the value of N and press the R/S key.
7. The program prompts you with "X?".
8. Either key in a value for x and press R/S, or just press R/S to select the default value of 1 for x.
9. The program will sound a short tone and display intermediate values for x as "x= #.#####" in alpha view mode.
10. When the program converges, it will beep and then display the value for x. The beep allows you to be notified at the end of typically time consuming calculations—allowing you to perform other tasks while the calculations are in progress.
11. Press the X<>Y key to view the time the calculations took in hh.mmss format.
12. To calculate x for different values of N and/or S, go to step 3.

Implementation for HP-41C/CV/CX with Advantage Module

This section presents an implementation for the HP-41C/CV/CX models using the Advantage module. This module has the SOLVE subroutine which finds the root of a function. Using SOLVE shortens the program significantly and guarantees accurate results.

The Registers Map

The registers used in the listings are:

Registers	Contents
00	S
01	N
02	x
03	sum
04	loop control in label LBLE

Table 10. Registers' map.

Listing

Step	Mnemonic	Comment
01	LBL "SOPLOG"	
02	LBL A	
03	"S?"	
04	PROMPT	
05	STO 00	
06	"N?"	
07	PROMPT	
08	STO 01	
09	0.05	
10	ENTER	
11	0.95	guesses for root
12	"LBLE"	name of fx(x)=0 in alpha register
13	SOLVE	in Advantage module
14	BEEP	
15	CLD	
16	RTN	
17	LBL "LBLE"	Calculate f(x)
18	TONE 5	
19	STO 02	store x
20	"X="	
21	ARCL 02	

Step	Mnemonic	Comment
22	AVIEW	
23	0	
24	STO 03	sum=0
25	RCL 01	
26	STO 04	reset loop counter = N
27	LBL 01	start of loop
28	RCL 04	
29	RCL 02	
30	Y^X	
31	STO+ 03	sum=sum+i^x
32	DSE 04	
33	GTO 01	end of loop
34	RCL 00	
35	RCL 03	
36	-	
37	RTN	

Table 11. Listing for HP-41CX with Advantage module.

When using the above listing with a HP-41CX you have the option of adding the stop-watch related commands (SETSW, RUNSW, STOPSW, and RCLSW) that appear in the listing before last (in the section titled *Implementation for the HP-41CX Calculator*).

Program Usage

1. Go to the program by executing GTO [ALPHA] SOPLOG [ALPHA].
2. Turn on the USER mode by pressing the [USER] button.
3. Press A (the Σ+ key). The program prompts you with “S?”
4. Enter the value of S and press the R/S key.
5. The program prompts you with “N?”
6. Enter the value of N and press the R/S key.
7. The program will sound a short tone and display intermediate values for x as “x= #.#####” in alpha view mode and plays an audible short tone.
8. When the program converges, it will beep and then display the value for x. The beep allows you to be notified at the end of typically time-consuming

calculations—allowing you to perform other tasks while the calculations are in progress.

9. To calculate x for different values of N and/or S, go to step 3.

Implementation for HP-15C (and HP-34C)

This section presents an implementation for the HP-15C models using the built-in SOLVE function which finds the root of a function. Using SOLVE shortens the program significantly and guarantees accurate results. The program should also work for the HP-34C.

The Registers Map

The registers used in the listings are:

Registers	***Contents***
0	S
1	N
2	x
3	sum
4	loop control in LBL B

Table 12. Registers' map.

Listing

Step	***Key Code***	***Mnemonic***	***Comment***
1	42,21,11	LBL A	
02	44 0	STO 0	
03	34	X<>Y	
04	44 1	STO 01	
05	48	.	
06	0	0	
07	5	5	
08	36	ENTER	
09	48	.	
10	9	9	
11	5	5	guesses for root

Step	Key Code	Mnemonic	Comment
12	42,10,12	SOLVE B	
13	43 32	RTN	
14	42,21,12	LBL B	Calculate f(x)
15	44 2	STO 2	store x
16	42 31	PSE	
17	0	0	
18	44 3	STO 3	sum=0
19	45 1	RCL 1	
20	44 4	STO 4	reset loop counter = N
21	42,21,1	LBL 1	start of loop
22	45 4	RCL 4	
23	45 2	RCL 2	
24	14	Y^X	
25	44,40,3	STO+ 3	sum=sum+i^x
26	42,5,4	DSE 4	
27	22 1	GTO 1	end of loop
28	45 0	RCL 0	
29	45 3	RCL 3	
30	30	-	
31	43 22	RTN	

Table 13. HP-15C Listing.

Program Usage

1. Enter the value of N, press ENTER, and then enter the value for S.
2. Press the A key.
3. The program will display intermediate values for x.
4. When the program converges, it displays the value for x.
5. To calculate x for different values of N and/or S, go to step 1.

The HP-67 Implementation

This section shows you the listing for the HP-67 calculator (and the HP-97). The program is not long but can require significant time for high values of N. You can use the HP-67 and/or HP-97 emulators (available for Windows, IOS, and Android devices) to take advantage of their speed.

The Registers Map

The registers used in the listings are:

Registers	Contents
0	S
1	N
2	x
3	toler=1e-8
4	h
5	f0
6	
7	sum
8	x' used in subroutine E
I	N loop counter

Table 14. Registers' map.

The Listing

Here is the listing for the HP-67 implementation.

Step	Key Code	Mnemonic	Comment
01	31 25 11	LBL A	
02	33 01	STO 1	
03	35 52	X<>Y	
04	33 00	STO 0	
05	01	1	
06	33 02	STO 2	x=1
07	43	EEX	
08	08	8	
09	42	CHS	
10	33 03	STO 3	store toler
11	31 25 01	LBL 1	start of loop
12	34 02	RCL 2	

Step	Key Code	Mnemonic	Comment
13	35 72	PSE	
14	43	EEX	
15	03	3	
16	42	CHS	
17	71	*	
18	33 04	STO 4	h
19	34 02	RCL 2	
20	31 22 15	GSB E	
21	33 05	STO 5	store f0
22	34 02	RCL 2	
23	34 04	RCL 4	
24	61	+	
25	31 22 15	GSB E	
26	34 05	RCL 5	
27	51	-	(minus)
28	35 62	1/X	
29	34 04	RCL 4	
30	71	*	
31	34 05	RCL 5	
32	71	*	
33	33 51 02	STO- 02	x = x -diff
34	35 64	ABS	
35	34 03	RCL 3	
36	32 71	X<=Y?	
37	22 01	GTO 1	end of loop
38	34 02	RCL 2	
39	35 22	RTN	
40	31 25 15	LBL E	Calculate f(x)
41	33 08	STO 08	
42	00	0	
43	33 07	STO 7	sum=0

Step	Key Code	Mnemonic	Comment
44	34 01	RCL 1	
45	35 33	ST I	reset loop counter = N
46	31 25 02	LBL 2	start of loop
47	35 34	RC I	
48	34 08	RCL 8	
49	35 63	Y^X	
50	33 61 07	STO+ 7	sum=sum+i^x
51	31 33	DSZ	
52	22 02	GTO 2	end of loop
53	34 00	RCL 0	
54	34 07	RCL 7	
55	51	-	(minus)
56	35 22	RTN	

Table 15. Listing for HP-67.

The HP-97 version is almost identical, The HP-67 commands ST I, RC I, and DSZ are replaced by the HP-97 commands STO I, RCL I and DSZ I. That's all!

Program Usage

1. Key in the value of S, press ENTER.
2. Key in the value of N.
3. Press A.
4. The program will pause to display intermediate values for x.
5. When the program converges, it will display the value for x.
6. To calculate x for different values of N and/or S, go to step 1.

The HP Prime Implementation

This section presents the HP Prime implementation of the SopLog function:

```
EXPORT SumFx(n, s, x)
BEGIN
  LOCAL sum = 1, i;

  FOR i FROM 2 TO n DO
    sum := sum + i^x;
  END;
  RETURN s - sum;
END;
```

```
EXPORT SopLog(n,s)
BEGIN
  LOCAL x, h, f0, toler, diff;
  toler:=1E-8;
  x := 1;
  diff := 2 * toler;
  WHILE (ABS(diff) >= toler) DO
    h := 0.001 * (1 + ABS(x));
    f0 := SumFx(n, s, x);
    diff := h * f0 /(SumFx(n, s, x+h) - f0);
    x := x - diff;
  END;
  RETURN x;
END;
```

The Detailed Data

This section presents tables for sets of x and S data for a selection of fixed N values (fixed for each set). I have condensed the results in three tables.

For N = 10, 50, 100, and 250

The table for N = 10, 50, 100, and 250 the table of x vs S value is:

	N=10	N=50	N=100	N=250
S	x	x	x	x
100	1.316415	0.227262	0	-0.20687
150	1.525649	0.355612	0.110121	-0.11402
250	1.784519	0.513746	0.245589	9.74E-14
500	2.129188	0.723424	0.424944	0.150678
750	2.327995	0.84402	0.527995	0.237144
1000	2.467998	0.928822	0.600423	0.297879
1100	2.514205	0.956792	0.624305	0.3179
1200	2.556317	0.982274	0.646061	0.336135
1300	2.594997	1.005673	0.666037	0.352877
1400	2.63076	1.027302	0.684499	0.368349
1500	2.664012	1.047408	0.701661	0.38273
1600	2.695083	1.066191	0.717693	0.396162
1700	2.724238	1.083814	0.732732	0.408762
1800	2.7517	1.100409	0.746895	0.420627
1900	2.777653	1.116091	0.760277	0.431837

S	N=10 x	N=50 x	N=100 x	N=250 x
2000	2.802254	1.130954	0.772959	0.44246
2100	2.825636	1.145078	0.785011	0.452554
2200	2.847913	1.158533	0.796492	0.46217
2300	2.869186	1.17138	0.807453	0.47135
2400	2.889539	1.18367	0.817938	0.480131
2500	2.909049	1.19545	0.827988	0.488548
2600	2.927783	1.206761	0.837637	0.496628
2700	2.9458	1.217637	0.846916	0.504397
2800	2.963152	1.228111	0.855851	0.511879
2900	2.979886	1.238211	0.864467	0.519093
3000	2.996046	1.247964	0.872786	0.526059
3100	3.011668	1.257391	0.880827	0.532792
3200	3.026787	1.266515	0.888609	0.539307
3300	3.041434	1.275353	0.896148	0.545619
3400	3.055639	1.283923	0.903458	0.551738
3500	3.069426	1.292241	0.910552	0.557678
3600	3.082819	1.300321	0.917443	0.563447
3700	3.095841	1.308177	0.924143	0.569055
3800	3.108511	1.315819	0.930661	0.574512
3900	3.120847	1.32326	0.937007	0.579824
4000	3.132867	1.33051	0.94319	0.584999
4100	3.144586	1.337578	0.949218	0.590045
4200	3.15602	1.344473	0.955098	0.594967
4300	3.167181	1.351204	0.960838	0.599772
4400	3.178082	1.357777	0.966443	0.604464
4500	3.188735	1.364201	0.971921	0.609049
4600	3.199151	1.370482	0.977277	0.613531
4700	3.20934	1.376625	0.982516	0.617916
4800	3.219312	1.382637	0.987643	0.622207
4900	3.229075	1.388524	0.992662	0.626408
5000	3.238639	1.39429	0.997579	0.630523
5100	3.248012	1.39994	1.002397	0.634555

S	N=10 x	N=50 x	N=100 x	N=250 x
5200	3.2572	1.405479	1.00712	0.638508
5300	3.266211	1.410911	1.011751	0.642384
5400	3.275052	1.41624	1.016295	0.646187
5500	3.283728	1.42147	1.020755	0.649919
5600	3.292246	1.426604	1.025132	0.653583
5700	3.300612	1.431646	1.029432	0.65718
5800	3.30883	1.4366	1.033655	0.660715
5900	3.316907	1.441467	1.037805	0.664188
6000	3.324846	1.446252	1.041884	0.667601
6100	3.332652	1.450956	1.045895	0.670958
6200	3.34033	1.455583	1.04984	0.674259
6300	3.347883	1.460135	1.053721	0.677507
6400	3.355316	1.464614	1.05754	0.680702
6500	3.362633	1.469023	1.061299	0.683848
6600	3.369837	1.473364	1.065	0.686944
6700	3.376931	1.477639	1.068644	0.689994
6800	3.383918	1.481849	1.072234	0.692997
6900	3.390803	1.485997	1.07577	0.695956
7000	3.397587	1.490085	1.079255	0.698872
7100	3.404275	1.494114	1.08269	0.701746
7200	3.410867	1.498086	1.086076	0.70458
7300	3.417368	1.502002	1.089415	0.707373
7400	3.423779	1.505865	1.092708	0.710128
7500	3.430103	1.509675	1.095956	0.712846
7600	3.436342	1.513433	1.09916	0.715527
7700	3.442499	1.517142	1.102322	0.718173
7800	3.448576	1.520803	1.105443	0.720784
7900	3.454574	1.524416	1.108523	0.723361
8000	3.460497	1.527984	1.111564	0.725905
8100	3.466344	1.531506	1.114567	0.728418
8200	3.47212	1.534985	1.117532	0.730899
8300	3.477824	1.538421	1.120462	0.733349

S	N=10 x	N=50 x	N=100 x	N=250 x
8400	3.483459	1.541815	1.123355	0.73577
8500	3.489027	1.545169	1.126214	0.738162
8600	3.49453	1.548483	1.129039	0.740526
8700	3.499967	1.551758	1.131831	0.742861
8800	3.505342	1.554996	1.134591	0.74517
8900	3.510656	1.558196	1.137319	0.747452
9000	3.51591	1.56136	1.140016	0.749709
9100	3.521105	1.564489	1.142683	0.75194
9200	3.526242	1.567583	1.14532	0.754147
9300	3.531324	1.570643	1.147929	0.756329
9400	3.53635	1.57367	1.150509	0.758488
9500	3.541323	1.576665	1.153062	0.760623
9600	3.546244	1.579628	1.155587	0.762736
9700	3.551112	1.58256	1.158087	0.764827
9800	3.55593	1.585461	1.16056	0.766896
9900	3.560699	1.588333	1.163008	0.768943
10000	3.56542	1.591176	1.16543	0.77097

Table 16. First table of data for X, S, and x.

For N = 500, 750, 1000, and 2500

The table for N = 500, 750, 1000, and 250 the table of x vs S values is:

S	N=500 x	N=750 x	N=1000 x	N=2500 x
100	-0.31952	-0.37345	-0.40748	-0.49755
150	-0.23641	-0.29511	-0.3322	-0.43053
250	-0.1345	-0.19913	-0.24001	-0.34858
500	7.45E-15	-0.07254	-0.11848	-0.24072
750	0.077118	2.4E-13	-0.04886	-0.17899
1000	0.131263	0.050922	2.3E-12	-0.13569
1100	0.149107	0.067702	0.0161	-0.12143
1200	0.165359	0.082984	0.030762	-0.10844
1300	0.180279	0.097012	0.044221	-0.09652

S	N=500 x	N=750 x	N=1000 x	N=2500 x
1400	0.194066	0.109975	0.056658	-0.0855
1500	0.206879	0.122023	0.068215	-0.07526
1600	0.218847	0.133275	0.07901	-0.06571
1700	0.230073	0.143829	0.089134	-0.05674
1800	0.240643	0.153766	0.098667	-0.0483
1900	0.25063	0.163154	0.107673	-0.04033
2000	0.260093	0.17205	0.116207	-0.03277
2100	0.269085	0.180503	0.124315	-0.02559
2200	0.277651	0.188555	0.132039	-0.01875
2300	0.285827	0.196241	0.139412	-0.01223
2400	0.293649	0.203594	0.146464	-0.00598
2500	0.301145	0.21064	0.153223	1.3E-12
2600	0.308342	0.217405	0.159711	0.005743
2700	0.315261	0.223909	0.16595	0.011266
2800	0.321925	0.230172	0.171957	0.016583
2900	0.32835	0.236211	0.17775	0.02171
3000	0.334553	0.242041	0.183342	0.02666
3100	0.340549	0.247677	0.188748	0.031444
3200	0.346351	0.25313	0.193978	0.036073
3300	0.351971	0.258413	0.199045	0.040557
3400	0.357421	0.263535	0.203958	0.044905
3500	0.36271	0.268506	0.208725	0.049124
3600	0.367847	0.273334	0.213356	0.053222
3700	0.372841	0.278028	0.217858	0.057206
3800	0.377699	0.282594	0.222237	0.061082
3900	0.38243	0.28704	0.226501	0.064855
4000	0.387038	0.291371	0.230655	0.06853
4100	0.391531	0.295593	0.234705	0.072114
4200	0.395913	0.299712	0.238655	0.075609
4300	0.400191	0.303732	0.242511	0.079021
4400	0.404368	0.307658	0.246276	0.082353
4500	0.408451	0.311495	0.249956	0.085609

S	N=500 x	N=750 x	N=1000 x	N=2500 x
4600	0.412442	0.315245	0.253553	0.088791
4700	0.416346	0.318914	0.257071	0.091905
4800	0.420166	0.322504	0.260515	0.094951
4900	0.423906	0.326019	0.263886	0.097934
5000	0.42757	0.329462	0.267188	0.100855
5100	0.43116	0.332836	0.270423	0.103718
5200	0.434679	0.336143	0.273595	0.106524
5300	0.43813	0.339386	0.276705	0.109276
5400	0.441515	0.342567	0.279756	0.111975
5500	0.444837	0.345689	0.28275	0.114624
5600	0.448099	0.348754	0.285689	0.117225
5700	0.451302	0.351764	0.288576	0.119779
5800	0.454448	0.354721	0.291412	0.122287
5900	0.45754	0.357626	0.294198	0.124752
6000	0.460579	0.360482	0.296937	0.127175
6100	0.463567	0.36329	0.299629	0.129557
6200	0.466505	0.366052	0.302278	0.1319
6300	0.469396	0.368768	0.304883	0.134205
6400	0.472241	0.371441	0.307447	0.136473
6500	0.475041	0.374072	0.30997	0.138705
6600	0.477798	0.376663	0.312454	0.140903
6700	0.480512	0.379214	0.3149	0.143067
6800	0.483186	0.381726	0.31731	0.145198
6900	0.48582	0.384201	0.319683	0.147298
7000	0.488416	0.38664	0.322022	0.149367
7100	0.490974	0.389044	0.324328	0.151407
7200	0.493496	0.391414	0.326601	0.153417
7300	0.495983	0.393751	0.328841	0.155399
7400	0.498435	0.396055	0.331051	0.157354
7500	0.500854	0.398329	0.333231	0.159282
7600	0.503241	0.400571	0.335382	0.161185
7700	0.505596	0.402784	0.337504	0.163062

S	N=500 x	N=750 x	N=1000 x	N=2500 x
7800	0.50792	0.404968	0.339598	0.164914
7900	0.510214	0.407123	0.341665	0.166743
8000	0.512478	0.409251	0.343706	0.168548
8100	0.514715	0.411353	0.345721	0.17033
8200	0.516923	0.413428	0.347711	0.17209
8300	0.519104	0.415477	0.349676	0.173829
8400	0.521259	0.417502	0.351618	0.175546
8500	0.523388	0.419502	0.353536	0.177243
8600	0.525492	0.421479	0.355432	0.17892
8700	0.527571	0.423433	0.357305	0.180577
8800	0.529626	0.425364	0.359157	0.182215
8900	0.531657	0.427272	0.360987	0.183834
9000	0.533665	0.429159	0.362797	0.185434
9100	0.535651	0.431025	0.364586	0.187017
9200	0.537615	0.432871	0.366356	0.188582
9300	0.539558	0.434696	0.368106	0.19013
9400	0.541479	0.436501	0.369837	0.191661
9500	0.54338	0.438287	0.37155	0.193176
9600	0.54526	0.440054	0.373244	0.194675
9700	0.547121	0.441803	0.374921	0.196158
9800	0.548962	0.443533	0.37658	0.197625
9900	0.550785	0.445245	0.378222	0.199078
10000	0.552589	0.44694	0.379848	0.200515

Table 17. Second table of data for X, S, and x.

For N = 5000, 7500, and 10000

The table for N = 5000, 7500, and 10000 the table of x vs S values is:

S	N=5000 x	N=7500 x	N=10000 x
100	-0.55149	-0.57874	-0.59645
150	-0.48956	-0.51942	-0.53885
250	-0.41392	-0.44702	-0.46858

	N=5000	N=7500	N=10000
S	x	x	x
500	-0.31446	-0.35189	-0.37628
750	-0.25759	-0.29751	-0.32354
1000	-0.21771	-0.25938	-0.28657
1100	-0.20458	-0.24683	-0.2744
1200	-0.19262	-0.2354	-0.26331
1300	-0.18164	-0.22491	-0.25314
1400	-0.1715	-0.21521	-0.24374
1500	-0.16208	-0.20621	-0.23501
1600	-0.15328	-0.1978	-0.22686
1700	-0.14502	-0.18991	-0.21921
1800	-0.13725	-0.18249	-0.21201
1900	-0.12991	-0.17547	-0.20521
2000	-0.12296	-0.16883	-0.19877
2100	-0.11635	-0.16251	-0.19265
2200	-0.11006	-0.1565	-0.18682
2300	-0.10405	-0.15076	-0.18125
2400	-0.09831	-0.14527	-0.17593
2500	-0.0928	-0.14001	-0.17083
2600	-0.08751	-0.13496	-0.16593
2700	-0.08243	-0.1301	-0.16122
2800	-0.07754	-0.12543	-0.15669
2900	-0.07282	-0.12092	-0.15232
3000	-0.06827	-0.11657	-0.1481
3100	-0.06386	-0.11236	-0.14403
3200	-0.0596	-0.10829	-0.14008
3300	-0.05548	-0.10435	-0.13626
3400	-0.05148	-0.10053	-0.13255
3500	-0.0476	-0.09682	-0.12896
3600	-0.04382	-0.09321	-0.12547
3700	-0.04016	-0.08971	-0.12207
3800	-0.03659	-0.0863	-0.11877
3900	-0.03312	-0.08299	-0.11555

	N=5000	N=7500	N=10000
S	x	x	x
4000	-0.02974	-0.07976	-0.11242
4100	-0.02644	-0.07661	-0.10937
4200	-0.02323	-0.07353	-0.10639
4300	-0.02009	-0.07054	-0.10348
4400	-0.01702	-0.06761	-0.10064
4500	-0.01403	-0.06475	-0.09787
4600	-0.0111	-0.06195	-0.09516
4700	-0.00823	-0.05921	-0.09251
4800	-0.00543	-0.05653	-0.08991
4900	-0.00269	-0.05391	-0.08737
5000	1.28E-13	-0.05134	-0.08488
5100	0.002633	-0.04883	-0.08244
5200	0.005215	-0.04636	-0.08005
5300	0.007746	-0.04394	-0.07771
5400	0.01023	-0.04157	-0.07541
5500	0.012667	-0.03924	-0.07315
5600	0.015059	-0.03696	-0.07094
5700	0.017408	-0.03471	-0.06876
5800	0.019716	-0.03251	-0.06662
5900	0.021984	-0.03034	-0.06452
6000	0.024213	-0.02821	-0.06246
6100	0.026404	-0.02612	-0.06043
6200	0.028559	-0.02406	-0.05843
6300	0.030679	-0.02204	-0.05647
6400	0.032766	-0.02004	-0.05454
6500	0.034819	-0.01808	-0.05264
6600	0.03684	-0.01615	-0.05077
6700	0.038831	-0.01425	-0.04892
6800	0.040792	-0.01238	-0.04711
6900	0.042723	-0.01053	-0.04532
7000	0.044627	-0.00871	-0.04356
7100	0.046502	-0.00692	-0.04182

	N=5000	N=7500	N=10000
S	x	x	x
7200	0.048352	-0.00515	-0.04011
7300	0.050175	-0.00341	-0.03842
7400	0.051973	-0.00169	-0.03676
7500	0.053747	2.98E-12	-0.03511
7600	0.055497	0.001671	-0.03349
7700	0.057223	0.003321	-0.03189
7800	0.058927	0.004948	-0.03032
7900	0.060609	0.006555	-0.02876
8000	0.062269	0.008141	-0.02722
8100	0.063909	0.009707	-0.0257
8200	0.065528	0.011254	-0.02421
8300	0.067127	0.012781	-0.02272
8400	0.068707	0.01429	-0.02126
8500	0.070267	0.015781	-0.01982
8600	0.07181	0.017254	-0.01839
8700	0.073334	0.01871	-0.01698
8800	0.07484	0.020149	-0.01558
8900	0.076329	0.021572	-0.0142
9000	0.077802	0.022978	-0.01284
9100	0.079257	0.024369	-0.01149
9200	0.080697	0.025744	-0.01016
9300	0.082121	0.027104	-0.00884
9400	0.083529	0.028449	-0.00754
9500	0.084922	0.02978	-0.00625
9600	0.086301	0.031096	-0.00497
9700	0.087665	0.032399	-0.00371
9800	0.089014	0.033689	-0.00246
9900	0.09035	0.034965	-0.00122
10000	0.091672	0.036228	1.17E-14

Table 18. Third table of data for X, S, and x.

Applications for the SopLog Function

The only current application I can think of for the SopLog function is benchmarking calculators and computers (various OS and programming languages). Vintage programmable calculators will take quite the time to calculate the SopLog iteratively for N values like 50, 100 and higher values.

Epilog: Calculating Sums of Integers Raised to Integer Powers

This section goes back to the Gauss sum of integers and similar sums of squares and cubes. The following equation shows the sums of integers for powers 1, 2, and 3:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \quad (8)$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \quad (9)$$

$$\sum_{k=1}^{n} k^3 = \frac{[n(n+1)]^2}{4} \quad (10)$$

Summations to higher power have more complex equations that involve the Bernoulli numbers using the Faulhaber's formula.

This section enquires about a logarithmic relation between the sum S and the power x:

$$\log_{10}(S) = a + b\ x \quad (11)$$

The next table shows that equation (11) holds well for a fixed value of N, the number of terms. The table has sets of sums for N = 100, 250, 500, 750, and 1000. The powers are in the range of 1 to 10.

N	x	S	ln(S)			
100	1	5050	8.527144			
100	2	338350	12.73184			
100	3	25502500	17.05429			
100	4	2050333330	21.44127			
100	5	1.71708E+11	25.86906			
100	6	1.47907E+13	30.32502			
100	7	1.30058E+15	34.80159			

N	x	S	ln(S)			
100	8	1.16178E+17	39.2939			
100	9	1.05075E+19	43.79862	Slope	Intercept	Rsqr
100	10	9.59924E+20	48.31339	4.431992	3.839656	0.999908
250	1	31375	10.35377			
250	2	5239625	15.47176			
250	3	984390625	20.70753			
250	4	1.97271E+11	26.00784			
250	5	4.118E+13	31.34897			
250	6	8.84187E+15	36.71827			
250	7	1.93801E+18	42.10819			
250	8	4.31525E+20	47.51386			
250	9	9.72862E+22	52.93194	Slope	Intercept	Rsqr
250	10	2.21544E+25	58.36008	5.345334	4.752887	0.999937
500	1	125250	11.73807			
500	2	41791750	17.54821			
500	3	15687562500	23.47613			
500	4	6.28129E+12	29.4686			
500	5	2.61982E+15	35.50188			
500	6	1.1239E+18	41.56334			
500	7	4.92197E+20	47.64541			
500	8	2.18972E+23	53.74323			
500	9	9.86357E+25	59.85348	Slope	Intercept	Rsqr
500	10	4.48791E+28	65.97377	6.037488	5.445025	0.99995
750	1	281625	12.54833			
750	2	140906375	18.76361			
750	3	79312640625	25.09666			
750	4	4.76193E+13	31.49426			
750	5	2.97819E+16	37.93268			
750	6	1.91582E+19	44.39926			
750	7	1.2581E+22	50.88647			
750	8	8.39289E+24	57.38943			

N	x	S	ln(S)			
750	9	5.66897E+27	63.9048	Slope	Intercept	Rsqr
750	10	3.86778E+30	70.43023	6.442621	5.850156	0.999956
1000	1	500500	13.12336			
1000	2	333833500	19.62615			
1000	3	2.505E+11	26.24673			
1000	4	2.005E+14	32.93184			
1000	5	1.67167E+17	39.65777			
1000	6	1.43358E+20	46.41187			
1000	7	1.25501E+23	53.1866			
1000	8	1.11612E+26	59.97707			
1000	9	1.00501E+29	66.77996	Slope	Intercept	Rsqr
1000	10	9.14099E+31	73.59291	6.730137	6.13767	0.99996

Table 19. Table for S, N, x, and ln(S).

The above table shows the calculated slope, intercept, and coefficient of determination for each set of fixed N values and fitting equation (11). The next table shows the values of N, the slope in equation (11), and log10(S).

N	Slope eqn 11	Log10(N)
100	4.431992	2
250	5.345334	2.39794
500	6.037488	2.69897
750	6.442621	2.875061
1000	6.730137	3

Table 20. Table for N, slope, and $log_{10}(N)$.

The following ANOVA table shows the results of the regression model:

Slope eqn 11 = a + b log10(N) (12)

SUMMARY OUTPUT					
Regression Statistics					
Multiple R	0.999999840				

R Square	0.999999681				
Adjusted R Square	0.999999574				
Standard Error	0.000602				
Observations	5				
ANOVA					
	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	3.412654	3.412654	9405644	7.65E-11
Residual	3	1.09E-06	3.63E-07		
Total	4	3.412655			
	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*
Intercept	-0.16492	0.001963	-84.028	3.72E-06	-0.17117
Log10(N)	2.298201	0.000749	3066.862	7.65E-11	2.295816

The next table shows the values of N, the intercept in equation (11) and log10(S).

N	Intercept eqn 11	Log10(N)
100	3.839656	2
250	4.752887	2.39794
500	5.445025	2.69897
750	5.850156	2.875061
1000	6.13767	3

Table 21. Table for N, intercept, and $log_{10}(N)$.

The following ANOVA table shows the results of the regression model:

Intercept eqn 11 = a + b log10(N) (13)

SUMMARY OUTPUT						
Regression Statistics						
Multiple R	0.999999824					
R Square	0.999999649					
Adjusted R Square	0.999999532					

Standard Error	0.000631					
Observations	5					
ANOVA						
	df	*SS*	*MS*	*F*	*Significance F*	
Regression	1	3.41228	3.41228	8559959	8.81E-11	
Residual	3	1.2E-06	3.99E-07			
Total	4	3.412282				
	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*
Intercept	-0.75703	0.002057	-367.977	4.43E-08	-0.76358	-0.75049
Log10(N)	2.298075	0.000785	2925.741	8.81E-11	2.295575	2.300574

Combining and simplifying equations (11), (12), and (13) we get:

$$\log_{10}(S) = a_0 + a_1 x + a_2 \log_{10}(N) + a_3 x*\log_{10}(N) \quad (14)$$

A multiple regression using equation (14) yields the following results:

SUMMARY OUTPUT				
Regression Statistics				
Multiple R	0.999975979			
R Square	0.999951958			
Adjusted R Square	0.999948825			
Standard Error	0.055414311			
Observations	50			
ANOVA				
	df	*SS*	*MS*	*F*
Regression	3	2940.074577	980.024859	319148.7999
Residual	46	0.14125431	0.003070746	
Total	49	2940.215831		

	Coefficients	*Standard Error*	*t Stat*	*P-value*
Intercept	-0.328775144	0.123348118	-2.665424894	0.010571473
x	-0.071625445	0.019879349	-3.603007549	0.000769298
log10(N)	0.998041133	0.047094165	21.19245861	2.25702E-25
x*log10(N)	0.998095806	0.007589912	131.502951	6.52553E-61

The above ANOVA table reveals the values in the symbolic constants of equation (14):

$$\log_{10}(S) = -0.328775144 - 0.071625445 \cdot x + 0.998041133 \cdot \log_{10}(N) + 0.998095806 \cdot x \cdot \log_{10}(N) \quad (15)$$

Equation (15) can be roughly approximated (for esthetics' purpose) as:

$$\log_{10}(S) = -1/3 - 1/14 \cdot x + \log_{10}(N) + x \cdot \log_{10}(N) \quad (16)$$

Or,

$$\log_{10}(S) = (1 + x) \cdot \log_{10}(N) - x/14 - 1/3 \quad (16b)$$

The regression results in equation (15) are based on the data in the following table:

log10(S)	x	log10(N)	x*log10(N)
3.703291	1	2	2
5.529366	2	2	4
7.406583	3	2	6
9.311824	4	2	8
11.23479	5	2	10
13.16999	6	2	12
15.11414	7	2	14
17.06512	8	2	16
19.0215	9	2	18
20.98224	10	2	20
4.496584	1	2.39794	2.39794
6.7193	2	2.39794	4.79588
8.993167	3	2.39794	7.19382

log10(S)	x	log10(N)	x*log10(N)
11.29506	4	2.39794	9.59176
13.61469	5	2.39794	11.9897
15.94654	6	2.39794	14.38764
18.28736	7	2.39794	16.78558
20.63501	8	2.39794	19.18352
22.98805	9	2.39794	21.58146
25.34546	10	2.39794	23.9794
5.097778	1	2.69897	2.69897
7.621091	2	2.69897	5.39794
10.19556	3	2.69897	8.09691
12.79805	4	2.69897	10.79588
15.41827	5	2.69897	13.49485
18.05073	6	2.69897	16.19382
20.69214	7	2.69897	18.89279
23.34039	8	2.69897	21.59176
25.99403	9	2.69897	24.29073
28.65204	10	2.69897	26.9897
5.449671	1	2.875061	2.875061
8.148931	2	2.875061	5.750123
10.89934	3	2.875061	8.625184
13.67778	4	2.875061	11.50025
16.47395	5	2.875061	14.37531
19.28236	6	2.875061	17.25037
22.09971	7	2.875061	20.12543
24.92391	8	2.875061	23.00049
27.7535	9	2.875061	25.87555
30.58746	10	2.875061	28.75061
5.699404	1	3	3
8.52353	2	3	6
11.39881	3	3	9
14.30212	4	3	12

log10(S)	x	log10(N)	x*log10(N)
17.22315	5	3	15
20.15642	6	3	18
23.09865	7	3	21
26.04771	8	3	24
29.00217	9	3	27
31.96099	10	3	30

Table 22. Table for $log_{10}(S)$, x, $log_{10}(N)$, and $x*log_{10}(N)$.

Now we compare the calculated S vs the estimated ones using equation (14):

N	x	S	S estimates	%Error
100	1	5050	3907.269	22.62834
100	2	338350	328427.5	2.932627
100	3	25502500	27606136	-8.24874
100	4	2050333330	2.32E+09	-13.1742
100	5	1.71708E+11	1.95E+11	-13.5917
100	6	1.47907E+13	1.64E+13	-10.8447
100	7	1.30058E+15	1.38E+15	-5.9576
100	8	1.16178E+17	1.16E+17	0.295809
100	9	1.05075E+19	9.74E+18	7.337661
100	10	9.59924E+20	8.18E+20	14.74264
250	1	31375	24334.14	22.44098
250	2	5239625	5104632	2.576381
250	3	984390625	1.07E+09	-8.77909
250	4	1.97271E+11	2.25E+11	-13.8671
250	5	4.118E+13	4.71E+13	-14.4256
250	6	8.84187E+15	9.88E+15	-11.7929
250	7	1.93801E+18	2.07E+18	-6.99189
250	8	4.31525E+20	4.35E+20	-0.79725
250	9	9.72862E+22	9.12E+22	6.210966
250	10	2.21544E+25	1.91E+25	13.60448
500	1	125250	97076.28	22.49399

N	x	S	S estimates	%Error
500	2	41791750	40674136	2.674246
500	3	15687562500	1.7E+10	-8.63457
500	4	6.28129E+12	7.14E+12	-13.6789
500	5	2.61982E+15	2.99E+15	-14.1991
500	6	1.1239E+18	1.25E+18	-11.5351
500	7	4.92197E+20	5.25E+20	-6.71014
500	8	2.18972E+23	2.2E+23	-0.49871
500	9	9.86357E+25	9.22E+25	6.519636
500	10	4.48791E+28	3.86E+28	13.91735
750	1	281625	218079.8	22.56377
750	2	140906375	1.37E+08	2.804557
750	3	79312640625	8.6E+10	-8.44144
750	4	4.76193E+13	5.4E+13	-13.4269
750	5	2.97819E+16	3.39E+16	-13.8959
750	6	1.91582E+19	2.13E+19	-11.19
750	7	1.2581E+22	1.34E+22	-6.3331
750	8	8.39289E+24	8.4E+24	-0.09951
750	9	5.66897E+27	5.28E+27	6.931978
750	10	3.86778E+30	3.31E+30	14.33483
1000	1	500500	387266.8	22.62402
1000	2	333833500	3.24E+08	2.917201
1000	3	2.505E+11	2.71E+11	-8.27446
1000	4	2.005E+14	2.27E+14	-13.209
1000	5	1.67167E+17	1.9E+17	-13.6338
1000	6	1.43358E+20	1.59E+20	-10.8918
1000	7	1.25501E+23	1.33E+23	-6.00743
1000	8	1.11612E+26	1.11E+26	0.24515
1000	9	1.00501E+29	9.32E+28	7.287837
1000	10	9.14099E+31	7.8E+31	14.69497

Table 23. Results comparing S and the estimates of S for various N and x values.

The above results show that the estimated summations do deviate *significantly* from the exact calculated ones. The reason is that the model in equation (14) was meant for estimating the values of $\log_{10}(S)$ and not S. This comment is verified by the following table:

N	x	S	log10(S)	log10(S) estm	%Error
100	1	5050	3.703291	3.59187329	3.008623
100	2	338350	5.529366	5.516439457	0.233783
100	3	25502500	7.406583	7.441005624	-0.46476
100	4	2050333330	9.311824	9.365571792	-0.57719
100	5	1.71708E+11	11.23479	11.29013796	-0.49264
100	6	1.47907E+13	13.16999	13.21470413	-0.33952
100	7	1.30058E+15	15.11414	15.13927029	-0.16628
100	8	1.16178E+17	17.06512	17.06383646	0.007539
100	9	1.05075E+19	19.0215	18.98840263	0.173996
100	10	9.59924E+20	20.98224	20.9129688	0.330127
250	1	31375	4.496584	4.386216041	2.454479
250	2	5239625	6.7193	6.707964462	0.168704
250	3	984390625	8.993167	9.029712883	-0.40637
250	4	1.97271E+11	11.29506	11.3514613	-0.49932
250	5	4.118E+13	13.61469	13.67320973	-0.42985
250	6	8.84187E+15	15.94654	15.99495815	-0.3036
250	7	1.93801E+18	18.28736	18.31670657	-0.1605
250	8	4.31525E+20	20.63501	20.63845499	-0.01671
250	9	9.72862E+22	22.98805	22.96020341	0.121141
250	10	2.21544E+25	25.34546	25.28195183	0.250573
500	1	125250	5.097778	4.987113135	2.17084
500	2	41791750	7.621091	7.609318332	0.154469
500	3	15687562500	10.19556	10.23152353	-0.35278
500	4	6.28129E+12	12.79805	12.85372873	-0.43506
500	5	2.61982E+15	15.41827	15.47593392	-0.37399
500	6	1.1239E+18	18.05073	18.09813912	-0.26266
500	7	4.92197E+20	20.69214	20.72034432	-0.13631

N	x	S	log10(S)	log10(S) estm	%Error
500	8	2.18972E+23	23.34039	23.34254952	-0.00926
500	9	9.86357E+25	25.99403	25.96475471	0.11264
500	10	4.48791E+28	28.65204	28.58695991	0.227154
750	1	281625	5.449671	5.338615402	2.037844
750	2	140906375	8.148931	8.136576546	0.151604
750	3	79312640625	10.89934	10.93453769	-0.32291
750	4	4.76193E+13	13.67778	13.73249884	-0.40004
750	5	2.97819E+16	16.47395	16.53045998	-0.34301
750	6	1.91582E+19	19.28236	19.32842112	-0.2389
750	7	1.2581E+22	22.09971	22.12638227	-0.12067
750	8	8.39289E+24	24.92391	24.92434341	-0.00173
750	9	5.66897E+27	27.7535	27.72230456	0.112416
750	10	3.86778E+30	30.58746	30.5202657	0.219684
1000	1	500500	5.699404	5.588010229	1.954482
1000	2	333833500	8.52353	8.510672203	0.15085
1000	3	2.505E+11	11.39881	11.43333418	-0.30289
1000	4	2.005E+14	14.30212	14.35599615	-0.37673
1000	5	1.67167E+17	17.22315	17.27865812	-0.32228
1000	6	1.43358E+20	20.15642	20.2013201	-0.22275
1000	7	1.25501E+23	23.09865	23.12398207	-0.10969
1000	8	1.11612E+26	26.04771	26.04664404	0.004092
1000	9	1.00501E+29	29.00217	28.96930602	0.113313
1000	10	9.14099E+31	31.96099	31.89196799	0.215967

Table 24. Results comparing $log_{10}(S)$ and the estimates of $log_{10}(S)$ for various N and x values.

Indeed, the percent errors in the common log values is much smaller. Raising these values of S and estimated S to the power 10 amplifies the errors. Thus equation (15) is good for estimating the *common log of the summations* given N and x.

SopLog's Kissing Cousins

The last part of this study is a heads-up look at SopLog variants. These variants may open a pandora's box, but I will keep it simple. In calculating SolpLog values, the power x remains constant for all integers. The idea behind the variants of SopLog is to alter the power of these integers in a scaled way. Here we have two main options on how we sequence the integers—from 1 to N, or from N down to 1. The powers to which these integers are raised, are scaled downward or upward by factors slightly above or below 1.

The Scaled Increasing SopLog Function siLog

The scaled increasing variant of the SopLog function, dubbed siLog is defined as:

$$\text{siLog}_N(S, sf) = x \qquad (17)$$

Where the integer N is the base of the function, S is a summation, and sf is a scaling factor defined as:

$$S = 1 + \sum_{i=2}^{N} i^{x*df\wedge(i-2)} \qquad (18)$$

The scaling factor is usually close to one with values below one (causing damping powers) or above one (causing exploding power).

Here is the implementation of function siLog (and its companion sumFx) in Excel VBA:

```
Function SumFx(ByVal N As Integer, ByVal S As Double, ByVal Pwr
As Double, _
               ByVal ScalingFactor As Double) As Double
  Dim I As Integer
  Dim Sum As Double, Factor As Double

  Sum = 1
  Factor = 1
  For I = 2 To N
    Sum = Sum + I ^ (Pwr * Factor)
    Factor = Factor * ScalingFactor
  Next I
  SumFx = Sum - S

End Function

Function siLog(ByVal N As Integer, ByVal S As Double, _
    ByVal ScalingFactor As Double, Optional ByVal Toler =
0.00000001) As Double
  Dim X As Double, r As Integer
  Dim h As Double, Diff As Double, F0 As Double

  r = 1
```

```
  X = 1
  Do
    h = 0.0001 * (1 + Abs(X))
    F0 = SumFx(N, S, X, ScalingFactor)
    Diff = h * F0 / (SumFx(N, S, X + h, ScalingFactor) - F0)
    X = X - Diff
    r = r + 1
  Loop Until Abs(Diff) < Toler Or r > 1000
  siLog = X
End Function
```

The Scaled Decreasing SopLog Function sdLog

The scaled decreasing variant of the SopLog function, dubbed sdLog is defined as:

$$\text{sdLog}_N(S, sf) = x \quad (19)$$

Where the integer N is the base of the function, S is a summation, and sf is a scaling factor defined as:

$$S = 1 + \sum_{i=N}^{2} i^{x*df^{\wedge}(N-i)} \quad (20)$$

The summation in equation (20) is of decreasing orders, handling integers from N down to 2. The scaling factor has values slightly below 1 (causing damped powers) or slightly above 1 (causing exploding power).
Here is the implementation of function sdLog (and its companion sumFx) in Excel VBA:

```
Function SumFx(ByVal N As Integer, ByVal S As Double, ByVal Pwr
As Double, ByVal ScalingFactor As Double) As Double
  Dim I As Integer
  Dim Sum As Double, Factor As Double

  Sum = 1
  Factor = 1
  For I = N To 2 Step -1
    Sum = Sum + I ^ (Pwr * Factor)
    Factor = Factor * ScalingFactor
  Next I
  SumFx = Sum - S

End Function

Function sdLog(ByVal N As Integer, ByVal S As Double, _
    ByVal ScalingFactor As Double, _
    Optional ByVal Toler = 0.00000001) As Double
  Dim X As Double, r As Integer
```

```
  Dim h As Double, Diff As Double, F0 As Double

  r = 1
  X = 1
  Do
    h = 0.0001 * (1 + Abs(X))
    F0 = SumFx(N, S, X, ScalingFactor)
    Diff = h * F0 / (SumFx(N, S, X + h, ScalingFactor) - F0)
    X = X - Diff
    r = r + 1
  Loop Until Abs(Diff) < Toler Or r > 1000
  sdLog = X
End Function
```

Both functions siLog and sdLog exhibit a logarithmic relationship between x and S for fixed N and scaling factor values (slightly above and below 1).

The scheme, that I mentioned in this section, for manipulating the powers of the summed integers is but one version. One can imagine other versions for changing the scaling factor such as a linear variation, quadratic variation, exponential decay, exponential rise, and so on. Some of these schemes may need to specify the limit of the initial and final scaling factors to take effect across N–1 integers (since we exclude the term with the integer 1 from being raised to a power).

Document History

Version	Date	Comments
1.0.0	March 29, 2021	Initial release.

