Issue 30, January 2013

HP Solve

Calculating solutions powered by HP

Taming computer chaos in the classroom with HP Classroom Manager

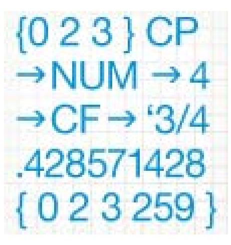
Every teacher dreams of better classroom control, so HP applied its calculator expertise to a program that encourages better student attention and performance.

Learn more

Continued fractions: A step-by-step tutorial

Joseph K. Horn

A very clear and easily understandable process for converting decimal and simple fractions into continued fractions.



Read more

Quiz - How well do you know RPN?

Richard J. Nelson

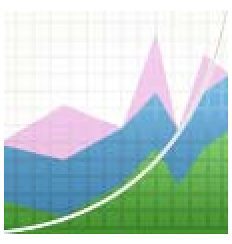
A fun way to remember all of those RPN tips that have appeared in every *HP Solve* issue up to Issue 20.

Take the quiz

HP 39gII Regression: Part I

Namir Shammas

As the power of calculators increases, so do the advanced math problems they conveniently solve. This first part explores the statistical regression with the HP 39gII.

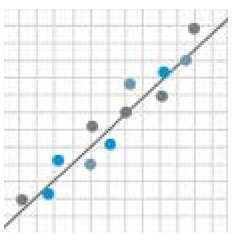


Learn more

HP 39gII Regression: Part II

Namir Shammas

This second part shows you how to perform linearized regression between two and three variables.



Learn more

HP 39gII Regression: Part III

Namir Shammas

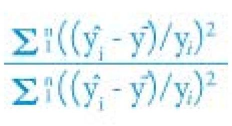
Part three looks at HP 39gII functions that systematically try a large number of regression models to see which ones are the best.

Learn more

HP 39gII Regression: Part IV

Namir Shammas

This final part looks at least-squares relative error (LSRE) regression using the HP 39gII calculator.



Learn more

Welcome to the thirtieth edition of the HP Solve newsletter. Learn more about current articles and feedback from the latest HP Solve newsletter, including a new One Minute Marvels and HP user community news.

Read more

Download the PDF version of articles

Join our Facebook Fan Page

Contact the editor

Update profile Change email HP home Support

HP respects your privacy. If you'd like to discontinue receiving e-mails from HP regarding special offers and information, please click here.

For more information regarding HP's privacy policy or to obtain contact information please visit our privacy statement or write to us at: HP Privacy Mailbox, 11445 Compaq Center Drive W., Mailstop 040307, Houston, TX 77070, ATTN: HP Privacy Mailbox.

© 2013 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice.

HP Classroom Manager

HP Solve #30 page 2

Article Next →

Return to top

Taming Computer Chaos in the Classroom with HP Classroom Manager

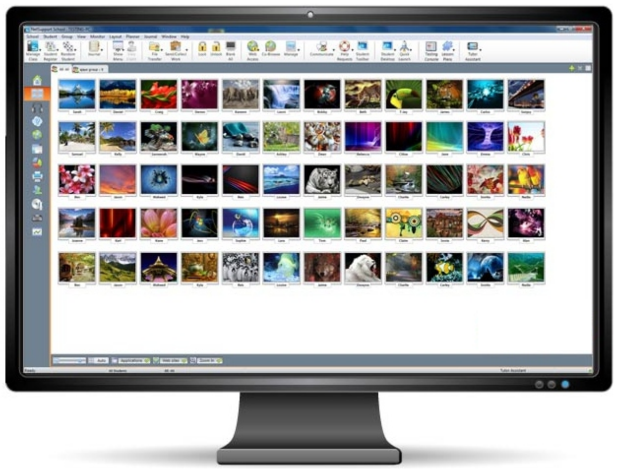
A new school day starts, Ms. Andrews welcomes her students as they rush into the classroom and scurry to turn on their computers. She begins her lesson in geometric proofs to the class, but suddenly 3 students start giggling in the back row, looking at someone's computer screen. She asks them to stop but then sees a student near the front row laughing at a YouTube video… its classroom chaos! What could others be up to… checking out Facebook photos, updating their Twitter, chatting with their friend in the front row? What can Ms. Anderson do to stop them? She can't disrupt the whole class to check her students' screens, and has no way to ensure they are viewing the correct website she wrote on the whiteboard. So she goes about her lesson, wishing she could do more to monitor their activity, secretly hoping the pop quiz later that evening will give her some answers…

As our world becomes more and more dependent on technology, so do our classrooms. With desktops, laptops, and tablets rapidly becoming the norm for students, educators have infinite possibilities to make teaching truly engaging but are also having to address the issues technology presents in the classroom. The scenario above is all too common in many schools today, but Classroom Management software now provides the best way to enhance the benefits of technology for students and educators. With HP's new software, *HP Classroom Manager*, teachers can take control of the classroom, manage activity in all class PCs, and develop unique ways to communicate with students all while making IT management cost effective and easy for school administrators.

Through PC management and an arsenal of interactive teaching tools, HP Classroom Manager redefines the digital classroom, preventing unwanted distractions and enhancing learning to elevate student achievement levels:

View and manage multiple student PCs at one time

- Easily view an entire classroom of PCs at one time
- Watch, share, or control the screen of any student PC
- Add names to student screens and arrange them to appear as they do in class
- Send and receive instant messages and alerts
- Initiate group or 1:1 chats



Control PCs in the classroom

• Power on or off all classroom computers from the teacher PC.

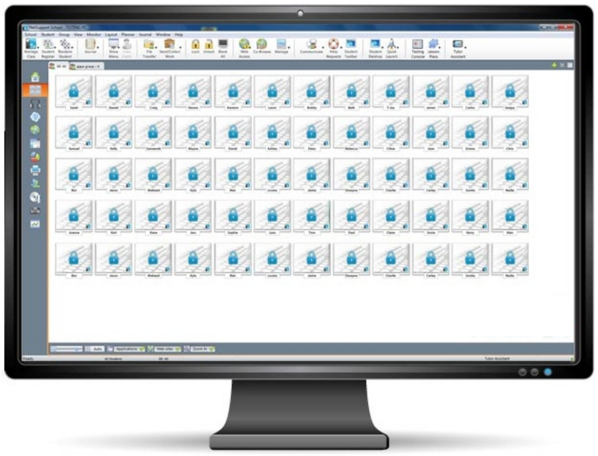
• Lock or black out student PCs with a single click.

• Specify allowed or restricted websites to limit internet activity.

• Open or close specified programs and application on one or multiple computers.

• Nominate a student to be assigned teacher rights and act as a Group Leader whenever desired

• Remotely launch applications, specific documents or websites instantly with the Quick Launch feature



Enrich learning

• Easily share digital content including pictures, documents, and videos

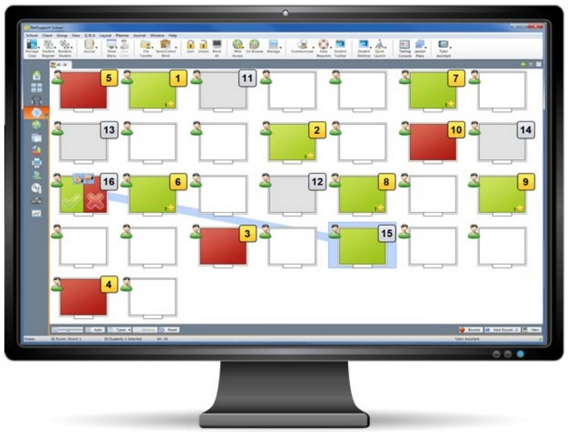
• Administer quizzes and surveys in real-time with instant reports and scoring

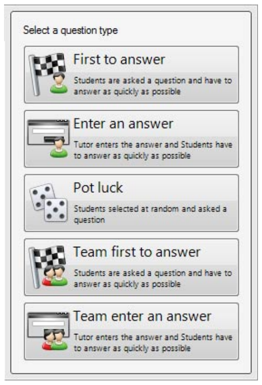
• Enable creative learning in "game show" style with the Question and Answer feature

• Create digital journals of a day's lessons for student revision and lesson plan management

• Enrich language lab learning with audio monitoring and recording tools

• Design tests and exams with minimum of effort, including text, picture, audio and video questions.





Select a question type

First to answer
Students are asked a question and have to answer as quickly as possible

Enter an answer
Tutor enters the answer and Students have to answer as quickly as possible

Pot luck
Students selected at random and asked a question

Team first to answer
Students are asked a question and have to answer as quickly as possible

Team enter an answer
Tutor enters the answer and Students have to answer as quickly as possible

Make IT management easy and cost efficient

• Monitor all computers across the school network in a single view.

• Control printer and connected devices such as keyboards and USB flash drives.

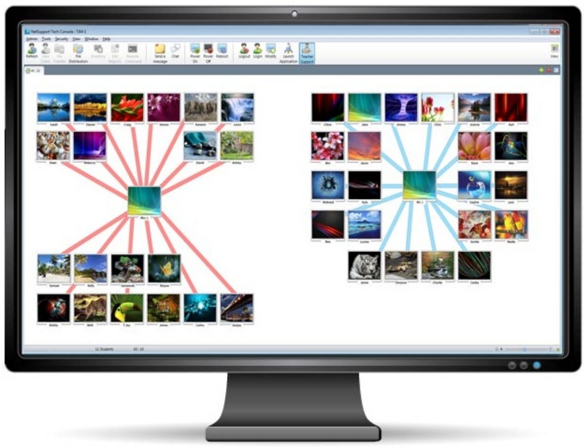
• Power on, Power off, Reboot and Login to classroom computers remotely.

• Monitor all computers across the school network in a single view.

• Set Security Policies to identify computers without anti-virus, Windows updates or Internet protection.

• Secure teacher profiles each allowing customized levels of functionality as required.

• Use the Policy Management tool to apply restrictions permanently across the school.



HP Classroom Manager can be deployed and used across multiple Windows-based platforms, including desktops, laptops and workstations. Its features lower electricity costs, reduced printing accidents and helps ensure PC security, immediately cutting costs to schools.

And with HP Classroom Manager's easily scalable low cost per seat, it's a win-win for students, teachers and school administrations.

For more information on HP Classroom Manager and to download a free 30 day trail, visit www.hp.com/go/hpclassroommanager.

Continued Fractions: A Step-by-Step Tutorial

HP Solve #30 page 6

Continued Fractions:
A Step-by-Step Tutorial, with User RPL Implementations

Joseph K. Horn

Table of contents

Chapter I
How to convert decimal fractions & regular fractions into continued fractions.

Part A: Step-by-step example.

Example: Convert the terminating decimal 3.1416 into its exactly equivalent continued fraction.

Step 1: Rewrite 3.1416 as an equivalent simple fraction: $\frac{31416}{10000}$. Reduce it: $\frac{3927}{1250}$.

Step 2: Rewrite that as a mixed number: $3+\frac{177}{1250}$.

Step 3: Remove the integer part: $\frac{177}{1250}$.

Step 4: Reciprocate: $\frac{1250}{777}$.

Repeat steps 2 through 4 until no fraction part remains:

$\rightarrow \frac{1250}{777}=7+\frac{11}{177}$.

$\rightarrow \frac{177}{11} = 16 + \frac{1}{11}$.

$\rightarrow$ 11, stop.

Step 5: Write all the integer parts (in red above) in a list: {3, 7, 16, 11}. Done.

Answer: 3.1416 is exactly equivalent to the continued fraction {3, 7, 16, 11}.

Note: The notation "{3, 7, 16, 11}" is mathematical shorthand for $3+\cfrac{1}{7+\cfrac{1}{16+\cfrac{1}{11}}}$.

Part B: Generalized method.

To convert a decimal fraction or a simple fraction into a continued fraction:

Step 1: If you have a decimal fraction, rewrite it as a simple fraction. Reduce the simple fraction.
Step 2: Rewrite it as a mixed number.
Step 3: Remove the integer part.
Step 4: Reciprocate.
 Repeat steps 2-4 until no fraction part remains.
Step 5: Write all the integer parts in a list. Done.

Part C: RPL implementation.

Step 1: Convert a decimal fraction into an equivalent reduced Simple Fraction:

```
« IF DUP FP THEN 11. OVER XPON - ALOG SWAP OVER * R→I
SWAP R→I SIMP2 END »
DUP '→SF' STO
BYTES → 65.5 #94E9h
```

This creates a program called `'→SF'` ("to Simple Fraction").
Input: Any decimal fraction (a floating point number w/ non-zero fractional part).
Output: Equivalent Simple Fraction.

Example: `3.14159265359 →SF` $\rightarrow \frac{314159265359}{100000000000}$.

Here's a method for doing steps 2 through 5 manually:

Step 2: Rewrite the simple fraction as a mixed number: `PROPFRAC`. Write down the integer part.

Step 3: Remove the integer part: `« DUP IP R→I - EVAL »` (or just subtract the integer that you see in the display, and simplify by pressing `EVAL`).

Step 4: Reciprocate: `INV` (the 1/x key).

Repeat steps 2 through 4 until no fraction part remains.

Step 5: Write the accumulated integer parts in a list. Done.

Here's a User RPL program for doing steps 2 through 5 automatically. It converts any simple fraction into its equivalent Continued Fraction in one step:

```
« FXND DEPTH DUP 1 + ROLLD
  DO SWAP OVER IDIV2 ROT SWAP DUP
  UNTIL 0 ==
  END DROP2 DEPTH ROLL DEPTH SWAP - 1 + →LIST »
DUP '→CF' STO
BYTES → 81 #C386h
```

This creates a program called `'→CF'` ("to Continued Fraction").
Input: Any simple fraction (as an algebraic object).
Output: Equivalent Continued Fraction (as a list).

Example: `'13/46' →CF → {0 3 1 1 6}`

This means that $\frac{13}{46} = 0 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{6}}}}$.

Use both programs in sequence to convert a decimal fraction into a continued fraction.

Example: `3.1416 →SF →CF → {3 7 16 11}`

This means that $3.1416 = 3 + \cfrac{1}{7 + \cfrac{1}{16 + \cfrac{1}{11}}}$

Chapter II
How to convert continued fractions into simple fractions & decimal fractions.

Part A: Step-by-step example.

Example: Convert the continued fraction {3 7 16 11} into its equivalent simple fraction and decimal fraction.

Step 1: Put list on the stack and press EVAL. Four stack levels are now occupied.

Step 2: Press 1/x + repeatedly until all the numbers are used up. This converts the shorthand notation {3 7 16 11} into the full notation $3+\cfrac{1}{7+\cfrac{1}{16+\cfrac{1}{11}}}$.

Step 3: Press EVAL and see the answer: $\frac{3927}{1250}$.

Optional Step 4: To convert that into a decimal fraction, press →NUM and see the answer: 3.1416.

Part B: Generalized method.

To convert a continued fraction into a simple fraction or a decimal fraction:

Step 1: Put the continued fraction on the stack in list form and press EVAL.
Step 2: Press 1/x + repeatedly until all the numbers are used up.
Step 3: Press EVAL. See the equivalent simple fraction.
Optional Step 4: Press →NUM. See the equivalent decimal fraction.

Part C: RPL implementation.

```
« LIST→ 1 0 1 4 ROLL
  START OVER 4 ROLL * + SWAP
  NEXT / »
DUP 'CF→' STO
BYTES → 47.5 #1900h
```

This creates a program called 'CF→' ("Continued Fraction out").
Input: Any Continued Fraction (as a list).
Output: Equivalent simple fraction.

To convert the output to a decimal fraction, just press →NUM, of course.

Example: Convert the continued fraction {3 7 16 11} into its equivalent simple fraction and decimal fraction.

```
{3 7 16 11} CF→  →
```
$\dfrac{3927}{1250}$.

```
→NUM  →
```
3.1416.

Chapter III
Fun application: Opinion Polls

Newspapers usually report the results of opinion polls as percentages. For example, they might report that 42.86% of the people in a recent opinion poll responded "Yes." This looks impressive, subconsciously implying that many people were polled. Of the very few readers for whom that thought actually bubbles into their awareness, most would rationalize that feeling thus: "42.86% means that 4286 out of every 10000 people responded Yes," implying that ten thousand people were polled (if not more). Of those few readers, fewer still would take the further step of reducing the fraction $\frac{4286}{10000}$ to the equivalent fraction $\frac{2143}{5000}$, thus recognizing that as few as 5000 people may have been polled, with 2143 of them responding "Yes."

However, none of these conclusions are correct, because they ignore the fact that statistics are *rounded off*. What the newspaper is really saying, therefore, is that the percentage of Yes responses may not have been exactly 42.86% at all, but was something which merely *rounds* to 42.86%. Therefore, the *actual* percentage could have been anything between 42.855% (inclusive) and 42.865% (exclusive).

So the question becomes: What is the *smallest* number of respondents which can make such a ratio possible? In other words, what is the "simplest fraction" (the ratio of the smallest possible numbers) in the interval [0.42855, 0.42865)?

The only way to solve that problem is with continued fractions.

First, convert both ends of the interval into a continued fraction. Let's use the programs listed above.

```
.42855 →SF →CF  →  {0 2 2 1 951 1 2}
.42865 →SF →CF  →  {0 2 3 259 2 5}
```

The lists are the same until the third terms. Truncate the list there, *using the smaller number (2 here) plus 1:*

```
{0 2 3} CF→  →  '3/7'
→NUM  →  .428571428571
```

As you can see, 3/7 rounded to four decimal places is .4286, so the statistic “42.86%” could have been achieved with a polling sample of only *SEVEN* people, of whom 3 responded “Yes”.

Example #2:

We asked all of the students at Wassamada University whether they preferred vanilla or chocolate. Those who preferred neither were forced to choose one anyway. 68.421% of the students responded that they preferred chocolate. How many students attend Wassamada U? Ten thousand, as the statistic seems to imply? Not necessarily. What is the “simplest fraction” in the interval [.684205, .684215)?

```
.684205 →SF →CF  → {0 1 2 6 501 10 2}
.684215 →SF →CF  → {0 1 2 5 1 618 2 1 5}
```

Truncate the list where they begin to differ, *using the smaller number (5 here) plus 1:*

```
{0 1 2 6} CF→  → '13/19'
```

$\frac{13}{19}$ is therefore the simplest fraction that rounds to 68.421%. Therefore it’s statistically possible that Wassamada U has only 19 students, of whom 13 prefer chocolate.

The rule “Truncate the lists where they begin to differ and use the smaller number plus one” might sound peculiar. Here’s a good example of that rule in action: What is the “simplest fraction” between $\frac{21}{31}$ and $\frac{41}{61}$, non-inclusive?

```
'21/31' →CF  → {0 1 2 10}
'41/61' →CF  → {0 1 2 20}
```

Truncate the list where they begin to differ, and use the smaller number (10 here) *plus 1:*

```
{0 1 2 11} CF→  → '23/34'
```

Therefore $\frac{23}{34}$ is the “simplest fraction” between $\frac{21}{31}$ and $\frac{41}{61}$, that is to say, every other fraction in that interval has a numerator larger than 23 and a denominator larger than 34.

Homework: Find the simplest fraction between π and $\sqrt{10}$. Hint: you already know it.

Chapter IV
Glossary

Fraction: The ratio of two numbers; the relation between two similar magnitudes with respect to the number of times the first contains the second.

Common Fraction: a fraction represented as a numerator above, and a denominator below, a horizontal line, e.g. $\frac{\pi}{\phi}$, or a diagonal line, e.g. ${}^{\pi}\!/_{\phi}$ or π/ϕ. The line is known as a "virgule", "solidus", "slash", or "stroke", but is usually read as "over" (e.g. $\frac{\pi}{\phi}$ is read "pi over phi") or "divided by" ("pi divided by phi"). Children are taught to use a special "division sign" instead of a slash (e.g. $6 \div 2 = 3$), but this sign is rarely used in higher math, even though it appears on the division key of even the most advanced handheld calculators. Children are also taught to represent $126 \div 2 = 63$ as $2\overset{\displaystyle 63}{\overline{\big)126}}$, and read it as "2 goes into 126, 63 times."

Decimal Fraction: a fraction whose denominator is some power of 10, and is represented by a dot ("decimal point" or "point") (or a comma in Europe and some other places) written with the numerator, e.g. 0.75 which is equivalent to the simple fraction $\frac{75}{100}$. In RPL calculators, decimal fractions are called "reals". In common parlance they are called "floating point numbers".

Simple Fraction: the ratio of an integer to a natural number, e.g. $\frac{22}{7}$. According to this strict definition, the numerator may be zero or negative, but not the denominator.

Compound Fraction: a fraction whose numerator and/or denominator contain one or more fractions.

Integer: A number with no fractional part; a member of the set {… -2, -1, 0, 1, 2, …}.

Whole Number: A non-negative integer; a member of the set {0, 1, 2, …}.

Natural Number: A non-zero whole number; a member of the set {1, 2, 3, …}, which is also known as the set of "Counting Numbers".

Proper Fraction: a fraction whose numerator is less than the denominator, e.g. $\frac{7}{11}$.

Improper Fraction: a fraction whose numerator is greater than the denominator, e.g. $\frac{11}{7}$.

Mixed Number: the sum of an integer and a proper fraction, e.g. $3+\frac{1}{7}$.

Continued Fraction: a compound fraction whose denominator contains a fraction whose denominator contains a fraction, and so on, e.g. $1+\dfrac{2}{3+\frac{4}{5}}$ which is equivalent to the simple fraction $\frac{29}{19}$. Continued fractions can contain a finite or infinite number of fractions.

Simple Continued Fraction: a continued fraction all of whose numerators are 1, e.g. $5+\dfrac{1}{3+\frac{1}{7}}$. Mathematical shorthand notation for simple continued fractions is a list of just the integer parts, ignoring the numerators which are always 1; the above example would be written as {5, 3, 7}.

Rational Number: A number which can be represented as a simple fraction, e.g. the terminating decimal 0.5 which can be represented as $\frac{1}{2}$, or the repeating decimal $0.\overline{3}$ which can be represented as $\frac{1}{3}$. All rational numbers can be represented as either a terminating decimal or a repeating decimal.

Irrational Number: A number which cannot be represented as a simple fraction, e.g. π, which is not exactly equal to any simple fraction. When represented as *decimal fractions*, irrational numbers neither terminate nor repeat. However, the *continued fraction* representation of some irrational numbers repeat (e.g. the square roots of non-square whole numbers); some contain some other non-repeating but obvious pattern (e.g. Euler's constant *e*); and some seem utterly patternless (e.g. π).

Terminating Decimal: A decimal fraction which has a finite number of non-zero digits, e.g. 0.125 (equal to $\frac{1}{8}$). Also known as a "finite decimal."

Non-terminating Decimal: A decimal fraction which contains an infinite number of non-zero digits, e.g. π or $\frac{1}{7}$, neither of which terminate when written as a decimal fraction. Also known as an "infinite decimal."

Repeating Decimal: a decimal numeral that, after a certain point, consists of a group of one or more digits repeated ad infinitum, e.g. $1.2\overline{34}$ (with the overscore meaning that those digits repeat forever; this example is equal to $\frac{611}{495}$, and can also be represented as 1.2343434…, with the ellipsis meaning "and so on"). Also known as a "circulating decimal", "periodic decimal", and "recurring decimal."

Simplest Fraction: The unique simple fraction with the smallest numerator and denominator which lies within a given interval. For example, the simplest fraction within the interval $\pi \pm 0.0005$ is $\frac{267}{85}$, because all the other simple fractions in that interval have larger numerators and denominators. The simplest fraction in the interval [.4, .6] is obviously ½. The simplest fraction in the interval [.12345, .12346] is $\frac{10}{81}$, which is less obvious, but is easily calculated using continued fractions, as shown in Chapter III above.

PDQ Algorithm: A method for calculating the simplest fraction in any given interval, performing all calculations on the integer domain, thus avoiding all round-off errors. Discovered by Joseph Horn (impatient Number Theory student), and optimized by Rodger Rosenbaum (patient Number Theory master) and Tony Hutchins (inveterate RPL programmer). It solves the Wassamada U example above in one step, like this:

```
.68421 .000005 PDQ → '13/19'
```

How Well Do You Know RPN?

HP Solve #30 page 15

QUIZ - How Well Do You Know RPN?

Richard J. Nelson

Introduction

The most distinctive feature of HP calculators is the use (on many models) of RPN**[1]**. Since HP RPN is older (40 years) than many current HP people it is understandable that RPN has evolved with subtle changes. These changes have been previously documented, explained, and defined in ***HP Solve*** issue #27, page 42 in an article titled HP RPN Evolves. This article differentiated RPN as either classical RPN or ENTRY RPN.

Today HP more broadly defines RPN based on the general post fix logic used for problem solving rather than as an automatic stack based on specific rules**[2]**. How well do you know (classical) RPN? Take the following quiz and then get your rating at the end of this column. It will be fun and I promise that you will learn something. Each question is worth one point.

The Quiz

Do not refer to any references other than a piece of blank paper, a pen, and your favorite Classical RPN calculator. Each answer is worth one point. The five classical RPN commands are: ENTER, ↑; X⇄Y; R↓; LAST X; and R↑.

1. The value of the Golden ratio is: $\frac{1 \pm \sqrt{5}}{2}$. The pressed keys to calculate the positive value are: 1, ENTER, 5, √X, +, 2, ÷ is 7 ks. For a very easy point write a sequence that saves one keystroke.

ANS: ______________________________

2. Two Resistors (R_1 & R_2) connected in parallel have a total equivalent resistance, R_T, as follows.

$$R_T = \frac{R_1 R_2}{R_1 + R_2}$$

Assuming that R_1 & R_2 are on the stack, how many keystrokes, ks, are required to obtain a solution?

A – 9 ks, **B** – 8 ks, **C** – 7 ks, **D** – 6 ks, **E** – 5 ks. ANS: _____

3. Using a stack diagram as shown below write the keystrokes and stack register contents for your answer in question #2.

ANS:

Step		1	2	3	4	5	6	7	8	9
T	~									
Z	~									
Y	R_2									
X	R_1									
Press	Start									

~ is don't care.

4. How many keystrokes are required to reverse the order of the stack as shown below?

	Start		End
T	D		A
Z	C		B
Y	B	⇒	C
X	A		D

The shift key of any shifted functions on your favorite RPN machine need not be counted[(3)].

A – 7 ks, **B** – 6 ks, **C** – 5 ks, **D** – 4 ks, **E** – 3 ks. ANS: ______

5. Using a stack diagram as shown below write the keystrokes and stack register contents for your answer in question #4.

ANS:

Step		Start	1	2	3	4	5	6	7
	T	4							
	Z	3							
	Y	2							
	X	1							
	Press		____	____	____	____	____	____	____

~ is don't care.

6. There are two methods (not including data register usage) of using a constant in RPN. The first is replication of the T register. What is the second? ANS: ______________________________

7. Is converting a classical RPN program to an ENTRY RPN program easy and always works?

True or False ANS: ______

8. How may you terminate an entry without pressing a function key? ANS: ____________________

9. An event counter may be accomplished by storing 1's on the stack and using the T register replication feature to add 1 to the X register each time the + key is pressed. What other technique may be used to accomplish the same thing – pressing a key and incrementing a counter? Programming is not allowed. ANS: ____

10. Which operator typically does NOT alter the LAST X register?
A. +, **B**. -, **C**., 1/X, **D**. x, **E**. ÷, **F**. none. ANS: _____

Bonus Question

11. Solve the Mach Equation.

$$M = \sqrt{5\Biggl[\Bigl(\Bigl\{\Bigl[\Bigl(1 + 0.2\Bigl[\frac{350}{661.5}\Bigr]^{2}\Bigr)^{3.5} - 1\Bigr]\Bigl[1 - (6.875 \times 10^{-6})\,25{,}500\Bigr]^{-5.2656}\Bigr\} + 1\Bigr)^{0.286} - 1\Biggr]}$$

ANS: ____________________

The answers are after the three notes.

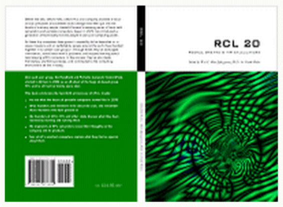
Notes: Quiz - How Well Do You Know RPN?

(1) *RPN is an acronym for Reverse Polish Notation. Any name beginning with Reverse starts out with a negative image for first time encounters with users. The name is so sensitive that users have been debating about it for decades. A presentation by Wlodek Mier-Jedrzejowicz at HHC 2012 even explored RPN using a chapter titled "Really Pathetic Notation" from the book RCL 20. See the HHC 2012 Report in* ***HP Solve*** *issue # 29 page 11 titled Hewlett-Packard Handheld Conference #39. Note (10) is reproduced below.*

(10) The book is RCL 20.which may be found at: http://www.limov.com/rcl20/

Bill Wickes' article may be found on page 105.

Many (22) of the HPUC leaders contributed To this book.



RCL 20: People, Dreams & HP Calculators
W.A.C. Mier-Jedrzejowicz Ph.D. & Frank Wales (Eds)
2002
ISBN: 0-9510733-3-8

Fig. 13 – RCL 20 book records the people history of HPCC.

Here is a video link to Wlodek's presentation.   http://www.youtube.com/watch?v=qRrAj-GCTQM

(2) *A test for the type of RPN an HP machine uses is quoted from the* ***HP Solve*** *article. "There are two basic forms of HP RPN. Classical RPN and Entry RPN. While they are very similar overall in that they are both a postscript user interface there are subtle differences, a few of which are described."*

"An example of a current model Entry RPN machine is the HP 30b. If you place the machine in RPN mode, and have a clear stack, you may compare the stack operation with a Classical RPN machine such as the HP 15C. Press 5, ENTER. Next press x. On the 30b you will see 0 because the Y register is zero and 0 x 5 = 0. On the 15C you will see 25 because the ENTER, ↑, raised the stack as shown in the three cases above."

See the article in the issue at the link below. Table 1 lists of all of HP's calculators as to which RPN they use. Also see Note 2.

http://h20331.www2.hp.com/hpsub/downloads/HP_Calculator_eNL_04_April_2012%20(2).pdf

(3). *For a table (by Jake Schwartz) of HP calculator stack commands and the keystrokes required to execute them see Table 1 (page 5 of 7) in* ***HP Solve*** *issue #4 of the article titled Introduction to RPN Tips V5.*

See the Answers on the next page. Count your points and look up your number in Table 1 at the end.

Answers: RPN Quiz - How Well Do You Know RPN?

Q 1. By changing the order of the operations the ENTER may be eliminated. 5, √X , 1, +, 2, ÷ is **6 ks**.

Q 2 & 3. Q #2: Take one point if you answered B, or E.
Q #3: Take one point if your stack diagram matches one of the three below.

<u>8 ks solution</u> Uses stack replication and Last X.

Step		1	2	3	4	5	6	7	8
T	~	~	R_1	R_2	R_2	R_2	R_2	R_2	R_2
Z	~	R_1	R_2	R_2	R_2	R_2	$R_{1x}R_2$	R_2	R_2
Y	R_1	R_2	R_2	R_2	R_2	$R_{1x}R_2$	R_2	$R_{1x}R_2$	R_2
X	R_2	R_2	R_2	R_1	$R_{1x}R_2$	R_2	R_1	$R_{2+}R_1$	$\frac{R_1R_2}{R_1+R_2}$
Press	Start	↑	↑	R↑	x	X⇄Y	LASTX	+	÷

~ is don't care.

<u>8 ks solution</u> Uses Last X only.

Step		1	2	3	4	5	6	7	8
T	~	~	R_2	R_2	R_2	R_2	~	~	~
Z	~	R_1	~	~	R_2	~	$R_{1x}R_2$	~	~
Y	R_1	R_2	R_1	R_2	~	$R_{1x}R_2$	R_2	$R_{1x}R_2$	~
X	R_2	R_2	R_2	R_1	$R_{1x}R_2$	R_2	R_1	$R_{2+}R_1$	$\frac{R_1R_2}{R_1+R_2}$
Press	Start	↑	R↓	X⇄Y	X	R↑	LASTX	+	÷

~ is don't care.

<u>5 ks solution</u> Uses alternate algebraic form of the equation:

$$R_T = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

Step		1	2	3	4	5
T	~	~	~	~	~	~
Z	~	~	~	~	~	~
Y	R_1	R_1	$1/R_2$	$1/R_2$	~	~
X	R_2	$1/R_2$	R_1	$1/R_1$	$1/R_{2+}1/R_1$	$1/(1/R_{2+}1/R_1)$
Press	Start	1/X	X⇄Y	1/X	+	1/X

~ is don't care.

Is there a minor accuracy caution that should be considered with this solution?

Q 4 & 5. Q #4: Take one point if you answered D.
Q #5: Take one point if your stack diagram matches one of the two below

<u>4 ks solution</u> Uses Roll Down, R↓ (primary operator on most RPN machines)

Step		1	2	3	4
T	4	4	2	1	1
Z	3	3	4	2	2
Y	2	1	3	4	3
X	1	2	1	3	4
Press	Start	X⇄Y	R↓	R↓	X⇄Y

<u>4 ks solution</u> Uses Roll Up, R↑ (often not found or is a shifted operator on most RPN machines)

Step		1	2	3	4
T	4	4	3	1	1
Z	3	3	1	2	2
Y	2	1	2	4	3
X	1	2	4	3	4
Press	Start	X⇄Y	R↑	R↑	X⇄Y

Q 6. By using the LAST X register. An example is a currency or other conversion factor. The Z and T registers are retained after multiple uses of the conversion calculation. See ***HP Solve*** RPN Tip # 12 in issue 12.

Q 7. False. An extra ENTER may be required for many stack operations for more modern machines using ENTRY RPN.

Q 8. By pressing X⇄Y twice.

Q 9. By using the statistical function Σ+ key. The number of data entries is tallied and this may be used as an event counter. See ***HP Solve*** RPN tip # 11 in issue 11.

Q 10. F. Test by filling the stack with zero and pressing + to store zero in LAST X. Perform each operation followed by LAST X.

Q 11. This is a classical calculator problem that has appeared in many manuals for the earlier HP RPN machines and it provides a good test of user RPN skill. For a complete discussion of all aspects of solving this equation see an article titled <u>HP Algebraic</u> by Palmer Hanson in ***HP Solve*** Issue #22 pages 12 - 15**.** **If you got 0.835724536 take one point**. You probably pressed 61 or more keys - 62 keys on an HP 35s.

How many points did you get? Include the bonus point if you answered # 11. Determine your rating from the table below.

Table 1 – Points vs. Skill/title

Points	**Skill/title**
11 or 10	RPN author, Super user
9 or 8	Expert
7 or 6	Average user

Points	**Skill/title**
5 or 4	Casual user
≤3	New user
0	History major

If you have been an avid reader/student of ***HP Solve*** you should be an Average user or better.

HP 39gII Regression: Part I

HP Solve #30 page 21

HP 39gII Regression: Part I
Exploring Statistical Regression with the HP 39gII

Namir Shammas

Introduction

The advent of programmable calculators provided scientists, engineers, mathematicians, and statisticians with personal computing devices that performed various statistical calculations. These calculations included regression analysis to perform various kinds of curve fitting. The newer machines that appeared on the market supported more advanced regression calculations. The HP 39gII follows this trend very faithfully. The calculator offers various type of two-variable curve fitting. This includes linear, logarithmic, exponential, power, inverse, quadratic, cubic, quartic, trigonometric, logistic, and user-defined curve fits. This list is versatile and impressive, putting the HP 39gII on par, if not slightly better, with other bestselling graphing calculators. The machine also support matrix/vector calculations and offers the **LSQ** function that performs, in one swoop, the core least-squares calculations used in regression analysis. This article is the first of a series that explores the HP 39gII regression analysis capabilities beyond the built-in regression features. If you like what the HP 39gII already does for two-variable regressions, you will really enjoy these articles. They are akin to injecting regression steroids in the HP 39gII. In this article I discuss the following topics:

1. The basics of working with the **LSQ** function with matrix/vector calculations to support regression calculations.

2. Using an HP 39gII function that calculates and returns the regression coefficients and the coefficient of determination of a general multiple regression model.

3. Polynomial regression.

4. Power curve fitting for two or more variables.

☞ In this series of articles, I use the term *regression model* to mean the equation that is used in the regression calculations to describe the relationship between a dependent variable and one or more independent variables.

The Basics

The built-in function **LSQ(X, y)** performs least-squares regression calculations on the matrix **X** and the column vector **y**. The matrix **X** has multiple columns that represent values for independent variables and/or their transformations (such as natural logarithm, reciprocal, and square, just to name a few). The first column in matrix **X** is typically filled with the constant 1 to allow **LSQ** to calculate a constant for the fitted regression model. The column vector **y** contains the values of the dependent variable or its transformations. The function **LSQ** calculates the constant and regression coefficients by evaluating the following matrix equation:

$$\text{LSQ}(\mathbf{X},\mathbf{y}) = (\mathbf{X}^T\,\mathbf{X})^{-1}\,(\mathbf{X}^T\,\mathbf{y})$$

The beauty of using matrix/vector operations is that they calculate the matrix and vector containing the statistical summations using few high-level operations. Thus, the result of $\mathbf{X}^T\,\mathbf{X}$ is a matrix that has the statistical summations for the independent variables and/or their transformations. The result of $\mathbf{X}^T \mathbf{y}$ is a column vector that contains the statistical summations for the product between the dependent and independent variables and/or their transformations. The matrix expression $(\mathbf{X}^T \mathbf{X})^{-1} (\mathbf{X}^T \mathbf{y})$ solves for the regression coefficients. Using the matrix operations reduces significantly the size of the source code needed to calculate the regression coefficients. If you read source code for statistical calculations, in languages like C++ and Visual Basic, you will be stunned! You will realize how much lower-level coding is needed, in these programming languages, to accumulate the values in the various statistical summations. You will also recognize how much more coding is needed to calculate the regression coefficients! The function **LSQ** is a wonderful gift from the designers of the HP 39gII.

The simplest case of using function **LSQ** is to provide it with values from the predefined matrices M0 through M9. Select one of these matrices to represent the matrix **X** and another matrix to represent the column vector **y**. The selected matrices would have an equal number of rows of raw data. When using the function **LSQ** inside a program function, its arguments can be locally defined matrices and vectors (which are really single-column matrices). Table 1 shows some sample data. The variables x_1, x_2, and x_3 are the independent variables. The variable y is the dependent variable.

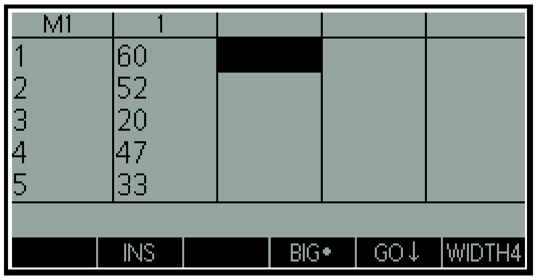
x_1	x_2	x_3	y
7	25	6	60
1	29	15	52
11	56	8	20
11	31	8	47
7	52	6	33

Table 1 – Sample data.

To calculate the coefficients of the following simple multiple regression model:

$y = a + b\, x_1 + c\, x_2 + d\, x_3$

You need to store the values of y in matrix M1 and the values of the independent variables in matrix M2. Figure 1 shows the contents of matrix M1 which stores the values for the column vector **y**.



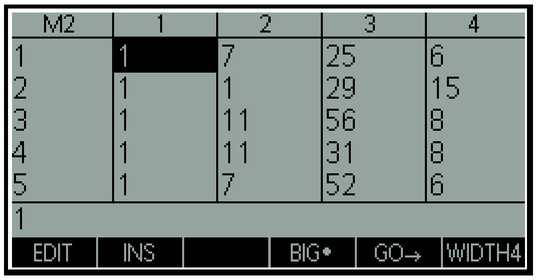
M1	1
1	60
2	52
3	20
4	47
5	33

Fig. 1 – The values in matrix M1.

Figure 2 shows the contents of matrix M2 which stores the values for the matrix **X**. Notice that the first column in matrix M2 is filled with the constant 1. The values for the independent variables occupy columns 2, 3, and 4.

Figure 3 shows the result of executing the command **LSQ(M2,M1)**.

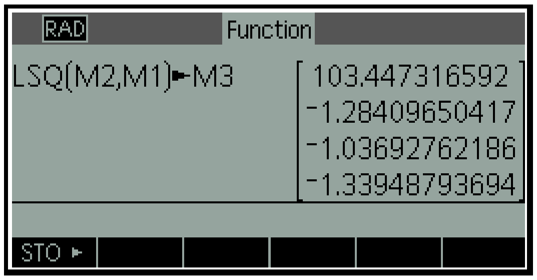
The model fitted (with the coefficients rounded to four decimal places) in Figure 3 is:



M2	1	2	3	4
1	1	7	25	6
2	1	1	29	15
3	1	11	56	8
4	1	11	31	8
5	1	7	52	6

```
1
```

Fig. 2 – The values in matrix M2.



```
LSQ(M2,M1)▸M3      [ 103.447316592  ]
                   [ -1.28409650417 ]
                   [ -1.03692762186 ]
                   [ -1.33948793694 ]
```

Fig. 3 – The results of executing function LSQ.

$y = 103.4473 – 1.2841\ x_1 – 1.0370\ x_2 – 1.3395\ x_3$

I have presented you with a bare-bone example of using the function **LSQ**. The example is simple because it uses the variables without performing any mathematical transformation on them. However, the calculations do not indicate how good the model is. One of the popular statistics used to indicate the goodness of fit is called the *coefficient of determination*, R^2. This statistic, which is the square of the *correlation coefficient*, indicates the amount of the variance in variable y that is explained by the regression model we obtain. When the coefficient of determination is 1, its largest value, we have a perfect fit. This means that the regression model accounts for all of the observed values of y. On the other extreme, when the coefficient of determination is 0, its smallest value, it means that the regression model has completely failed to explain the variation in variable y. The value for R^2 is calculated using the following equation:

$$R^2 = \frac{\sum_1^n(\hat{y}_i - \bar{y})^2}{\sum_1^n(y_i - \bar{y})^2}$$

Where $\widehat{y_i}$ is the predicted value of y at the values of the independent variables used in the calculations, $\bar{y}$ is the average value of y, and y_i is the values of y entering in the regression calculations. Keep in mind that the coefficient of determination cannot tell how well the regression model can predict values of y that are based on *new* values for the independent variable(s).

First Things First!

After showing you how to use the function **LSQ** and presenting the coefficient of determination, let me present the function **MLR2** which performs the calculations for the regression coefficients and also the coefficient of determination. This function is the first step in automating regression analysis calculations. Table 2 shows the source code for function **MLR2**. This function has the following parameters:

- The parameter **MatX** which represents the matrix **X**.
- The parameter **VectY** which represents the vector **y**.

The function returns a list containing the coefficient of determination and the regression coefficients (as a column matrix).

Statement

```
EXPORT MLR2(MatX,VectY)
BEGIN
  LOCAL i,j,y,Rsqr;
  LOCAL lstDimX,NumRowsX;
  LOCAL YMean,Sum1,Sum2,Yhat,RegCoeff;

  // calculate the regression coefficients
  RegCoeff:=LSQ(MatX,VectY);
  // calculate the number of data points
  lstDimX:=SIZE(MatX);
  NumRowsX:=lstDimX(1);
  // calculate ymean
  Sum1:=0;
  FOR i FROM 1 TO NumRowsX DO
    y:=VectY(i,1);
    Sum1:=Sum1+y;
  END;
  YMean:=Sum1/NumRowsX;

  // calculate the coefficient of determination
  Sum1:=0;
  Sum2:=0;
  Yhat:=MatX*RegCoeff;
  FOR i FROM 1 TO NumRowsX DO
    Sum1:=Sum1+(Yhat(i,1)-YMean)^2;
    Sum2:=Sum2+(VectY(i,1)-YMean)^2;
  END;
  Rsqr:=Sum1/Sum2;
  // return the results
  RETURN {Rsqr,RegCoeff};
END;
```

Table 2 –The source code for function MLR2.

The function **MLR2** performs the following tasks:

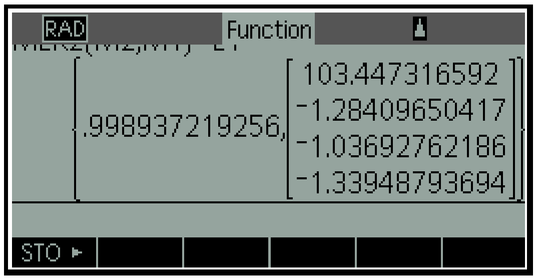
- Calculates the regression coefficients using the function **LSQ**. The function stores the result of function **LSQ** in the local variable **RegCoeff**.
- Calculates the mean for the y values.
- Calculates the coefficient of determination using the definition I presented in the last section. Notice that the function calculates the array of $\hat{y}$ values using the matrix/vector expression

- **MatX*RegCoeff**. This expression multiplies the matrix **MatX** with the column vector **RegCoeff**. The second **FOR** loop calculates the sums of squared differences between $\widehat{y_i}$ and $\bar{y}$ and also between y_i and $\bar{y}$.

Let's use function **MLR2** with the data in Table 1. This time, function **MLR2** yields the regression coefficients *and* the coefficient of determination. To use function **MLR2** with the data in matrix M1 and M2, type the following command:

```
MLR2(M2,M1))→L1
```

The above command stores the results in list L1 for further examination if so desired. Figure 4 shows the results of executing the function **MLR2(M2,M1)**:



```
{.998937219256, [ 103.447316592  ]
                [ -1.28409650417 ]
                [ -1.03692762186 ]
                [ -1.33948793694 ]}
```

Fig. 4 – The results of executing function MLR2.

The results in Figure 4 show that R^2 is 0.99893 and the regression coefficients are the same values obtained in the last section. Thus, the fitted model explains 99.89% the variation in the observed values of y. This high value is expected since we are fitting five data points with a regression model that has a total of four regression coefficients. There is little room for variation in variable y that the model cannot explain.

Polynomial Fitting

This section presents an HP 39gII function that allows you to perform polynomial regression between two variables, x and y. The function calculates the coefficients for the polynomial and also returns the coefficient of determination.

Table 3 shows the source code for the function **PolyReg**. This function has the following parameters:

- The parameter **DSMat** represents the matrix that has the x and y data.
- The parameter **SelXCol** designates the column in matrix **DSMat** which contains the values for x.
- The parameter **SelYCol** designates the column in matrix **DSMat** which contains the values for y.
- The parameter **Order** selects the order for the polynomial regression. If you pass an argument of 1 to this parameter, the function **PolyReg** performs a linear regression. Passing values of 2 and 3 to the parameter **Order** causes the function to fit the data with quadratic and cubic polynomials, respectively. Remember that the HP 39gII can fit data with up to fourth order polynomials. The function **PolyReg** can perform the same polynomial fits and go beyond.

The function **PolyReg** returns a list that contains the value of the coefficient of determination and a column matrix containing the regression coefficients. I recommend that you store the results of calling function **PolyReg** in a list so that you can further examine and/or use the results.

Statement
`EXPORT PolyReg(DSMat,SelXCol,SelYCol,Order)`
`BEGIN`
`  LOCAL i,j,x,y;`
`  LOCAL lstDimX,NumRows;`
`  LOCAL MatX, VectY, RegCoeff;`
`  LOCAL YMean,Sum1,Sum2,Yhat,Rsqr;`
`  lstDimX:=SIZE(DSMat);`
`  NumRows:=lstDimX(1);`
`  // initialize matrix and vector`
`  MatX:=MAKEMAT(1,NumRows,Order+1);`
`  VectY:=MAKEMAT(1,NumRows,1);`
`  Sum1:=0;`
`  // populate matrix X and vector y with data`
`  FOR i FROM 1 TO NumRows DO`
`    y:=DSMat(i,SelYCol);`
`    x:=DSMat(i,SelXCol);`
`    VectY(i,1):=y;`
`    Sum1:=Sum1+y;`
`    FOR j FROM 1 TO Order DO`
`      MatX(i,j+1):=x^j;`
`    END;`
`  END;`
`  // calculate ymean`
`  YMean:=Sum1/NumRows;`
`  // calculate regression coefficients`
`  RegCoeff:=LSQ(MatX,VectY);`
`  // calculate the sums of squares of`
`  // (yhat - ymean) and (y - ymean)`
`  Sum1:=0;`
`  Sum2:=0;`
`  FOR i FROM 1 TO NumRows DO`
`    y:=DSMat(i,SelYCol);`
`    x:=DSMat(i,SelXCol);`
`    Yhat:=RegCoeff(1,1);`
`    FOR j FROM 1 TO Order DO`
`     Yhat:=Yhat+RegCoeff(j+1,1)*x^j;`
`    END;`
`    Sum1:=Sum1+(Yhat-YMean)^2;`
`    Sum2:=Sum2+(y-YMean)^2;`
`  END;`
`  // calculate coefficient of determination`

```
   Rsqr:=Sum1/Sum2;
   RETURN {Rsqr,RegCoeff};
END;
```

Table 3 –The source code for function PolyReg.

Let's use the function **PolyReg** to fit a cubic polynomial using the data in Table 4. Enter the data in matrix M1.

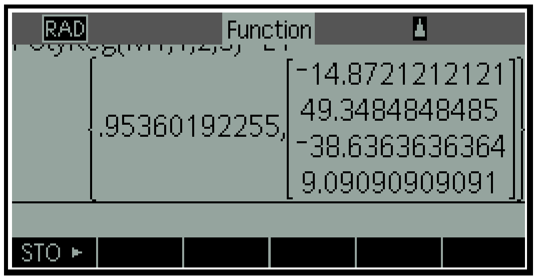
x	*y*
1	5.1
1.1	4.4
1.2	4.6
1.3	4.0
1.4	3.2
1.5	3.2
1.6	2.4
1.7	2.2
1.8	1.3
1.9	2.0

Table 4 – Sample data for a cubic polynomial fit.

To obtain the regression coefficients and coefficient of determination for the cubic polynomial fit using the data in Table 4, execute the following command:

```
PolyReg(M1,1,2,3)→L1
```

The first argument of calling function **PolyReg** is the matrix M1 which contains the (x, y) data points. The second argument is 1 which tells the function that the values for the independent variable x are in column 1 of M1. The third argument is 2, which tells the function that the values for the dependent variable y are in column 2 of M1. The last argument is 3, which specifies the order of the sought polynomial. I assigned the results to list L1 so that I can examine the results later, if I needed to. Figure 5 shows the output of using function **PolyReg**.



```
              ⎡-14.8721212121⎤
 {.95360192255, 49.3484848485 }
              ⎢-38.6363636364⎥
              ⎣ 9.09090909091⎦
```

Fig. 5 – The results of executing function PolyReg.

The results show that R^2 is 0.953602 and the fitted polynomial is:

$y = -14.8721 + 49.3485\,x - 38.6364\,x^2 + 9.0909\,x^3$

The value of R^2 indicates that the polynomial explains about 95% of the variation in the values of y.

Power Curve Fitting

Power curve fits allow you to create models where the relations between the logarithmic values of the variables are linear. In the case of two variables we have:

$y = a\ x^b$

Which is the same as the linear form:

$\ln(y) = \ln(a) + b \ln(x)$

In the case of multiple variables, such as:

$y = a\ x_1^b\ x_2^c\ x_3^d$

The linear form is:

$\ln(y) = \ln(a) + b \ln(x_1) + c \ln(x_2) + d \ln(x_3)$

Creating a program function that performs power curve fits for two or more variables is really simple. The HP 39gII function has to translate the input data into logarithmic values and then use the **LSQ** function with the matrix and vector of the logarithmic values. Table 5 shows the source code for the function **PowerFit**. The function **PowerFit** has two parameters. The first parameter is **MatX**--the name of the matrix that contains the values for the independent variables. The second parameter is **VectY**—the name of the column vector that contains the values for the dependent variable. The function returns a list containing the coefficient of determination and the column matrix that stores the regression coefficients.

Statement
`EXPORT PowerFit(MatX,VectY)`
`BEGIN`
`  LOCAL i,j;`
`  LOCAL lstDimX,NumRowsX,NumColsX;`
`  LOCAL TMatX,TVectY,RegCoeff,Rsqr;`
`  LOCAL YMean,Sum1,Sum2,Yhat;`
`  // get the number of rows and columns of matrix MatX`
`  lstDimX:=SIZE(MatX);`
`  NumRowsX:=lstDimX(1);`
`  NumColsX:=lstDimX(2);`
`  // create matrices to store transformed data`
`  TMatX:=MAKEMAT(1,NumRowsX,NumColsX+1);`
`  TVectY:=MAKEMAT(1,NumRowsX,1);`
`  FOR i FROM 1 TO NumRowsX DO`
`    TVectY(i,1):=LN(VectY(i,1));`
`    FOR j FROM 1 TO NumColsX DO`
`      TMatX(i,j+1):=LN(MatX(i,j));`
`    END;`
`  END;`
`  // calculate the regression coefficients`
`  RegCoeff:=LSQ(TMatX,TVectY);`

```
  // calculate ymean
  Sum1:=0;
  FOR i FROM 1 TO NumRowsX DO
    Sum1:=Sum1+TVectY(i,1);
  END;
  YMean:=Sum1/NumRowsX;

  // calculate the coefficient of determination
  Sum1:=0;
  Sum2:=0;
  Yhat:=TMatX*RegCoeff;
  FOR i FROM 1 TO NumRowsX DO
    Sum1:=Sum1+(Yhat(i,1)-YMean)^2;
    Sum2:=Sum2+(TVectY(i,1)-YMean)^2;
  END;
  Rsqr:=Sum1/Sum2;
  // return the results
  RETURN {Rsqr,RegCoeff};
END;
```

Table 5 –The source code for function PowerFit.

The source code in Table 5 shows that the function **PowerFit** creates the local matrices **TMatX** and **TVectY**. The function uses these matrices to store the logarithmic transformations of the data in matrices **MatX** and **VectY**. The function uses the local matrices **TMatX** and **TVectY** to calculate the regression coefficients, the mean of the logarithm of y values, and the coefficient of determination.

Let's test function **PowerFit** with the data in Table 6.

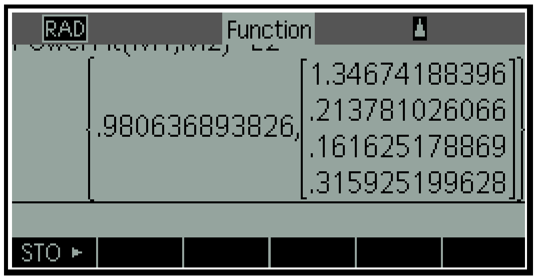
x	*z*	*t*	*y*
1	1	7	7
2	1	5	7.7
3	2	3	7.9
4	2	1	5.3
5	3	2	8.4
6	3	5	11.6
7	4	8	13.6
8	4	9	14.3
9	5	4	12.4
10	5	2	10.6

Table 6 – Sample data for a power fit.

Store the values of the first three columns of Table 6 in matrix M1. Store the values of the rightmost column of Table 6 in the matrix M2. To calculate the regression coefficient of a power fit between the independent variables x, z, t, and the dependent variable y, execute the following command:

```
PowerFit(M1,M2)→L2
```

Figure 6 shows the results of executing the above command.



```
{.980636893826,[[1.34674188396]
                [.213781026066]
                [.161625178869]
                [.315925199628]]}
```

Fig. 6 – The results of executing function PowerFit.

The results show that R^2 is 0.98063 and the power fit is:

y = 1.34674 + 0.213781026 ln(x) + 0.161625179 ln(z) + 0.3159252 ln(t)

Which also has the following nonlinear form,

y = 3.844878 (x^0.213781026) (z^0.161625179) (t^0.3159252)

The value of R^2 indicates that the power fit explains about 98% of the variation in the values of y.

About Entering the Source Code

I found it much easier to enter source code by first using an HP 39gII emulator. You can do that too by following these steps:

1. Search the Internet for the HP 39gII emulator software. Download and install the software on your PC
2. Using your PC, copy the source code from this and other articles into a text editor of your choice. Alternatively you can type in your own source code in a text editor, if you are writing your own functions. This step offers a significant time-saver.
3. Turn on the visible spaces in your text editor (choose a text editor that has this feature).
4. Locate and remove any unusual and extended ASCII characters. Also replace tab characters with spaces.
5. Save the source code to your PC.
6. Copy all of the source code into the PC's clipboard buffer.
7. Run the HP 39gII emulator and create the HP 39gII function for the code you wish to insert.
8. Select the new function for editing.
9. Delete all of the default source code that the emulator inserts for the new function.
10. Use the **Edit | Paste** menu commands in the emulator. This step pastes the source code from the clipboard buffer to the emulator.
11. Examine the source code to make sure that the pasting yielded a good listing. You can also use the **CHECK** command to detect errors in the source code. If these errors require minor editing, then

12. do so. If not go back to step 3. In most cases, this step goes without any problems.

13. Test the function in the emulator to make sure that it runs correctly. This step may require entering data in the predefined matrices. Correct runtime errors if they occur, by examining the source code. The **CHECK** command does not always catch all errors. Common undetected errors include (a) a missing colon in an assignment statement and (b) missing semicolons at the end of statements. If you are writing your own functions, then any errors in the source code you typed in will cause errors to show up in this step or in the last one.

14. Connect your PC to the physical HP 39gII calculator using a USB cable.

15. Select and copy the function from the emulator to the calculator by using the **SEND** command (available in Prgm mode).

16. Verify that the physical HP 39gII calculator shows the function you just copied.

Don't be intimidated by the number of the above steps. Once you get the hang of it, you can really speed up developing HP 39gII programs. The Pascal-like programming language is powerful and allows the HP 39gII to perform sophisticated calculations.

Observations and Conclusions

This article introduced you to regression analysis calculations with the HP 39gII calculator. You learned about using the built-in **LSQ** function. The article also presented functions that prepared the matrices and vectors used by **LSQ** to perform multiple regression, polynomial regression, and power fitting. In addition, these functions included code to calculate the coefficient of determination, which indicates the goodness of fit. The reader can conclude that the HP 39gII can perform regression calculations with ease using the built-in **LSQ** function and the matrix operations. These operations complement the built-in regression models the machine offers for two variables. Moreover, the matrix editor is a suitable tool to enter and edit data used in regression calculations.

The next article shows you how to perform linearized regression between two and three variables.

References

1. Wikipedia article *Coefficient of Determination*.

2. Wikipedia article *Linear Regression*.

3. Wikipedia article *Simple Linear Regression*.

4. Draper and Smith, *Applied Regression Analysis*, Wiley-Interscience; 3rd edition (April 23, 1998)

5. Neter, Kuther, Wasserman, and Nachtsheim, *Applied Linear Statistical Models*, McGraw-Hill/Irwin; 4th edition (February 1, 1996).

6. Fox, *Applied Regression Analysis and Generalized Linear Models*, Sage Publications, Inc; 2nd edition (April 16, 2008).

7. Montgomery, Peck, and Vining, *Introduction to Linear Regression Analysis*, Wiley-Interscience; 4th edition (2006).

8. Seber and Lee, *Linear Regression Analysis*, Wiley; 2nd edition (February 5, 2003).

About the Author

Namir Shammas is a native of Baghdad, Iraq. He resides in Richmond, Virginia, USA. Namir graduated with a degree in Chemical Engineering. He received a master degree in Chemical engineering from the University of Michigan, Ann Arbor. He worked for a few years in the field of water treatment before focusing for 17 years on writing programming books and articles. Later he worked in corporate technical documentation. He is a big fan of HP calculators and collects many vintage models. His hobbies also include traveling, music, movies (especially French movies), chemistry, cosmology, Jungian psychology, mythology, statistics, and math. As a former PPC and CHHU member, Namir enjoys attending the HHC conferences. *Email me at: nshammas@aol.com*

HP 39gII Regression: Part II

HP Solve #30 page 34

HP 39gII Regression: Part II
Linearized Regression

Namir Shammas

Introduction

This article shows you how to perform linearized regression between two and three variables. In addition, it presents functions that will help you perform regression analysis calculations on a wide variety of custom linearized equations. These functions allow you to select variables from matrices and apply temporary transformations that shift, scale, and raise to powers the values of these variables. Thus, these functions deliver very flexible transformation schemes.

☞ In this series of articles, I use the term *regression model* to mean the equation that is used in the regression calculations to describe the relationship between a dependent variable and one or more independent variables.

Linearized Regression for Two Variables

When fitting equations to data you often need to transform the original equations into a linear form. For example, consider the following equation:

$y = a\ x^b$

The above form can be linearized by taking the natural logarithm of both sides to yield:

$\ln(y) = \ln(a) + b \ln(x)$

Thus ln(y) and ln(x) have a linear relation. The intercept of that linear relation is ln(a) and the slope is b. Thus to calculate the value of a, you need to calculate the exponential value of the intercept.

Consider a second example:

$y = a\ x / (b + x)$

Again, you can linearize the above nonlinear equation by taking the reciprocal of both sides and obtaining the following linear form:

$1/y = 1/a + (b/a) / x$

In the above case, 1/y and 1/x have a linear relation. The intercept of that linear relation is 1/a and the slope is b/a. Thus to obtain the value of a, you need to calculate the reciprocal of the intercept. To calculate the value of b you need to divide the slope by the intercept.

Linearizing relations allow you to use linear or multiple-linear regression to calculate the regression coefficients. Keep in mind that the regression coefficients are transformed versions of the coefficients in the original nonlinear equations. Thus, you will need to perform a few extra calculations to obtain the coefficients of the nonlinear equations.

Table 1 presents the function **LinearizedReg**. This function supports linearized regression for two variables, x and y. The function has the following parameters:

- The parameter **DSMat** represents the matrix that has the x and y data.

- The parameter **lstSelXData** is a list that specifies data for variable x. The list contains an integer that selects the **DSMat** column storing values for variable x, and values to shift the values of x, a scale value to multiply the values of x, and the power used for variable x.
- The parameter **lstSelYData** is a list that specifies data for variable y. The list contains an integer that selects the **DSMat** column storing values for variable y, and a value to shift the values of y, a scale value to multiply the values of y, and the power used for variable y.

Both parameters **lstSelXData** and **lstSelYData** are lists that have similar contents. The first list element is a column index that selects a variable in the **DSMat** matrix. The second and third list elements are the shift and scale values, respectively, used with each of the x and y values. The shift and scale transformations use the following equation:

Intermdiate_transformed_variable = scale_value * variable + shift_value

The last list element is the power value used with the above (intermediate) transformed values. The argument for this power value can be a negative number, zero, or a positive number. You can use integers and non-integer powers. When the argument is zero, the function **LinearizedReg** treats this argument as a special case and applies the natural logarithm to the result of the scaled and shifted values. When the argument is not zero, the function applies that argument as the power of the scaled and shifted values. Thus, the final transformation for each variable is:

Final_transformed_variable = (scale_value * variable + shift_value)^Power

When the value of Power is not zero. Otherwise, the final transformation is:

Final_transformed_variable = ln(scale_value * variable + shift_value)

This transformation scheme allows you to cover a wide range of transformations without taxing the size of the source code. For example you can generate linear, squared, cubed, square-root, cube root transformations and all of their reciprocals! If the transformations generate an error, the function execution will stop. Experience shows that typical values for the scale and shift are 1 and 0, respectively. However, the ability to shift and scale the original observed values, using values other than 1 and 0, may well prove to be valuable when you need them!

Let me give you a few examples for the list for parameter **lstSelXData**. You can use a similar approach with the **lstSelYData** parameter. If you pass the argument {2,0,1,1} to parameter **lstSelXData**, you tell function **LinearizedReg** that the source values for variable x are in the second column of the data source matrix **DSMat**. You also tell the function that variable x should be transformed using:

x = (1*x + 0)^1

Which basically tells the function **LinearizedReg** to use the values of variable x as they appear in the source data matrix. If you pass the argument {3,1,1.1,0.5} you tell function **LinearizedReg** that the source values for variable x are in the third column of the data source matrix **DSMat**. You also tell the function that variable x should be transformed using:

x = (1.1*x + 1)^0.5

If you pass the argument {3,2,3,–1} you tell function **LinearizedReg** that the source values for variable x are in the third column of the data source matrix **DSMat**. You also tell the function that variable x should be transformed using:

x = (3*x + 2)^(–1) = 1/(3*x+2)

The function **LinearizedReg** returns a list that contains the value of the coefficient of determination and a column matrix containing the regression coefficients. I recommend that you store the results of calling function **LinearizedReg** in a list so that you can further examine and/or use the results.

Statement

```
EXPORT LinearizedReg(DSMat,lstSelXData,lstSelYData)
BEGIN
  LOCAL i,x,y;
  LOCAL lstDimX,NumRowsX;
  LOCAL MatX,VectY,RegCoeff;
  LOCAL SelXCol,ShiftX,ScaleX,PowerX;
  LOCAL SelYCol,ShiftY,ScaleY,PowerY;
  LOCAL Sum1,Sum2,YMean,Yhat,Rsqr;

  lstDimX:=SIZE(DSMat);
  NumRowsX:=lstDimX(1);

  // get parameters for x
  SelXCol:=lstSelXData(1);
  ShiftX:=lstSelXData(2);
  ScaleX:=lstSelXData(3);
  PowerX:=lstSelXData(4);

  // get parameters for y
  SelYCol:=lstSelYData(1);
  ShiftY:=lstSelYData(2);
  ScaleY:=lstSelYData(3);
  PowerY:=lstSelYData(4);

  // create regression matrix and vector
  MatX:=MAKEMAT(1,NumRowsX,2);
  VectY:=MAKEMAT(1,NumRowsX,1);
  // populate matrix and vector
  FOR i FROM 1 TO NumRowsX DO
    x:=ScaleX*DSMat(i,SelXCol)+ShiftX;
    y:=ScaleY*DSMat(i,SelYCol)+ShiftY;

    // transform x
    IF PowerX==0 THEN
      x:=LN(x);
    ELSE
      x:=x^PowerX;
    END;
    // transform y
    IF PowerY==0 THEN
      y:=LN(y);
    ELSE
      y:=y^PowerY;
    END;

    MatX(i,2):=x;
    VectY(i,1):=y;
  END;
  // calculate coefficient of determination
  RegCoeff:=LSQ(MatX,VectY);

  // calculate ymean
  Sum1:=0;
  FOR i FROM 1 TO NumRowsX DO
    y:=VectY(i,1);
    Sum1:=Sum1+y;
  END;
  YMean:=Sum1/NumRowsX;

  // calculate coefficient of determination
  Sum1:=0;
  Sum2:=0;
  Yhat:=MatX*RegCoeff;
  FOR i FROM 1 TO NumRowsX DO
    Sum1:=Sum1+(Yhat(i,1)-YMean)^2;
    Sum2:=Sum2+(VectY(i,1)-YMean)^2;
  END;
  Rsqr:=Sum1/Sum2;
  RETURN {Rsqr,RegCoeff};
END;
```

Table 1 – The function LinearizedReg.

Let's use the function **LinearizedReg** to fit the data in Table 2. Enter the data in matrix M1.

x	*y*
1	2
2	14
3	54
4	120
5	237
6	440
7	890
8	1000
9	1500
10	2000

Table 2 – Sample data for testing function LinearizedReg.

The nonlinear regression model I want to use is:

$y = a*(2*x+1)^b$

The linearized form of the above equation is:

ln(y) = ln(a) + b*ln(2*x+1)

Thus ln(y) and ln(2*x+1) have a linear relation. To obtain the regression coefficients and coefficient of determination for data in Table 2, execute the following command:

```
LinearizedReg(M1,{1,1,2,0},{2,0,1,0})→L1
```

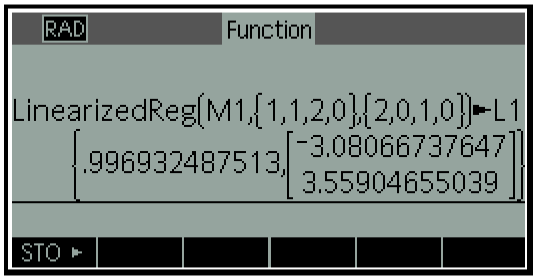
The first argument of calling function **LinearizedReg** is the matrix M1 which contains the (x, y) data points. The second argument is the list {1,1,2,0}. This list tells the function **LinearizedReg** that values for variable x are in the first column of matrix M1. The remaining list elements tell the function that the transformed values for variable x are calculated using:

x = ln(2*x+1)

The third argument is the list {2, 0,1,0}. This list tells the function **LinearizedReg** that values for variable y are in the second column of matrix M1. The remaining list elements tell the function that the transformed values for variable y are calculated using:

y = ln(1*y + 0) = ln(y)

Figure 1 shows the output of calling function **LinearizedReg**.



```
LinearizedReg(M1,{1,1,2,0},{2,0,1,0})►L1
      {.996932487513,[-3.08066737647]
                     [3.55904655039 ]}
```

Fig. 1 – The results of executing function LinearizedReg.

The results show that R^2 is 0.996932488 and the fitted polynomial is:

ln(y) = –3.080667377 + 3.55904655 ln(2*x+1)

The value of R^2 indicates that the polynomial explains about 99.7% of the variation in the values of y. The values of the regression coefficients a and b are 0.045928595 and 3.55904655, respectively.

Linearized Regression for Three Variables

This section offers the function **LinerizedReg3** that performs linearized regression between two independent variables, x and z, and the dependent variable y. Function **LinerizedReg3** is very similar to function **LinearizedReg** and has the following parameters:

- The parameter **DSMat** represents the matrix that has the x, z, and y data.
- The parameter **lstSelXData** is a list that specifies data for variable x. The list contains an integer

that selects the **DSMat** column storing values for variable x, a value to shift the values of x, a scale value to multiply the values of x, and the power used for variable x.

- The parameter **lstSelZData** is a list that specifies data for variable z. The list contains an integer that selects the **DSMat** column storing values for variable z, a value to shift the values of z, a scale value to multiply the values of z, and the power used for variable z.
- The parameter **lstSelYData** is a list that specifies data for variable y. The list contains an integer that selects the **DSMat** column storing values for variable y, a value to shift the values of y, a scale value to multiply the values of y, and the power used for variable y.

You can regard function **LinerizedReg3** as the big brother of function **LinearizedReg**. Table 3 shows the source code for function **LinearizedReg3**.

Statement
EXPORT LinearizedReg3(DSMat,lstSelXData,lstSelZData,lstSelYData)
BEGIN
LOCAL i,x,y,z;
LOCAL lstDimX,NumRowsX;
LOCAL MatX,VectY,RegCoeff;
LOCAL SelXCol,ShiftX,ScaleX,PowerX;
LOCAL SelYCol,ShiftY,ScaleY,PowerY;
LOCAL SelZCol,ShiftZ,ScaleZ,PowerZ;
LOCAL Sum1,Sum2,YMean,Yhat,Rsqr;
lstDimX:=SIZE(DSMat);
NumRowsX:=lstDimX(1);
SelXCol:=lstSelXData(1);
ShiftX:=lstSelXData(2);
ScaleX:=lstSelXData(3);
PowerX:=lstSelXData(4);
// get parameters for z
SelZCol:=lstSelZData(1);
ShiftZ:=lstSelZData(2);
ScaleZ:=lstSelZData(3);
PowerZ:=lstSelZData(4);
// get parameters for y
SelYCol:=lstSelYData(1);
ShiftY:=lstSelYData(2);
ScaleY:=lstSelYData(3);
PowerY:=lstSelYData(4);
// create regression matrix and vector
MatX:=MAKEMAT(1,NumRowsX,3);
VectY:=MAKEMAT(1,NumRowsX,1);
// populate matrix and vector
FOR i FROM 1 TO NumRowsX DO

Statement
x:=ScaleX*DSMat(i,SelXCol)+ShiftX;
y:=ScaleY*DSMat(i,SelYCol)+ShiftY;
z:=ScaleZ*DSMat(i,SelZCol)+ShiftZ;
// transform x
IF PowerX==0 THEN
x:=LN(x);
ELSE
x:=x^PowerX;
END;
// transform z
IF PowerZ==0 THEN
z:=LN(z);
ELSE
z:=z^PowerZ;
END;
// transform y
IF PowerY==0 THEN
y:=LN(y);
ELSE
y:=y^PowerY;
END;
MatX(i,2):=x;
MatX(i,3):=z;
VectY(i,1):=y;
END;
// calculate regression coefficients
RegCoeff:=LSQ(MatX,VectY);
// calculate ymean
Sum1:=0;
FOR i FROM 1 TO NumRowsX DO
y:=VectY(i,1);
Sum1:=Sum1+y;
END;
YMean:=Sum1/NumRowsX;
// calculate coefficient of determination
Sum1:=0;
Sum2:=0;
Yhat:=MatX*RegCoeff;
FOR i FROM 1 TO NumRowsX DO

Statement
`    Sum1:=Sum1+(Yhat(i,1)-YMean)^2;`
`    Sum2:=Sum2+(VectY(i,1)-YMean)^2;`
`  END;`
`  Rsqr:=Sum1/Sum2;`
`  RETURN {Rsqr,RegCoeff};`
`END;`

Table 3 – The function LinearizedReg3.

Let's use the function **LinearizedReg3** to fit the data in Table 4. Enter the data in matrix M1.

x	*z*	*y*
1	1	1.6
2	1	1.2
3	2	–0.7
4	2	–1.9
5	3	–1.9
6	3	–3.4
7	4	–3.0
8	4	–4.0
9	5	–2.8
10	5	–3.5

Table 4 – Sample data for testing function LinearizedReg3.

The regression model I want to use is:

y = a + b*ln(1.5*x+2) + c/(2*z – 1)

Thus ln(y) has a linear relation with ln(1.5*x+2) and 1/(2*z – 1). To obtain the regression coefficients and coefficient of determination for data in Table 1, execute the following command:

```
LinearizedReg3(M1,{1,2,1.5,0},{2,–1,2,–1},{3,0,1,1})→L1
```

The first argument of calling function **LinearizedReg3** is the matrix M1 which contains the (x, z, y) data points. The second argument is the list {1,2,1.5,0}. This list tells the function LinearizedReg3 that values for variable x are in the first column of matrix M1. The remaining list elements tell the function that the transformed values for variable x are calculated using:

x = ln(1.5*x+2)

The third argument is the list {2, –1,2, –1}. This list tells the function **LinearizedReg3** that values for variable z are in the second column of matrix M1. The remaining list elements tell the function that the transformed values for variable z are calculated using:

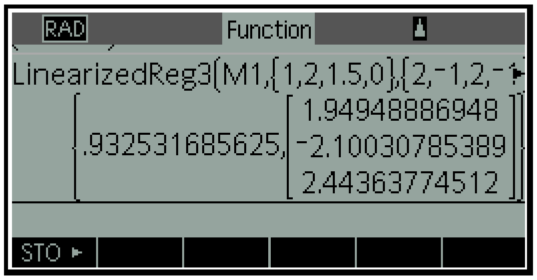
z = 1/(2*z – 1)

The fourth argument is the list {3,0,1,1}. This list tells the function LinearizedReg3 that values for variable y are in the third column of matrix M1. The remaining list elements tell the function that the transformed values for variable:

y = 1*y + 0

That is, to take the values of y from the third column of matrix M1.

Figure 2 shows the output of calling function **LinearizedReg3**.



```
LinearizedReg3(M1,{1,2,1.5,0},{2,-1,2,-
               [ 1.94948886948 ]
{.932531685625,[-2.10030785389 ]}
               [ 2.44363774512 ]
```

Fig. 2 – The results of executing function LinearizedReg.

The results show that R^2 is 0.932531686 and the fitted polynomial is:

$y = 1.949488869 – 2.100307854\ \ln(1.5*x+2) + 2.443637745 /(2*z – 1)$

The value of R^2 indicates that the polynomial explains about 93 % of the variation in the values of y.

Regression Beyond Three Variables

What about fitting regression models beyond a total of three variables? I could present the function, **LinearizedReg4** to support regression with four variables. It would look like a version of functions **LinearizedReg** and **LinearizedReg3** on steroids!

I decided to take a different approach and present you with a more powerful function. This function can handle elaborate regression models such as:

$y = a_0 + a_1\ (2x+4)^2 + a_2\ (4z+3)^3 + a_3 \ln(t+1) + a_4\ (u+3)^{0.5} + a_5\ (v+2)$
$y = a_0 + a_1\ (2x+4)^2 + a_2\ (3x+3)/(4x+3)^3$
$y = a_0 + a_1\ (2x+4)^2 + a_2\ (4z+3)^3 + a_3 \ln(t+1)/(3z+7) + a_4\ (x+1)(3z-5)/(t=7)$

The above examples hint at the following features of the next function:

- The ability to handle more than three independent variables.
- The ability to build regression models with terms that have multiplicative factors. The key word here is *multiplicative*. These factors have transformations of the same or different variables.

Table 5 contains the source code for function **MLRX**. This powerful and versatile function has the following parameters:

- The parameter **DSMat** specifies the data source matrix containing the dependent and independent variables.
- The parameter **MaxTerms** designates the number of terms in the regression models (not counting the constant term).
- The parameter **TrnfMat** specifies the name of the transformations matrix that contains the values used to select, transform, and store variables in the internal regression matrix **X** (stored in variable **MatX**) and vector **y** (stored in variable **VectY**).

The parameter **TrnfMat** is a matrix with six columns. Each row groups the data to select, transform, and store a specific regression variable. The columns of the transformations matrix are:

- Column 1 is the index that selects a regression variable from matrix **DSMat**. You can select a dependent variable or an independent variable.
- Column 2 specifies the scale value used to multiply the values of the selected variable.
- Column 3 specifies the shift value used to add to the scaled values of the variable.
- Column 4 specifies the power value used to raise the scaled and shifted values. If the value in this column is zero, the function **MLRX** calculates the natural logarithm of the scaled and shifted values.
- Column 5 is a numeric switch that tells function **MLRX** where to store the results of the transformed values. When the value of this column is positive, the function stores the transformed values in variable **MatX**. Otherwise, it stores the transformed values in variable **VectY**.
- Column 6 specifies the column index of variable **MatX** where the transformed values are stored. The values of this column are relevant only when the corresponding values in Column 5 are positive.

You can think of parameter **TrnfMat** as numerically-coded meta-program (or instruction set) for function **MLRX**. This meta-program tells the function which variable to select, what transformations to apply, and where to store the transformation results. Each row in parameter **TrnfMat** represents a meta-program instruction. The totality of these instructions helps the function **MLRX** to build the data in the variables **MatX** and **VectY**.

The function **MLX** returns the coefficient of determination and the vector matrix of the regression coefficients. If a value in column 6 of the transformations matrix is greater than the maximum number of terms, plus 1, the function **MLRX** displays an error message box and returns the text “ERROR”.

Statement
`EXPORT MLRX(DSMat,MaxTerms,TrnfMat)`
`BEGIN`
`  LOCAL i,j,x,Rsqr;`
`  LOCAL lstDimX,lstDimT,NumRowsX,NumColsX;`
`  LOCAL MatX,VectY,RegCoeff;`
`  LOCAL SelXCol,Shift,Scale,PowerX;`
`  LOCAL InsMat,InsCol,NumTrnf;`
`  LOCAL YMean,Sum1,Sum2,Yhat;`
`  // calculate the number of data points`
`  lstDimX:=SIZE(DSMat);`
`  NumRowsX:=lstDimX(1);`
`  NumColsX:=lstDimX(2);`
`  // get the number of transformations`
`  lstDimT:=SIZE(TrnfMat);`
`  NumTrnf:=lstDimT(1);`
`  // create the data matrices`
`  MatX:=MAKEMAT(1,NumRowsX,MaxTerms+1);`
`  VectY:=MAKEMAT(1,NumRowsX,1);`

Statement

```
  FOR j FROM 1 TO NumTrnf DO
    // get the transformation/insertion parameters
    SelXCol:=TrnfMat[j,1];
    Scale:=TrnfMat[j,2];
    Shift:=TrnfMat[j,3];
    PowerX:=TrnfMat[j,4];
    InsMat:=TrnfMat[j,5];
    InsCol:=TrnfMat[j,6];

    // process all rows for current variable selection,
    // transformation, and insertion
    FOR i FROM 1 TO NumRowsX DO
      // get x
      x:=DSMat[i,SelXCol];
      // transform x by scaling and shifting
      x:=Scale*x+Shift;
      // raise x to power or take ln() value
      IF PowerX==0 THEN
        x:=LN(x);
      ELSE
        x:=x^PowerX;
      END;

      // insert in targeted matrix
      IF InsMat>0 THEN
        // insert in matrix of independent variables
        IF InsCol>(MaxTerms+1) THEN
          // display an error message
          MSGBOX("Column "+InsCol+" is outside the range of columns");
          RETURN "ERROR";
        END;
        MatX[i,InsCol]:=MatX[i,InsCol]*x;
      ELSE
        // insert in vector of dependent variable
        VectY[i,1]:=VectY[i,1]*x;
      END;
    END;
  END;

  // calculate the regression coefficients
  RegCoeff:=LSQ(MatX,VectY);

  // calculate ymean
  Sum1:=0;
  FOR i FROM 1 TO NumRowsX DO
```

Statement
`    Sum1:=Sum1+VectY(i,1);`
`  END;`
`  YMean:=Sum1/NumRowsX;`
`  // calculate the correlation coefficient`
`  Sum1:=0;`
`  Sum2:=0;`
`  Yhat:=MatX*RegCoeff;`
`  FOR i FROM 1 TO NumRowsX DO`
`    Sum1:=Sum1+(Yhat(i,1)-YMean)^2;`
`    Sum2:=Sum2+(VectY(i,1)-YMean)^2;`
`  END;`
`  Rsqr:=Sum1/Sum2;`
`  // return the results`
`  RETURN {Rsqr,RegCoeff};`
`END;`

Table 5 – The source code for function MLRX.

Let's use function **MLRX** to fit data for with the regression model:

$\ln(y) = a + b/(2x + 1) + c \ln(3z + 5) + d (5t - 2)^2$

Table 6 shows the data that I will use to calculate the regression coefficients for the above equation. Store the data of Table 6 in matrix M1. Table 7 shows the transformations matrix. The matrix rows give function **MLRX** the instructions to build the data in the variables **MatX** and **VectY**. Store the data of Table 7 in matrix M2.

x	*z*	*t*	*y*
1	1	7	3000
2	3	6	250
3	2	3	500
4	2	1	50
5	3	2	200
6	3	5	1500
7	4	8	4500
8	4	9	5500
9	5	4	1000
10	5	2	200

Table 6 – Sample data for testing function MLRX.

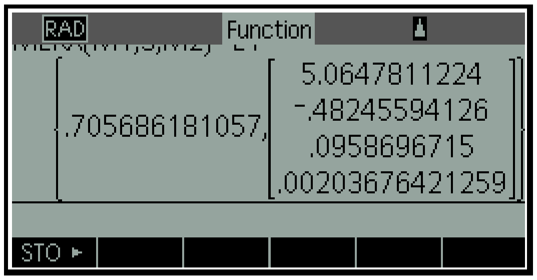
Select Variable	*Scale Value*	*Shift Value*	*Power*	*Target Matrix*	*Target Matrix Column*
4	1	0	0	0	0
1	2	1	–1	1	2
2	3	5	0	1	3
3	5	–2	2	1	4

Table 7 – Transformations matrix.

Calculate the coefficient of determination and regression coefficients by executing the following command:

```
MLRX(M1,3,M2)→L1
```

Figure 3 shows the results of calling function **MLRX**.



```
RAD   Function
MLRX(M1,3,M2)▸L1
 [             [ 5.0647811224    ]]
 [.705686181057,[-.48245594126   ]]
 [             [ .0958696715     ]]
 [             [.00203676421259  ]]
STO ▸
```

Fig. 3 – The results of executing functions InsertVar and MLR2.

The coefficient of determination is 0.705686. The fitted model is:

$\ln(y) = 5.064781119 - 0.482455938/(2x + 1) + 0.095869673 \ln(3*z + 5) + 0.002036764 (5t - 2)^2$

The above model explains about 70.5 % of the variation in y.

Further Exploring the Power of Function MLRX

The last section showed you how to work with a custom multi-variable linearized regression. The example I gave you tested fitting the following empirical equation:

$\ln(y) = a + b/(2x + 1) + c \ln(3z + 5) + d (5t - 2)^2$

The above equation has several terms, each with a single variable. You may recall that the introduction for function **MLRX** hailed its ability to handle elaborate regression models. This section looks at this feature. Consider the following empirical equation:

$\ln(y) = a + b (5x - 2)/(2x + 1) + c \ln(3z + 5)/(x+1) + d (x+1)(z+3)(5t - 2)^2$

Notice the following terms of the above equation:

- The term $(5x - 2)/(2x + 1)$ has two factors that use the same variable, x.
- The term $(\ln(3z + 5))/(x+1)$ has two factors that use two different variables, z and x.
- The term $(x+1)(z+3)(5t - 2)^2$ has three factors that use three different variables, x, z and t.

To use function **MLRX** with the above empirical regression model we use the values in Table 8. The rows of that table map the terms of the empirical regression model from left to right. Store the values of

Select Variable	*Scale Value*	*Shift Value*	*Power*	*Target Matrix*	*Target Matrix Column*
4	1	0	0	0	0
1	5	–2	1	1	2
1	2	1	–1	1	2
2	3	5	0	1	3
1	1	1	–1	1	3
1	1	1	1	1	4
2	1	3	1	1	4
3	5	–2	2	1	4

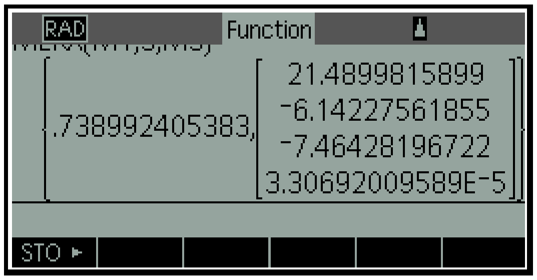
Table 8 – The second transformations matrix.

Table 8 in matrix M3. I will use the same data in Table 6, which should still reside in matrix M1.

Calculate the coefficient of determination and regression coefficients by executing the following command:

```
MLRX(M1,3,M3)
```

Figure 4 shows the output of function **MLRX**.



```
RAD   Function
MLRX(M1,3,M3)
               [ 21.4899815899        ]
[.738992405383,  -6.14227561855       ]
               [ -7.46428196722       ]
               [ 3.30692009589E-5     ]
STO ▸
```

Fig. 4 – The results of executing function MLRX.

The coefficient of determination is 0.7399. The fitted model is:

ln(y) = 21.49 – 6.1423 (5x – 2)/(2x + 1) – 7.464 ln(3z + 5)/(x+1) + 3.3069E–5 (x+1)(z+3)$(5t – 2)^2$

The above model explains about 74 % of the variation in y.

The versatility of function **MLRX** comes from the following **IF** statement in Table 5:

```
IF InsMat>0 THEN
  // insert in matrix of independent variables
  IF InsCol>(MaxTerms+1) THEN
    // display an error message
    MSGBOX("Column "+InsCol+" is outside the range of columns");
    RETURN "ERROR";
  END;
  MatX[i,InsCol]:=MatX[i,InsCol]*x;
ELSE
  // insert in vector of dependent variable
  VectY[i,1]:=VectY[i,1]*x;
END;
```

The assignment statements that write values to matrices **MatX** and **VectY**empower you to build the product of multiple factors containing the same or different variable. You can even include values from the dependent variable, if your model requires it.

Observations and Conclusions

This article presented you with HP 39gII functions that performed linearized regression between two and between three variables. In addition, the article presented you with the special function **MLRX** that allows you to perform linearized regression between four or more variables. The function also supports regression models that contains terms with one or more transformed variables. All of these tools allow you to select variables and then temporarily scale, shift, and raise their values to powers (or take their natural logarithms) before performing regression calculations.

The next article presents HP 39gII functions that perform the best linearized regression between two, three, and four variables. The article also offers an HP 39gII function that selects the best polynomial that fits (x, y) data points.

References

1. Wikipedia article *Coefficient of Determination*.
2. Wikipedia article *Linear Regression*.
3. Draper and Smith, *Applied Regression Analysis*, Wiley-Interscience; 3rd edition (April 23, 1998)
4. Neter, Kuther, Wasserman, and Nachtsheim, *Applied Linear Statistical Models*, McGraw-Hill/Irwin; 4th edition (February 1, 1996).
5. Fox, *Applied Regression Analysis and Generalized Linear Models*, Sage Publications, Inc; 2nd edition (April 16, 2008).
6. Montgomery, Peck, and Vining, *Introduction to Linear Regression Analysis*, Wiley-Interscience; 4th edition (2006).
7. Seber and Lee, *Linear Regression Analysis*, Wiley; 2nd edition (February 5, 2003).

About the Author

Namir Shammas is a native of Baghdad, Iraq. He resides in Richmond, Virginia, USA. Namir graduated with a degree in Chemical Engineering. He received a master degree in Chemical engineering from the University of Michigan, Ann Arbor. He worked for a few years in the field of water treatment before focusing for 17 years on writing programming books and articles. Later he worked in corporate technical documentation. He is a big fan of HP calculators and collects many vintage models. His hobbies also include traveling, music, movies (especially French movies), chemistry, cosmology, Jungian psychology, mythology, statistics, and math. As a former PPC and CHHU member, Namir enjoys attending the HHC conferences. *Email me at: nshammas@aol.com*

HP 39gII Regression: Part III

HP Solve #30 page 50

HP 39gII Regression: Part III
The Best Curve Fits in Town!

Namir Shammas

Introduction

In the last two articles I presented various HP 39gII functions that performed regression calculations on different models. Using these functions assumed that you either knew the regression model suitable for your data or were willing to try a few different regression models to see which one best fit your data. This article looks at HP 39gII functions that systematically try a large number of regression models to see which ones are the best. In this article I discuss the following topics:

1. The best regression models for two variables, fitting up to 81 regression models.

2. The best regression models for three variables, fitting up to 729 regression models.

3. The best regression models for four variables fitting up to 6561 regression models.

4. The best polynomial for two variables.

☞ In this series of articles, I use the term *regression model* to mean the equation that is used in the regression calculations to describe the relationship between a dependent variable and one or more independent variables.

The Best Regression Model for Two Variables

Given two variables, x and y, what is the best linearized regression model when each variable has a list of powers used to transform it? Here is a partial list of linearized regression models that can be tested:

y = a + b x
y = a + b/ln(x)
ln(y) = a + b x
ln(y) = a + b ln(x)
y = a + b/x
1/y = a + b x
1/y = a + b/x
ln(y) = a + b/x
1/y = a + b ln(x)

And so on! I can further extend the list by adding more regression models that use combinations of square root, squared, cubed, reciprocal square root, reciprocal square, and reciprocal cube transformations for each of x and y. The total number of regression models equals the number of product of the transformation applied to x and y.

What is my end game here? Rather than selecting just the very best regression model (with the highest coefficient of determination), I want to apply a more informative approach. I want to find out the best N regression models. In this article I choose N to be 20. You can easily change that limit to 10, 15, 25, or any other number you want. In general it's a good idea to look at the top N regression model and not be fixated with just the very best regression model.

Table 1 shows you the source code for function **BestLR**. This function has the following parameters:

1. The parameter **Data** is the source data matrix that contains the variables x and y. The matrix must have at least two columns of data.

2. The parameter **lstSel** is a list containing two elements. The first element is the index of parameter **Data** that selects the variable x. The second list element is the index of parameter **Data** that selects the variable y.

3. The parameter **lstX** is a list that enumerates the set of powers used to transform the variable x. These powers can be integers and non-integers, and also positive, zero, and negative. The function treats the zero power as a special case and applies the natural logarithm. If you supply an empty list to this parameter, the function automatically uses the list {–3, –2, –1, –0.5,0,0.5,1,2,3}. This list allows the values of x to be transformed into reciprocal cube, reciprocal square, reciprocal, reciprocal square root, natural logarithm, square root, linear, square, and cube values. You can pass arguments for this parameter that are subsets of these values to choose fewer transformations. You can also pass arguments for this parameter that are supersets of these values to choose more transformations. You can also pick and choose any valid combination of powers. The key point in all these cases is to use valid powers that lead to error-free transformations. All of the transformation you choose must NOT GENERATE RUNTIME errors. Otherwise, the function will stop executing.

4. The parameter **lstY** is a list that enumerates the set of powers used to transform the variable y. It works just like parameter **lstX** but on the data for variable y.

The function uses the variable **MaxRes** to manage the number of best regression models to report back to you. The function assigns the value of 20 to this variable. You can change this value to alter the number of best regression models stored in the results matrix.

The function returns a results matrix containing the best 20 results. These results are sorted by the coefficient of determination of each regression model. The results matrix contains the following columns:

1. The values of the coefficient of determination.

2. The powers used to transform variable y.

3. The powers used to transform variable x.

4. The intercept values.

5. The slope values.

Statement
`EXPORT BestLR(Data,lstSel,lstX,lstY)`
`BEGIN`
`  LOCAL Tx,Ty,i,j,k;`
`  LOCAL SelXCol,SelYCol;`
`  LOCAL PowerX,PowerY;`
`  LOCAL MatX,VectY,RegCoeff,MatRes;`
`  LOCAL lstDim,NumRows,ResUpdated;`
`  LOCAL Sum1,Sum2,Rsqr,YMean,Yhat;`
`  LOCAL MaxRes,NumColRes;`
`  // use the default transformation list`

Statement

```
  // for variable x if lstX is empty
  IF SIZE(lstX)==0 THEN
    lstX:={-3,-2,-1,-0.5,0,0.5,1,2,3};
  END;

  // use the default transformation list
  // for variable y if lstY is empty
  IF SIZE(lstY)==0 THEN
    lstY:={-3,-2,-1,-0.5,0,0.5,1,2,3};
  END;
  // set the maximum number of results
  MaxRes:=20;
  // set the number of columns in
  // the results matrix
  NumColRes:=5;
  // get the indices for variable x and y
  SelXCol:=lstSel(1);
  SelYCol:=lstSel(2);
  // get the number of rows of data
  lstDim:=SIZE(Data);
  NumRows:=lstDim(1);
  // create the result and regression matrices
  MatX:=MAKEMAT(1,NumRows,2);
  VectY:=MAKEMAT(1,NumRows,1);
  MatRes:=MAKEMAT(0,MaxRes,NumColRes);

  // iterate for each transformation in x
  FOR Tx FROM 1 TO SIZE(lstX) DO
    // get the current power of x
    PowerX:=lstX(Tx);

    // transform x
    IF PowerX==0 THEN
      FOR i FROM 1 TO NumRows DO
        MatX(i,2):=LN(Data(i,SelXCol));
      END;
    ELSE
      FOR i FROM 1 TO NumRows DO
        MatX(i,2):=Data(i,SelXCol)^PowerX;
      END;
    END;

    // iterate for each transformation in y
    FOR Ty FROM 1 TO SIZE(lstY) DO
      // get the current power of y
      PowerY:=lstY(Ty);
```

Statement

```
    // transform y
    IF PowerY==0 THEN
      FOR i FROM 1 TO NumRows DO
        VectY(i,1):=LN(Data(i,SelYCol));
      END;
    ELSE
      FOR i FROM 1 TO NumRows DO
        VectY(i,1):=Data(i,SelYCol)^PowerY;
      END;
    END;

    // calculate regression coefficients
    RegCoeff:=LSQ(MatX,VectY);

    // calculate ymean
    Sum1:= 0;
    FOR i FROM 1 TO NumRows DO
      Sum1:=Sum1+VectY(i,1);
    END;
    YMean:=Sum1/NumRows;

    // calculate coefficient of determination
    Sum1:=0;
    Sum2:=0;
    Yhat:=MatX*RegCoeff;
    FOR i FROM 1 TO NumRows DO
      Sum1:=Sum1+(Yhat(i,1)-YMean)^2;
      Sum2:=Sum2+(VectY(i,1)-YMean)^2;
    END;
    Rsqr:=Sum1/Sum2;

    // Rsqr is better than last entry
    // in the results matrix?
    IF Rsqr>MatRes(MaxRes,1) THEN
      ResUpdated:=0;
      // loop for each row in the
      // results matrix
      FOR i FROM 1 TO MaxRes DO
        // compare with other Rsqr values
        IF ResUpdated==0 AND Rsqr>MatRes(i,1) THEN
          // found better Rsqr for row i
          // push rows below?
          IF i<MaxRes THEN
            j:=MaxRes-1;
            REPEAT
            // FOR j FROM MaxRes-1 TO n STEP -1 DO
```

Statement
`                FOR k FROM 1 TO NumColRes DO`
`                  MatRes(j+1,k):=MatRes(j,k);`
`                END;`
`                j:=j-1;`
`              //END;`
`              UNTIL j<i;`
`            END;`
`            // store new best results`
`            MatRes(i,1):=Rsqr;`
`            MatRes(i,2):=PowerY;`
`            MatRes(i,3):=PowerX;`
`            MatRes(i,4):=RegCoeff(1,1);`
`            MatRes(i,5):=RegCoeff(2,1);`
`            ResUpdated:=1;`
`          END;`
`        END;`
`      END;`
`     END; // FOR Ty`
`  END; // FOR Tx`
`  RETURN MatRes;`
`END;`

Table 1 – The source code for function BestLR.

The source code for function **BestLR** uses two nested **FOR** loops to go through each transformation for the variables x and y. Inside the nested loops, the function calculates the regression coefficients and the coefficient of determination. The next phase compares the newly calculated coefficient of determination with similar values stored in the first column of the results matrix. If the newly calculated coefficient of determination is better than any value in the first column of the results matrix, the function inserts the data from the newly calculated regression into the results matrix. The function determines which row will store the new data. Then, the insertion process copies old data from the insertion row downward in the results matrix. The last row in the results matrix has its data overwritten by the ones in the row above it, or possibly by the newly calculated regression data.

Let's test the function **BestLR** using the data in Table 2. The values in the table come from the equation:

$y = 3 + 2\,x^2$

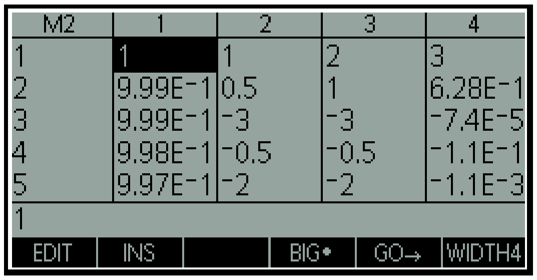
x	*y*
1	5
2	11
3	21
4	35
5	53
6	75
7	101
8	131
9	165
10	203

Table 2 – Sample data used to test the function BestLR.

Enter the values of Table 2 in the global matrix M1 and then execute the following command:

```
BestLR(M1,{1,2},{},{})→M2
```

The above call for function **BestLR** uses the default transformations for both variables x and y. Since the function uses 9 transformations (including the linear one) on each variable, the total number of regression models tested is 81. Figure 1 shows the contents of matrix M2 which stores the results of the best model selection. Table 3 shows the best five regression models. The best model in that table is indeed the one used to create the data in Table 2. Keep in mind that if the values of y included an error component, the function **BestLR** may not specify the equation $y = 3 + 2\ x^2$ as the best model. That model may be superseded by other models depending on the error components associated with the values of variable y.



M2	1	2	3	4
1	1	1	2	3
2	9.99E-1	0.5	1	6.28E-1
3	9.99E-1	-3	-3	-7.4E-5
4	9.98E-1	-0.5	-0.5	-1.1E-1
5	9.97E-1	-2	-2	-1.1E-3

1

Fig. 1 – The values in matrix M2.

Rsqr	***Power y***	***Power x***	***Regression Model***	***Intercept***	***Slope***
1.000000	1	2	$y = a+b\ x^2$	3	2
0.999189	0.5	1	$\sqrt{y} = a+b\ x$	0.6279425	1.3509222
0.998873	-3	-3	$1/y^3 = a + b/x^3$	-7.441533E-5	8.0478988E-3
0.997834	-0.5	0.5	$1/\sqrt{y} = a+b/\sqrt{x}$	-0.1115322	0.56543287
0.997208	-2	-2	$1/y^2 = a + b/x^2$	-1.1075566E-3	0.0407642

Table 3 – The Best five regression models found by function BestLR.

The Best Regression Model for Three Variables

The last section gave you have a taste of finding the best regression model between one dependent variable and one independent variable. This section presents a function that finds the best regression model between the dependent variable, y, and the independent variables x and z.

Table 4 shows you the source code for function **BestMLR2**. This function has the following parameters:

1. The parameter **Data** is the source data matrix that contains the variables x, z, and y. The matrix must have at least two columns of data.

2. The parameter **lstSel** is a list containing three elements. The first element is the index of parameter **Data** that selects the independent variable x. The second list element is the index of parameter Data that selects the independent variable z. The third list element is the index of parameter Data that selects the dependent variable y.

3. The parameter **lstX** is a list that enumerates the set of powers used to transform the variable x. These powers can be integers and non-integers, and also positive, zero, and negative. The function treats the

zero power as a special case and applies the natural logarithm. If you supply an empty list to this parameter, the function automatically uses the list {–3, –2, –1, –0.5,0,0.5,1,2,3}. This list allows the values of x to be transformed into reciprocal cube, reciprocal square, reciprocal, reciprocal square root, natural logarithm, square root, linear, square, and cube values. You can pass arguments for this parameter that are subsets of these values to choose fewer transformations. You can also pass arguments for this parameter that are supersets of these values to choose more transformations. All of the transformation you choose must NOT GENERATE RUNTIME errors.

4. The parameter **lstZ** is a list that enumerates the set of powers used to transform the variable z. It works just like parameter **lstX** but on the data for variable z.

5. The parameter **lstY** is a list that enumerates the set of powers used to transform the variable y. It works just like parameter **lstX** but on the data for variable y.

The function uses the variable **MaxRes** to manage the number of best regression models to store and report back to you. The function assigns the value of 20 to this variable. You can change this value to alter the number of best regression models stored in the results matrix.

The function returns a results matrix containing the best 20 results. These results are sorted by the coefficient of determination of each regression model. The results matrix contains the following columns:

1. The values for the coefficient of determination.
2. The powers used to transform variable y.
3. The powers used to transform variable x.
4. The powers used to transform variable z.
5. The intercept values.
6. The values for the regression coefficient of variable x.
7. The values for the regression coefficient of variable z.

Statement

```
EXPORT BestMLR2(Data,lstSel,lstX,lstZ,lstY)
BEGIN
  LOCAL Tx,Tz,Ty,i,j,k;
  LOCAL SelXCol,SelZCol,SelYCol;
  LOCAL PowerX,PowerZ,PowerY;
  LOCAL MatX,VectY,RegCoeff,MatRes;
  LOCAL lstDim,NumRows,ResUpdated;
  LOCAL Sum1,Sum2,Rsqr,YMean,Yhat;
  LOCAL MaxRes,NumColRes;

  // use default transformation list
  // if lstX is an empty list
  IF SIZE(lstX)==0 THEN
    lstX:={-3,-2,-1,-0.5,0,0.5,1,2,3};
  END;
```

Statement
`// use default transformation list`
`// if lstZ is an empty list`
`IF SIZE(lstZ)==0 THEN`
`  lstZ:={-3,-2,-1,-0.5,0,0.5,1,2,3};`
`END;`
`// use default transformation list`
`// if lstY is an empty list`
`IF SIZE(lstY)==0 THEN`
`  lstY:={-3,-2,-1,-0.5,0,0.5,1,2,3};`
`END;`
`// Set the number of rows and columns in the results matrix`
`MaxRes:=20;`
`NumColRes:=7;`
`// get the variable selectors`
`SelXCol:=lstSel(1);`
`SelZCol:=lstSel(2);`
`SelYCol:=lstSel(3);`
`// get the number of rows in matrix Data`
`lstDim:=SIZE(Data);`
`NumRows:=lstDim(1);`
`// create the regression and results matrices`
`MatX:=MAKEMAT(1,NumRows,3);`
`VectY:=MAKEMAT(1,NumRows,1);`
`MatRes:=MAKEMAT(0,MaxRes,NumColRes);`
`// start the calculations`
`// process transformations for variable x`
`FOR Tx FROM 1 TO SIZE(lstX) DO`
` PowerX:=lstX(Tx);`
` // transform x`
` IF PowerX==0 THEN`
`   FOR i FROM 1 TO NumRows DO`
`     MatX(i,2):=LN(Data(i,SelXCol));`
`   END;`
`  ELSE`
`    FOR i FROM 1 TO NumRows DO`
`      MatX(i,2):=Data(i,SelXCol)^PowerX;`
`    END;`
`  END;`
`  // process transformations for variable z`
`  FOR Tz FROM 1 TO SIZE(lstZ) DO`
`    PowerZ:=lstZ(Tz);`

Statement
`// transform z`
`IF PowerZ==0 THEN`
`  FOR i FROM 1 TO NumRows DO`
`    MatX(i,3):=LN(Data(i,SelZCol));`
`  END;`
`ELSE`
`  FOR i FROM 1 TO NumRows DO`
`    MatX(i,3):=Data(i,SelZCol)^PowerZ;`
`  END;`
`END;`
`// process transformations for variable y`
`FOR Ty FROM 1 TO SIZE(lstY) DO`
`  PowerY:=lstY(Ty);`
`  // transform y`
`  IF PowerY==0 THEN`
`    FOR i FROM 1 TO NumRows DO`
`      VectY(i,1):=LN(Data(i,SelYCol));`
`    END;`
`  ELSE`
`    FOR i FROM 1 TO NumRows DO`
`      VectY(i,1):=Data(i,SelYCol)^PowerY;`
`    END;`
`  END;`
`  // calculate regression coefficients`
`  RegCoeff:=LSQ(MatX,VectY);`
`  // calculate ymean`
`  Sum1:=0;`
`  FOR i FROM 1 TO NumRows DO`
`    Sum1:=Sum1+VectY(i,1);`
`  END;`
`  Ymean:=Sum1/NumRows;`
`  // calculate the coefficient of determination`
`  Sum1:=0;`
`  Sum2:=0;`
`  Yhat:=MatX*RegCoeff;`
`  FOR i FROM 1 TO NumRows DO`
`    Sum1:=Sum1+(Yhat(i,1)-Ymean)^2;`
`    Sum2:=Sum2+(VectY(i,1)-Ymean)^2;`
`  END;`
`  Rsqr:=Sum1/Sum2;`

Statement

```
    // Rsqr is better than last entry
    // in the results matrix?
    IF Rsqr>MatRes(MaxRes,1) THEN
     ResUpdated:=0;
     // check which row to insert better
     // regression results
     FOR i FROM 1 TO MaxRes DO
      // insert new results in row i?
      IF ResUpdated==0 AND Rsqr>MatRes(i,1) THEN
       // inserting inside the results matrix?
       IF i<MaxRes THEN
        // downward copy rows from MaxRes-1 to i
        j:=MaxRes-1;
        REPEAT
         FOR k FROM 1 TO NumColRes DO
          MatRes(j+1,k):=MatRes(j,k);
         END;
         j:=j-1;
        UNTIL j<i;
       END;
       // insert better results in row i
       MatRes(i,1):=Rsqr;
       MatRes(i,2):=PowerY;
       MatRes(i,3):=PowerX;
       MatRes(i,4):=PowerZ;
       FOR k FROM 1 TO 3 DO
        MatRes(i,4+k):=RegCoeff(k,1);
       END;
       ResUpdated:=1;
      END;
     END;
    END;
   END; // FOR Ty
  END; // FOR Tz
 END; // FOR Tx

 RETURN MatRes;

END;
```

Table 4 – The source code for function BestMLR2.

The source code for function **BestMLR2** uses three nested **FOR** loops to go through each transformation for the variables x, z, and y. Inside the nested loops, the function calculates the regression coefficients and the coefficient of determination. The next phase compares the newly calculated coefficient of determination with similar values stored in the first column of the results matrix. If the newly calculated coefficient of determination is better than any value in the first column of the results matrix, the function inserts the data from the newly calculated regression into the results matrix. The function determines which row will store the new data. Then, the insertion process copies old data from the insertion row downward in the results matrix. The last row in the results matrix has its data overwritten by the ones in the row above it, or possibly by the newly calculated regression data.

Let’s test the function **BestMLR2** using the data in Table 5. The values in the table come from to the equation:

$y = 3 + 2\ x^2 + 20/z$

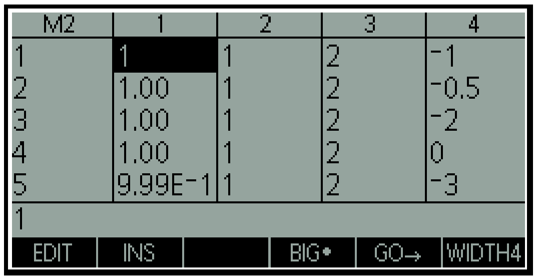
x	*z*	*y*
1	1	25
2	1	31
3	2	31
4	2	45
5	4	58
6	5	79
7	2	111
8	4	136
9	5	169
10	5	207

Table 5 – Sample data used to test the function BestMLR2.

Enter the values of Table 5 in the global matrix M1 and then execute the following command:

```
BestMLR2(M1,{1,2,3},{},{},{})→M2
```

The above call for function **BestMLR2** uses the default transformations for variables x, z, and y. Since the function uses 9 transformations (including the linear one) on each variable, the total number of regression models tested is 729. Figure 2 shows the contents of matrix M2 which stores the results of the best model selection. Table 6 shows the best five regression models. The best model is the same one used to create the data in Table 5.



M2	1	2	3	4
1	1	1	2	-1
2	1.00	1	2	-0.5
3	1.00	1	2	-2
4	1.00	1	2	0
5	9.99E-1	1	2	-3

```
1
```

Fig. 2 – The values in matrix M2.

Rsqr	*Power y*	*Power z*	*Power z*	*Intercept*	*Slope X*	*Slope Z*
1.000000	1	2	-1	3	2	20
0.999893	1	2	-0.5	-7.371050	2.00917	29.73563
0.999772	1	2	-2	8.661370	1.976927	14.78107
0.999542	1	2	0	21.432295	2.011851	-10.008505
0.999459	1	2	-3	10.607556	1.960595	12.795117

Table 6 – The Best five regression models found by function BestMLR2.

The appendix contains the listing for function **BestML3** which obtains the best regression models for the independent variables x, z, and t, and the dependent variable y. This function performs tasks that are similar to functions **BestLR** and **BetMLR2**.

The Best Polynomial Fit

If you want to fit pairs of (x, y) data points with a polynomial, one of the first and common questions you may ask regards the best order of the polynomial. This section deals with how to obtain the best polynomial order in fitting (x, y) data points. The first issue in dealing with fitting polynomials that have different orders is how to compare the goodness of fit for regression models that have a different number of terms. Up till now, the functions I presented used the coefficient of determination to compare models that have the same number of terms. To compare models that have different number of terms we need to calculate the *adjusted coefficient of determination*[2]. The following equation calculates this statistic based on the coefficient of determination:

$$R^2_{adj} = 1 - (1 - R^2)\,(n - 1) / (n - k - 1)$$

Where n is the number of data points and k is the number of independent variables that are in the regression model.

Using the adjusted coefficient of determination seems like a good idea at first. The reality is that high-order polynomials also generate high value for their adjusted coefficient of determination. Often, you get to a certain polynomial order where the gain in the adjusted coefficient of determination is small. You ask yourself if that small gain is justified!

What would be nice is find a different statistic that balances the following aspects of goodness of fit:

- Reward the regression models that generate smaller values for the sum of squared errors.
- Penalize the regression models for using more independent variables or terms.

The *Akaike information criterion*[1] (AIC) is one of the new goodness-of-fit statistics that follow the above two rules. The corrected AIC statistic, AIC_C, is a refined version that I use in this article. To calculate the AIC_C I use the following equation:

$$AIC_C = n\,ln\left(\textstyle\sum_1^n (\hat{y}_i - y_i)^2 / n\right) + 2k + (2\,k\,(k+1))/(n-k-1)$$

Where n is the number of observations, $\hat{y}_i$ is the projected value of y, y_i is the observed value of y (or its transformed value), and k is the polynomial order plus 1 (that is, the total number of regression coefficients, including the constant term). What kind of values for the AIC_C statistics are we looking for? The smallest value of AIC_C picks the best the regression model. The AIC_C statistic works better than the adjusted coefficient of determination with polynomials. The AIC_C is able to penalize higher order polynomials if they fail to significantly reduce the sum of the errors squared. The AIC_C does not justify a very small increase in the coefficient of determination in a higher order polynomial fit.

Table 7 presents the source code for the **BestPolyReg** function. This function has the following parameters:

- The parameter **Data** is the source data matrix that contains the values for variables x and y.
- The parameter **SelXCol** selects the column in matrix **Data** that contains the values for variable x.
- The parameter **SelYCol** selects the column in matrix **Data** that contains the values for variable y.

- The parameters **MinOrder** and **MaxOrder** define the range of polynomial orders to test.

The function returns the following list of results for the best polynomial fit:

- The polynomial order.
- The coefficient of determination.
- The adjusted coefficient of determination.
- The value for the AIC$_C$ statistic.
- The vector column containing the regression coefficients.

During the calculation phase, the function also displays the values for the polynomial order, sum of squared errors, and AIC$_C$ statistic for all the polynomial order. This output should give you an idea of how the different polynomial order compare with each other. When you press the [Home] button, the calculator switches to the Home display and shows you the list of final results.

Statment

```
EXPORT BestPolyReg(Data,SelXCol,SelYCol,MinOrder,MaxOrder)
BEGIN
  LOCAL i,k,x,y,Order;
  LOCAL MatX,VectY;
  LOCAL lstDim,NumRows;
  LOCAL Sum1,Sum2,Sum3,AICc,BestAICc;
  LOCAL Rsqr,RsqrAdj,YMean,Yhat,RegCoeff;
  LOCAL BestOrder,BestRsqr,BestRsqrAdj,BestRegCoeff;

  lstDim:=SIZE(Data);
  NumRows:=lstDim(1);

  IF MinOrder<1 THEN
    MinOrder:=1;
  END;

  // initialize best regression data
  BestOrder=0;
  BestRsqr:=0;
  BestRsqrAdj=0;
  BestAICc:=1E499;
  // initialize vector y
  VectY:=MAKEMAT(1,NumRows,1);

  // calculate ymean … needs to be done once!
  Sum1:=0;
  FOR i FROM 1 TO NumRows DO
    y:=Data(i,SelYCol);
    VectY(i,1):=y;
    Sum1:=Sum1+y;
  END;
  YMean:=Sum1/NumRows;
```

Statment

```
  // calculate sum of y – ymean squared
  Sum2:=0;
  FOR i FROM 1 TO NumRows DO
    y:=Data(i,SelYCol);
    Sum2:=Sum2+(y-YMean)^2;
  END;

  // iterate for the specified range of polynomial orders
  FOR Order FROM MinOrder TO MaxOrder DO
    // (re)create the matrix MatX
    MatX:=MAKEMAT(1,NumRows,1+Order);
    // fill the columns to to Order+1 with x^power values
    FOR i FROM 1 TO NumRows DO
      x:=Data(i,SelXCol);
      FOR k FROM 1 TO Order DO
        MatX(i,k+1):=x^k;
      END;
    END;

    // calculate regression coefficients
    RegCoeff:=LSQ(MatX,VectY);

    // calculate the coefficient of determination
    Sum1:=0;
    Sum3:=0;
    Yhat:=MatX*RegCoeff;
    FOR i FROM 1 TO NumRows DO
      Sum1:=Sum1+(Yhat(i,1)-YMean)^2;
      Sum3:=Sum3+(VectY(i,1)-Yhat(i,1))^2;
    END;
    Rsqr:=Sum1/Sum2;

    // calculate the adjusted coefficient of determination
    RsqrAdj:=1-(1-Rsqr)*(NumRows-1)/(NumRows-Order-1);

    k:=Order+1;
    // if Sum3 is 0 then adjust it to a small value
    // to avoid getting a LN(0) error
    IF Sum3==0 THEN
      Sum3:=1E-499;
    END;
    // calculate AICc statistic
    AICc:=NumRows*LN(Sum3/NumRows)+2*k+(2*k*(k+1))/(NumRows-k-1);
    // display intermediate results
    PRINT("Order="+Order+", AICc="+AICc+", Sum3="+Sum3);
    // found a better fit?
```

Statment
`    IF AICc<BestAICc THEN`
`      // update best regression data`
`      BestOrder:=Order;`
`      BestRsqr:=Rsqr;`
`      BestRsqrAdj:=RsqrAdj;`
`      BestAICc:=AICc;`
`      BestRegCoeff:=RegCoeff;`
`    END;`
`  END;`
`  RETURN {BestOrder,BestRsqr,BestRsqrAdj,BestAICc,BestRegCoeff};`
`END;`

Table 7 – The source code for function BestPolyReg

Let's test the function **BestPolyReg**. Table 8 shows sample (x,y) data. Enter the data in the global matrix M2.

x	*y*
1	1
2	5
3	10
4	15
5	25
6	35
7	50
8	65
9	80
10	100

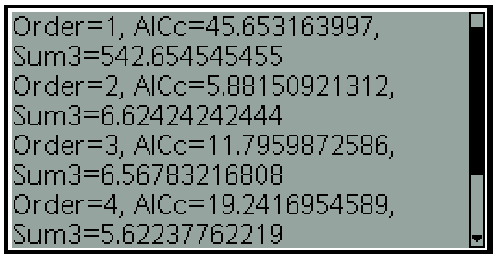
Table 8 – The sample data used to test function BestPolyReg.

Invoke the function **BestPolyReg** by using the following command:

```
BestPolyReg(M2,1,2,1,5)
```

The above command specifies matrix M2 as the data source matrix. The second and third arguments specify that variable x and y are in columns 1 and 2, respectively, of matrix M2. The last two arguments in the call to function **BestPolyReg** specify that the polynomial orders examined are in the range of 1 to 5.

Figures 3 and 4 show the intermediate output of the function using the **PRINT** statement. Figure 3 shows the upper portion of the **PRINT** statement output. Figure 4 shows the lower portion of the **PRINT** statement output. Notice that the polynomial order of 2 has the smallest AIC$_C$ statistic. At the same time,



```
Order=1, AICc=45.653163997,
Sum3=542.654545455
Order=2, AICc=5.88150921312,
Sum3=6.62424242444
Order=3, AICc=11.7959872586,
Sum3=6.56783216808
Order=4, AICc=19.2416954589,
Sum3=5.62237762219
```

Fig. 3 – The upper portion of the PRINT statements output.

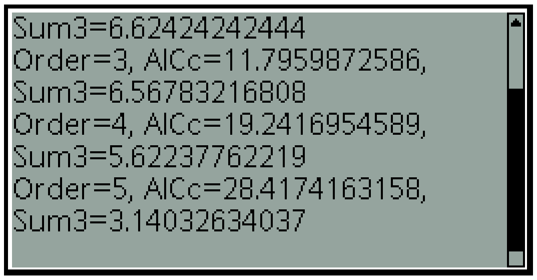

```
Sum3=6.62424242444
Order=3, AICc=11.7959872586,
Sum3=6.56783216808
Order=4, AICc=19.2416954589,
Sum3=5.62237762219
Order=5, AICc=28.4174163158,
Sum3=3.14032634037
```

Fig. 4 – The lower portion of the PRINT statements output.

Figures 5 and 6 show the left and right sides of the list of output values. The output indicates that the quadratic polynomial is the best fit for the data in Table 8. The values for the coefficient of determination and adjusted coefficient of determination are 0.999364666385 and 0.999183142495, respectively. The AIC$_C$ statistic for the best polynomial fit is 5.88. The best regression polynomial is:

y = 0.56666666 – 0.137878789 x + 1.007575758 x^2

Note that values of y in Table 5 are based on the quadratic polynomial y = x^2 with added errors. The function **BestPolyReg** has succeeded in identifying the quadratic polynomial as the one providing the best regression model.

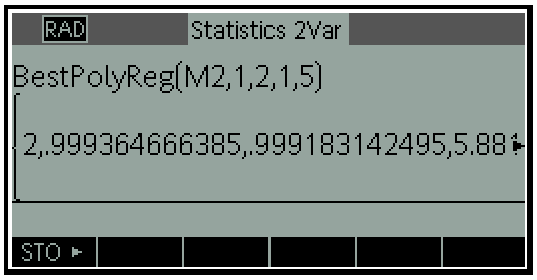

```
RAD   Statistics 2Var
BestPolyReg(M2,1,2,1,5)
[
 2,.999364666385,.999183142495,5.881▸
]
STO ▸
```

Fig. 5 – The left side of the output.

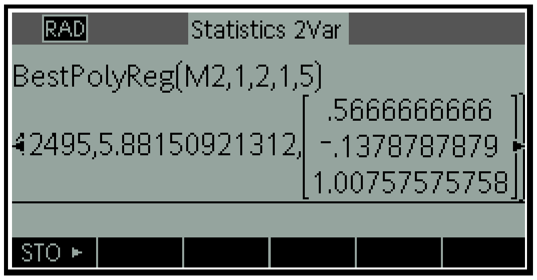

```
RAD   Statistics 2Var
BestPolyReg(M2,1,2,1,5)
                          [ .5666666666  ]]
◂2495,5.88150921312,  -.1378787879   ▸
                          [1.00757575758]]
STO ▸
```

Fig. 6 – The right side of the output.

Observations and Conclusions

This article presented HP 39gII functions that perform bets linearized regression between two, three, and four variables. These functions go through a long list of linearized regression models and find the nest models, sorting the results using the values of the coefficient of determination. The article also offered a function that finds the best polynomial that fits (x, y) data points. The article demonstrated that the modified Akaike information criterion is very suitable to select the best polynomial.

The next article discusses least-squares relative error regression. The article will introduce you to the math behind relative error regression. Moreover, it will present several HP 39gII functions that apply calculations for this type of regression to general linear, polynomial, and linearized models.

Appendix

Here is the listing for function **BestML3**:

Statement

```
EXPORT BestMLR3(Data,lstSel,lstX,lstZ,lstT,lstY)
BEGIN
  LOCAL Tx,Tz,Ty,Tt,i,j,k;
  LOCAL SelXCol,SelZCol,SelTCol,SelYCol;
  LOCAL PowerX,PowerZ,PowerT,PowerY;
  LOCAL MatX,VectY,RegCoeff,MatRes;
  LOCAL lstDim,NumRows,ResUpdated;
  LOCAL Sum1,Sum2,Rsqr,YMean,Yhat;
  LOCAL MaxRes,NumColRes;

  // use default transformation list
  // if lstX is an empty list
  IF SIZE(lstX)==0 THEN
    lstX:={-3,-2,-1,-0.5,0,0.5,1,2,3};
  END;

  // use default transformation list
  // if lstZ is an empty list
  IF SIZE(lstZ)==0 THEN
    lstZ:={-3,-2,-1,-0.5,0,0.5,1,2,3};
  END;

  // use default transformation list
  // if lstT is an empty list
  IF SIZE(lstT)==0 THEN
    lstT:={-3,-2,-1,-0.5,0,0.5,1,2,3};
  END;

  // use default transformation list
  // if lstY is an empty list
  IF SIZE(lstY)==0 THEN
    lstY:={-3,-2,-1,-0.5,0,0.5,1,2,3};
  END;

  MaxRes:=20;
  NumColRes:=9;
  // get the variable selectors
  SelXCol:=lstSel(1);
  SelZCol:=lstSel(2);
  SelTCol:=lstSel(3);
  SelYCol:=lstSel(4);
```

Statement
// get the number of rows in matrix Data
lstDim:=SIZE(Data);
NumRows:=lstDim(1);
// create the regression and results matrices
MatX:=MAKEMAT(1,NumRows,3);
VectY:=MAKEMAT(1,NumRows,1);
MatRes:=MAKEMAT(0,MaxRes,NumColRes);
// start the calculations
// process transformations for variable x
FOR Tx FROM 1 TO SIZE(lstX) DO
PowerX:=lstX(Tx);
// transform x
IF PowerX==0 THEN
FOR i FROM 1 TO NumRows DO
MatX(i,2):=LN(Data(i,SelXCol));
END;
ELSE
FOR i FROM 1 TO NumRows DO
MatX(i,2):=Data(i,SelXCol)^PowerX;
END;
END;
// process transformations for variable z
FOR Tz FROM 1 TO SIZE(lstZ) DO
PowerZ:=lstZ(Tz);
// transform z
IF PowerZ==0 THEN
FOR i FROM 1 TO NumRows DO
MatX(i,3):=LN(Data(i,SelZCol));
END;
ELSE
FOR i FROM 1 TO NumRows DO
MatX(i,3):=Data(i,SelZCol)^PowerZ;
END;
END;
// process transformations for variable t
FOR Tt FROM 1 TO SIZE(lstT) DO
PowerT:=lstT(Tt);
// transform t
IF PowerT==0 THEN

Statement

```
  FOR i FROM 1 TO NumRows DO
    MatX(i,4):=LN(Data(i,SelTCol));
  END;
ELSE
  FOR i FROM 1 TO NumRows DO
    MatX(i,4):=Data(i,SelTCol)^PowerT;
  END;
END;

// process transformations for variable y
FOR Ty FROM 1 TO SIZE(lstY) DO
 PowerY:=lstY(Ty);

 // transform y
 IF PowerY==0 THEN
   FOR i FROM 1 TO NumRows DO
     VectY(i,1):=LN(Data(i,SelYCol));
   END;
 ELSE
   FOR i FROM 1 TO NumRows DO
     VectY(i,1):=Data(i,SelYCol)^PowerY;
   END;
 END;

 // calculate regression coefficients
 RegCoeff:=LSQ(MatX,VectY);

 // calculate ymean
 Sum1:=0;
 FOR i FROM 1 TO NumRows DO
   Sum1:=Sum1+VectY(i,1);
 END;
 YMean:=Sum1/NumRows;

 // calculate the coefficient of determination
 Sum1:=0;
 Sum2:=0;
 Yhat:=MatX*RegCoeff;
 FOR i FROM 1 TO NumRows DO
   Sum1:=Sum1+(Yhat(i,1)-YMean)^2;
   Sum2:=Sum2+(VectY(i,1)-YMean)^2;
 END;
 Rsqr:=Sum1/Sum2;

 // Rsqr is better than last entry
 // in the results matrix?
```

Statement

```
      IF Rsqr>MatRes(MaxRes,1) THEN
        ResUpdated:=0;
        // check which row to insert better
        // regression results
        FOR i FROM 1 TO MaxRes DO
          // insert new results in row i?
          IF ResUpdated==0 AND Rsqr>MatRes(i,1) THEN
            // inserting inside the results matrix?
            IF i<MaxRes THEN
              // downward copy rows from MaxRes-1 to i
              j:=MaxRes-1;
              REPEAT
                FOR k FROM 1 TO NumColRes DO
                  MatRes(j+1,k):=MatRes(j,k);
                END;
                j:=j-1;
              UNTIL j<i;
            END;
            // insert better results in row i
            MatRes(i,1):=Rsqr;
            MatRes(i,2):=PowerY;
            MatRes(i,3):=PowerX;
            MatRes(i,4):=PowerZ;
            MatRes(i,5):=PowerT;
            FOR k FROM 1 TO 4 DO
              MatRes(i,5+k):=RegCoeff(k,1);
            END;
            ResUpdated:=1;
          END;
        END;
      END;
     END; // FOR Ty
    END; // FOR Tt
   END; // FOR Tz
  END; // FOR Tx

  RETURN MatRes;
END;
```

References

1. Wikipedia article *Akaike information criterion*.
2. Wikipedia article *Coefficient of Determination*.
3. Wikipedia article *Linear Regression*.
4. Draper and Smith, *Applied Regression Analysis*, Wiley-Interscience; 3rd edition (April 23, 1998)

5. Neter, Kuther, Wasserman, and Nachtsheim, *Applied Linear Statistical Models*, McGraw-Hill/Irwin; 4th edition (February 1, 1996).
6. Fox, *Applied Regression Analysis and Generalized Linear Models*, Sage Publications, Inc; 2nd edition (April 16, 2008).
7. Montgomery, Peck, and Vining, *Introduction to Linear Regression Analysis*, Wiley-Interscience; 4th edition (2006).
8. Seber and Lee, *Linear Regression Analysis*, Wiley; 2nd edition (February 5, 2003).

About the Author

Namir Shammas is a native of Baghdad, Iraq. He resides in Richmond, Virginia, USA. Namir graduated with a degree in Chemical Engineering. He received a master degree in Chemical engineering from the University of Michigan, Ann Arbor. He worked for a few years in the field of water treatment before focusing for 17 years on writing programming books and articles. Later he worked in corporate technical documentation. He is a big fan of HP calculators and collects many vintage models. His hobbies also include traveling, music, movies (especially French movies), chemistry, cosmology, Jungian psychology, mythology, statistics, and math. As a former PPC and CHHU member, Namir enjoys attending the HHC conferences. *Email me at: nshammas@aol.com*

HP 39gII Regression: Part IV

HP Solve #30 page 72

HP 39gII Regression: Part IV
Least-Squares Relative Error Regression

Namir Shammas

Introduction

Ordinary least-squares (OLS) errors regression minimizes the sum of squared errors for the regression models. These regression models include all kinds of linear and nonlinear equations that describe the relationship between a dependent variable and one or more independent variables. Conceptually, OLS regression treats all errors, in the dependent variable, as having an equal weight, concern, and importance. There are cases where you need to reduce the relative (or percentage) errors. Consider the example where you are measuring distance versus time to calculate the speed as the slope of the first two variables. You have distance readings between 10 ft, and 100 ft in increments of 5 feet. Repeated measurements tell you that you get an error of $\pm$0.5 ft for each reading. Of course, an error of $\pm$0.5 ft for a 10 ft reading is more serious than that for the higher readings. You like to perform a curve fit that minimizes the least-squares *relative* errors so that your regression model gives better predictions for smaller distance values. This is where least-squares relative error (LSRE) regression is relevant. The history of statistical calculations has favored OLS regression far more than LSRE regression, since the OLS equations are simpler to derive. This favoritism is somewhat unfortunate, since it does not give researchers a fair choice between OLS and LSRE regression analysis tools. This article looks at LSRE regression using the HP 39gII calculator.

In this article I present three HP 39gII functions that are the RLSE versions of OLS regression functions that I presented in part I of this series. The two flavors of regression functions have the same parameter lists and returned results. These similarities make the RLSE functions easy to use once you have become familiar with their OLS counterpart.

☞ In this series of articles, I use the term *regression model* to mean the equation that is used in the regression calculations to describe the relationship between a dependent variable and one or more independent variables.

LSRE Regression 101

Consider the simplest linear regression model:

$y = A + B\,x$

Where A and B are the intercept and slope, respectively. Variables x and y are the independent and the dependent variables, respectively. The following equation calculates the error for the linear regression:

$OE_i = y_i - A - B\,x_i$

In the case of least-squares relative errors, the relative error is:

$RE_i = OE_i / y_i = (y_i - A - B\,x_i) / y_i = (1 - A/y_i - B\,x_i/y_i)$

☞ The RLSE regression cannot have zeros as values for the dependent variable or its transformation. Such values generate runtime divide-by zero errors when calculating the least-squares relative errors.

OLS regression minimizes the sum of squared errors as expressed in the following equation:

$L_{OLS} = \sum(y_i - A - B\ x_i)^2$

Whereas LSRE regression minimizes the sum of squared relative errors as expressed in the following equation:

$L_{LSRE} = \sum(1 - A/y_i - B\ x_i/y_i)^2$

To obtain the values of A and B for OLS regression, you perform the following steps:

1. Expand the summation of L_{OLS}.
2. Derive L_{OLS} with respect to A and with respect to B to obtain two linear equations.
3. Solve the two linear equations to evaluate A and B using various statistical summations of x and y.

The above steps yield the following commonly known equations for the slope and intercept:

$B = [n \cdot \sum x_i y_i - \sum x_i \cdot \sum y_i] / [\ n \cdot \sum x_i\text{^}2 - (\sum x_i)\text{^}2]$

$A = (\sum y_i - B \cdot \sum x_i) / n$

Where n is the number of observations. In the case of LSRE regression we follow a similar procedure to calculate A and B from the equation for L_{LSRE}. The equations that calculate B and A are:

$B = b1 / b2$

$b1 = \sum(x_i/y_i) \cdot \sum(1/y_i\text{^}2) - \sum(x_i/y_i\text{^}2) \cdot \sum(1/y_i)$

$b2 = \sum((x_i/y_i)\text{^}2) \cdot \sum(1/y_i\text{^}2) - [\sum(x_i/y_i\text{^}2)]\ \text{^}2$

$A = [\sum(1/y_i) - B \cdot \sum(x_i/y_i\text{^}2)]\ / \sum(1/y_i\text{^}2)$

The above equations are quite different from their OLS counterpart. Also notice that calculating the RLSE slope and intercept does not involve the number of observations.

In matrix form, you can calculate the OLS regression coefficients vector, **b**, for a linearized regression model using the following equation:

$\mathbf{b} = (\mathbf{X}^T\ \mathbf{X})^{-1}\ (\mathbf{X}^T\ \mathbf{y})$

Where matrix **X** has multiple columns that represent values for the independent variables and/or their transformations. These transformations include the natural logarithm, reciprocal, and square, just to name a few. The first column in matrix **X** is typically filled with the constant 1 to calculate a constant for the fitted regression model. The column vector **y** contains the values of the dependent variable or its transformations.

In the case of LSRE regression the matrix form for solving vector **b** is:

$\mathbf{b} = (\mathbf{X}^T\ \mathbf{D}^2\ \mathbf{X})^{-1}\ (\mathbf{X}^T\ \mathbf{D}^2\ \mathbf{y})$

Where **D** is a square matrix with zeros except for diagonal elements having $1/y_i$ values.

To calculate the goodness of fit for the LSRE regression, you obtain the value for the coefficient of determination using the following equation:

$$R^2_{LSRE} = \frac{\sum_1^n((\hat{y}_i - \bar{y})/y_i)^2}{\sum_1^n((y_i - \bar{y})/y_i)^2}$$

Where $\hat{y}_i$ is the predicted value of y at the values of the independent variables used in the calculations, $\bar{y}$ is the average value of y, and y_i is the values of y entering in the regression calculations. Compare the above equation with its OLS regression counterpart:

$$R^2_{OLSE} = \frac{\sum_1^n(\hat{y}_i - \bar{y})^2}{\sum_1^n(y_i - \bar{y})^2}$$

A Calculation Trick

You may have noticed that I jumped from the basic equations for calculating a simple LSRE linear regression model to presenting the matrix form that describes the LSRE calculations for a wide variety of regression models. I could have gradually presented more advanced LSRE regression equations. For example, I could have listed the equations for an LSRE quadratic polynomial regression and also for the simplest LSRE multiple regression that involves two independent variables. This process would have added several pages filled with equations. Instead I chose to present a calculation trick where we can use the OLS regression calculations to calculate the coefficients for LSRE regression models. The trick relies on looking at the expression for L_{LSRE} and viewing it in a different light--as if it were L_{OLS}:

$L_{OLS} = \sum(1 – A/y_i – B\ x_i/y_i)^2$

The above equation indicates that we are fitting the following OLS regression model:

$1 = A/y + B\ x/y$

Or,

$\alpha = A/y + B\ x/y$

The above (very strange) equation is an OLS multiple regression model with the following features:

- The values for the (new) dependent variable (call it α) are always 1.
- The model is a linearized multiple regression. It uses the independent variables x and y in the transformations 1/y and x/y. Notice that I have changed the status of the original dependent variable y, temporarily making it an independent variable.
- The regression model has NO constant term.

Likewise, if we have a quadratic polynomial regression model:

$y = A + B\ x + C\ x^2$

Our trick yields the following equation that can use OLS regression calculations:

$1 = A/y + B\ x/y + C\ x^2/y$

In the case of a simple multiple linear regression model:

$y = A + B\ x + C\ z$

Our trick yields the following equation that can use OLS regression calculations:

1 = A/y + B x/y + C z/y

I have offered you the above examples so that you can see the pattern used in the trick that allows us to apply OLS regression calculations with any RLSE regression model. Notice that in all of the above examples (and for any other regression model), each right-hand-side term in the regression model is divided by the dependent variable y. Also notice that each left-hand-side is the new variable α, whose values are systematically equal to 1.

The above features allow us to use OLS regression calculation tools, such as the function **LSQ** on the HP 39gII. However, we *MUST* observe the following simple calculation rules.

- The matrix **X** has no column populated with 1s, since the regression calculations yields NO constant term.
- Each column in matrix **X** has a term from the modified multiple regression model.
- The column vector **y** now represents the new variable α and is populated with 1s.

To obtain the LSRE version for the coefficient of determination, we use the definition for that statistic that I showed you earlier. We must also observe the following rules:

- The values of $\hat{y}_i$ used to calculate the LSRE coefficient of determination must be based on the *original* regression model. This is the model that shows variable y as the dependent variable.
- If you transformed the observed values of y, then you need to work with the transformed values and not the original observed ones. Likewise, the values for $\hat{y}_i$ must be for the transformed values of y.
- We cannot use the formula for the OLS coefficient of determination with LSRE regression models, because each version is defined differently. That would be like mixing apples and oranges!

General RLSE Regression

Let's start with the general case of simple multiple regression, with no transformation of variables. I introduced you, in part I, the function **MLR2** to calculate the coefficient of determinations and the regression coefficients. Table 1 presents function **RErMLR2** which is the RLSE counterpart of function **MLR2**. The function **RErMLR2** has the following parameters:

- The parameter **MatX** which represents the matrix **X**.
- The parameter **VectY** which represents the vector **y**.

The function returns a list containing the coefficient of determination and the regression coefficients (as a column matrix).

Statement
`EXPORT RErMLR2(MatX,VectY)`
`BEGIN`
`  LOCAL i,j,y,Rsqr;`
`  LOCAL lstDimX,NumRows,NumCols;`
`  LOCAL YMean,Sum1,Sum2,Yhat;`
`  LOCAL TMatX,VectOnes,RegCoeff;`
`  // calculate the number of observations and variables`

Statement

```
  lstDimX:=SIZE(MatX);
  NumRows:=lstDimX(1);
  NumCols:=lstDimX(2);

  // create transformation matrices
  TMatX:=MAKEMAT(1,NumRows,NumCols+1);
  VectOnes:=MAKEMAT(1,NumRows,1);
  FOR i FROM 1 TO NumRows DO
    y:=VectY(i,1);
    TMatX(i,1):=1/y;
    FOR j FROM 1 TO NumCols DO
      TMatX(i,j+1):=MatX(i,j)/y;
    END;
  END;
  // calculate the regression coefficients
  RegCoeff:=LSQ(TMatX,VectOnes);

  // calculate ymean
  Sum1:=0;
  FOR i FROM 1 TO NumRows DO
    y:=VectY(i,1);
    Sum1:=Sum1+y;
  END;
  YMean:=Sum1/NumRows;

  // calculate the coefficient of determination
  Sum1:=0;
  Sum2:=0;
  FOR i FROM 1 TO NumRows DO
    y:=VectY(i,1);
    Yhat:=RegCoeff(1,1);
    FOR j FROM 1 TO NumCols DO
      Yhat:=Yhat+RegCoeff(j+1,1)*MatX(i,j);
    END;
    Sum1:=Sum1+((Yhat-YMean)/y)^2;
    Sum2:=Sum2+((y-YMean)/y)^2;
  END;
  Rsqr:=Sum1/Sum2;
  // return the results as a list
  RETURN {Rsqr,RegCoeff};
END;
```

Table 1 – The source code for function RErMLR2.

Since the topic of LSRE regression is not as popular as OLS regression, I would like to point out the following aspects of the source code in Table 1:

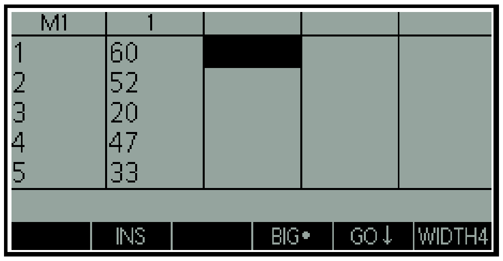
- The function creates the column vector **VectOnes** using the **MAKEMAT** function. This task fills the vector **VectOnes** with 1s. These values remain unchanged during the function execution because they represent the values of the variable α.
- The function creates the transformations matrix **TMatX**. This matrix has the same number of rows as parameter **MatX**, but has one more column than **MatX**,
- The values for the first column of matrix **TMatX** are 1/y instead of 1 (as is the case in OLS regression).
- The values for the remaining columns of matrix **TMatX** are x_n divided by y.
- The loop that calculates the LSRE coefficient of determination uses values for $\hat{y}_i$ calculated as $a_1 + a_2 x_1 + a_3 x_2 + \ldots + a_{m+1} x_m$. The column matrix variable **RegCoeff** contains the regression coefficients a_1, a_2, a_3, …, and a_{m+1}.
- The function calculates the LSRE coefficient of determination using the definition that I presented for that statistic.

Let's use function **RErMLR2** with the sample data in Table 2. This is the same data I used to test function **MLR2** in part I. I am reusing the same data so I can compare the results of functions **RErMLR2** and **MLR2**.

x_1	x_2	x_3	y
7	25	6	60
1	29	15	52
11	56	8	20
11	31	8	47
7	52	6	33

Table 2 – Sample data.

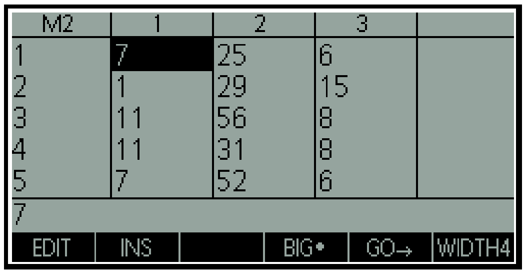
You need to store the values of the dependent variable y as a column in matrix M1 and the values of the independent variables in matrix M2. Figure 1 shows the contents of matrix M1 which stores the values for the column vector **y**.



M1	1
1	60
2	52
3	20
4	47
5	33

Fig. 1 – The values in matrix M1.

Figure 2 shows the contents of matrix M2 which stores the values for the matrix **X**.



M2	1	2	3
1	7	25	6
2	1	29	15
3	11	56	8
4	11	31	8
5	7	52	6

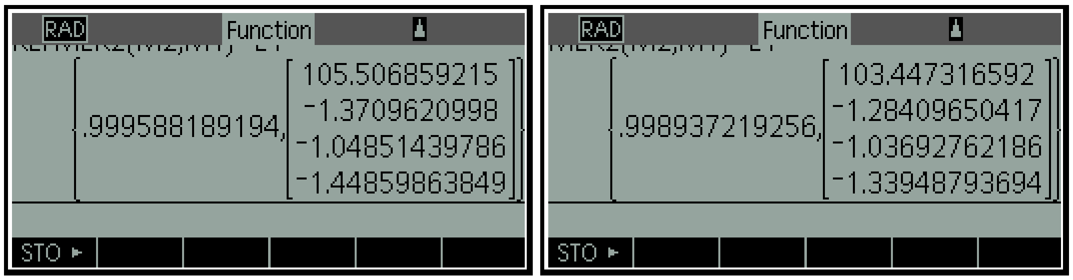
7

Fig. 2 – The values in matrix M2.

Let's use function **RErMLR2** with the data in Table 2. Type the following command:

```
RErMLR2(M2,M1)→L1
```

The above command stores the results in list L1 for further examination if so desired. Figure 3 shows the results of executing the function **RErMLR2** along with the output of function **MLR2** using the same data:



```
{.999588189194, [105.506859215 ]
                [-1.3709620998 ]
                [-1.04851439786]
                [-1.44859863849]}
```

```
{.998937219256, [103.447316592 ]
                [-1.28409650417]
                [-1.03692762186]
                [-1.33948793694]}
```

Fig. 3 – The results of executing function RErMLR2 (left) and function MLR2 (right).

The results in Figure 3 show that the RLSE R^2 is 0.99959. This value is slightly higher than the OLS R^2 which is 0.99894. The RLSE regression model is:

$y = 105.5069 - 1.3710\ x_1 - 1.0485\ x_2 - 1.44860\ x_3$

The regression coefficients for both RLSE and OLS regression are somewhat close to each other, as expected.

LSRE Polynomial Regression

Next we will be working with LSRE polynomial regression. Table 3 shows the source code for the function **RErPolyReg**. This function has the following parameters:

- The parameter **DSMat** represents the matrix that has the x and y data.
- The parameter **SelXCol** designates the column in matrix **DSMat** which contains the values for x.
- The parameter **SelYCol** designates the column in matrix **DSMat** which contains the values for y.
- The parameter **Order** selects the order for the polynomial regression. If you pass an argument of 1 to this parameter, the function **RErPolyReg** performs an LSRE linear regression. Passing values of 2 and 3 to the parameter **Order** cause the function to fit the data with a quadratic and cubic polynomial, respectively.

The function **RErPolyReg** returns a list that contains the value of the LSRE coefficient of determination and a column matrix containing the regression coefficients. I recommend that you store the results of calling function **RErPolyReg** in a list so that you can further examine and/or use the results.

Statement
`EXPORT RErPolyReg(DSMat,SelXCol,SelYCol,Order)`
`BEGIN`
`  LOCAL i,j,x,y;`
`  LOCAL lstDimX,NumRows;`
`  LOCAL MatX,VectOnes,RegCoeff;`
`  LOCAL YMean,Sum1,Sum2,Yhat,Rsqr;`

Statement
// get the number of rows
lstDimX:=SIZE(DSMat);
NumRows:=lstDimX(1);
// create the regression matrices
MatX:=MAKEMAT(1,NumRows,Order+1);
VectOnes:=MAKEMAT(1,NumRows,1);
// populate the regression matrices
FOR i FROM 1 TO NumRows DO
y:=DSMat(i,SelYCol);
x:=DSMat(i,SelXCol);
MatX(i,1):=1/y;
FOR j FROM 1 TO Order DO
MatX(i,j+1):=(x^j)/y;
END;
END;
// calculate the regression coefficients
RegCoeff:=LSQ(MatX,VectOnes);
// calculate ymean
Sum1:=0;
FOR i FROM 1 TO NumRows DO
Sum1:=Sum1+DSMat(i,SelYCol);
END;
YMean:=Sum1/NumRows;
// calculate the coefficient of determination
Sum1:=0;
Sum2:=0;
FOR i FROM 1 TO NumRows DO
x:=DSMat(i,SelXCol);
y:=DSMat(i,SelYCol);
Yhat:=RegCoeff(1,1);
FOR j FROM 1 TO Order DO
Yhat:=Yhat+RegCoeff(j+1,1)*x^j;
END;
Sum1:=Sum1+((Yhat-YMean)/y)^2;
Sum2:=Sum2+((y-YMean)/y)^2;
END;
Rsqr:=Sum1/Sum2;
// return the list of results
RETURN {Rsqr,RegCoeff};
END;

Table 3 – The source code for function RErPolyReg.

Since the topic of LSRE polynomial regression is not as popular as OLS polynomial regression, I would like to point out the following aspects of the source code in Table 3:

- The function creates the column vector **VectOnes** using the **MAKEMAT** function. This task fills the vector **VectOnes** with 1s. These values remain unchanged during the function execution because they represent the values of the variable α.
- The function creates the transformations matrix **MatX** to store the terms for the polynomials. The number of columns is equal to the polynomial order plus one/
- The values for the first column of matrix **MatX** are 1/y.
- The values for the remaining columns of matrix **MatX** are the result of raising x to some power, divided by y.
- The loop that calculates the LSRE coefficient of determination uses values for $\hat{y}_i$ calculated as $a_1 + a_2 x + a_3 x^2 + \ldots + a_{m+1} x^m$. The column matrix variable **RegCoeff** contains the regression coefficients a_1, a_2, a_3, …, and a_{m+1}.

Let's use the function **RErPolyReg** to fit a cubic polynomial using the data in Table 4. Enter the data in matrix M1.

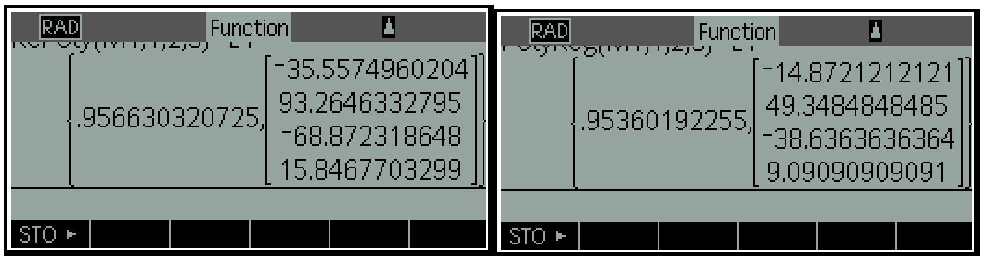
x	*y*
1	5.1
1.1	4.4
1.2	4.6
1.3	4.0
1.4	3.2
1.5	3.2
1.6	2.4
1.7	2.2
1.8	1.3
1.9	2.0

Table 4 – Sample data for a cubic polynomial fit.

To obtain the regression coefficients and coefficient of determination for the cubic polynomial fit using the data in Table 4, execute the following command:

```
RErPolyReg(M1,1,2,3)→L1
```

The first argument of calling function **RErPolyReg** is the matrix M1 which contains the (x, y) data points. The second argument is 1 which tells the function that the data for the independent variable x are in column 1 of M1. The third argument is 2, which tells the function that the data for the dependent variable y are in column 2 of M1. The last argument is 3, which represent the order of the sought polynomial. I assigned the results to list L1 so that I can examine the results later, if I needed to. Figure 4 shows the output of using function **RErPolyReg** as well as function **PolyReg** (from part I)/.



```
{.956630320725,[-35.5574960204]
               [93.2646332795 ]
               [-68.872318648 ]
               [15.8467703299 ]}
```

```
{.95360192255,[-14.8721212121]
              [49.3484848485 ]
              [-38.6363636364]
              [9.09090909091 ]}
```

Fig. 4 – The results of executing function LSRErPolyReg (left) and function PolyReg (right).

The results show that R^2 is 0.95663 and the fitted polynomial is:

$Y = -35.5575 + 93.2646\ x - 68.87232\ x^2 + 15.84677\ x^3$

The value of R^2 indicates that the polynomial explains about 95.7% of the variation in the values of y. The value of the RLSE R^2 is slightly higher than of its OLS counterpart. The coefficients of RLSE polynomial regression are quite different in values from their OLS counterpart.

Power Curve Fitting

This section looks at the RLSE version of power fitting. I presented function **PowerFit** in part I. In this section I present, in Table 5, its RLSE counterpart, function **RErPowerFit**. The function **RErPowerFit** has two parameters.

- The first parameter is **MatX** which is the matrix that contains the values for the independent variables.
- The second parameter is **VectY** which is the column vector that contains the values for the dependent variable.

The function returns a list containing the RLSE coefficient of determination and the column matrix that stores the regression coefficients.

Statement
`EXPORT RErPowerFit(MatX,VectY)`
`BEGIN`
`  LOCAL i,j,y,LnY;`
`  LOCAL lstDimX,NumRows,NumCols;`
`  LOCAL TMatX,VectOnes,RegCoeff,Rsqr;`
`  LOCAL YMean,Sum1,Sum2,Yhat;`
`  lstDimX:=SIZE(MatX);`
`  NumRows:=lstDimX(1);`
`  NumCols:=lstDimX(2);`
`  TMatX:=MAKEMAT(1,NumRows,NumCols+1);`
`  VectOnes:=MAKEMAT(1,NumRows,1);`
`  Sum1:=0;`
`  FOR i FROM 1 TO NumRows DO`
`    LnY:=LN(VectY(i,1));`
`    Sum1:=Sum1+LnY;`
`    TMatX(i,1):=1/LnY;`
`    FOR j FROM 1 TO NumCols DO`
`      TMatX(i,j+1):=LN(MatX(i,j))/LnY;`
`    END;`
`  END;`
`  // calculate regression coefficients`
`  RegCoeff:=LSQ(TMatX,VectOnes);`
`  // calculate mean ln(y)`
`  YMean:=Sum1/NumRows;`

Statement
`  // calculate coefficient of determination`
`  Sum1:=0;`
`  Sum2:=0;`
`  FOR i FROM 1 TO NumRows DO`
`    Yhat:=RegCoeff(1,1);`
`    FOR j FROM 1 TO NumCols DO`
`      Yhat:=Yhat+RegCoeff(j+1,1)*LN(MatX(i,j));`
`    END;`
`    y:=LN(VectY(i,1));`
`    Sum1:=Sum1+((Yhat-YMean)/y)^2;`
`    Sum2:=Sum2+((y-YMean)/y)^2;`
`  END;`
`  Rsqr:=Sum1/Sum2;`
`  RETURN {Rsqr,RegCoeff};`
`END;`

Table 5 – The source code for function RErPowerFit.

To test the function **RErPowerFit** let’s used the data in Table 6.

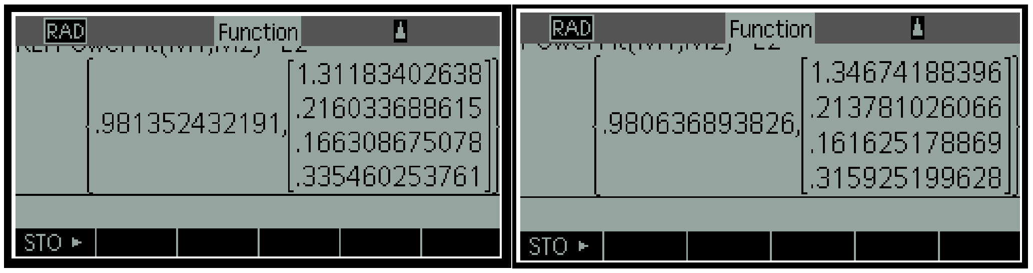
x	*z*	*t*	*y*
1	1	7	7
2	1	5	7.7
3	2	3	7.9
4	2	1	5.3
5	3	2	8.4
6	3	5	11.6
7	4	8	13.6
8	4	9	14.3
9	5	4	12.4
10	5	2	10.6

Table 6 – Sample data for a power fit.

Store the values in the first three columns of Table 6 in matrix M1. Store the values of the rightmost column of Table 6 in the matrix M2. To calculate the coefficient of determination and regression coefficients of a power fit between variables x, z, t, and y, execute the following command:

```
RErPowerFit(M1,M2)→L2
```

Figure 5 shows the results of executing the above command as well as the results from using function **PowerFit** (in Part I).



```
{.981352432191, [[1.31183402638]
                 [.216033688615]
                 [.166308675078]
                 [.335460253761]]}
```

```
{.980636893826, [[1.34674188396]
                 [.213781026066]
                 [.161625178869]
                 [.315925199628]]}
```

Fig. 5 – The results of executing function RErPowerFit (left) and function PowerFit (right).

The results show that R^2 is 0.98135 and the power fit is:

y = 1.31183 + 0.216033 * ln(x) + 0.1663086 * ln(z) + 0.33546 * ln(t)

Or is in the nonlinear form,

y = 3.712962 * (x^ 0.216033)*(z^0.1663086)*(t^0.33546)

The value of R^2 indicates that the power fit explains about 98% of the variation in the values of y. The RLSE R^2 value is slightly higher than its OLS counterparts. The corresponding coefficients of the RLSE and OLS polynomial regression are close to each other, as expected.

Comparing RLSE and OLS Curve Fitting

The three HP 39gII RLSE regression functions that I presented in this article produced better values for the coefficient of determination than their OLS counterparts. In his paper, Tofallis[1] makes reference to the work of Saez and Rittmann[2] that have performed Monte Carlo simulations and found that RLSE regression produced better results than OLS regression models. These researchers discovered that the RLSE regression models yielded regression coefficients with 90% confidence regions that were approximately centered on the true coefficient values. This was not the case with OLS results.

To Infinity and Beyond!

You have seen the source code for three RLSE regression functions. You should be able to convert other HP 39gII regression functions that I presented in parts II and III into RLSE versions. The tasks include:

- Populating the matrix **X** with the correct values. The values in the first column are always $1/y_i$. The values for the other columns should be calculated as the values of the independent variables (or their transformations) divided by y_i.
- Populating the vector **y** with the constant 1.
- Making sure that you correctly calculate the RSLE version of the coefficient of determination.
-

Observations and Conclusions

This article discussed least-squares relative errors. The article first presented some theoretical foundation for basic RLSE calculations. The article also showed you how apply a trick to use OLS regression calculations for RLSE regression. You also learned about calculating the RLSE coefficient of determination. The article also presented three RLSE regression functions for the HP 39gII. These functions are the RSLE versions of OLS regression functions that appear in part I of this series.

References

1. Chris Tofallis, *Least Squares Percentage Regression*, Journal Of Modern Applied Statistical Methods, November, 2008, Vol. 7, No. 2, 368-631.
2. Saez and Rittmann, *Model-parameter estimation using least squares*. Water Research, 26(6), 789-796, 1992.
3. Wikipedia article *Coefficient of Determination*.
4. Wikipedia article *Linear Regression*.

5. Wikipedia article *Simple Linear Regression*.
6. Draper and Smith, *Applied Regression Analysis*, Wiley-Interscience; 3rd edition (April 23, 1998)
7. Neter, Kuther, Wasserman, and Nachtsheim, *Applied Linear Statistical Models*, McGraw-Hill/Irwin; 4th edition (February 1, 1996).
8. Fox, *Applied Regression Analysis and Generalized Linear Models*, Sage Publications, Inc; 2nd edition (April 16, 2008).
9. Montgomery, Peck, and Vining, *Introduction to Linear Regression Analysis*, Wiley-Interscience; 4th edition (2006).

About the Author

Namir Shammas is a native of Baghdad, Iraq. He resides in Richmond, Virginia, USA. Namir graduated with a degree in Chemical Engineering. He received a master degree in Chemical engineering from the University of Michigan, Ann Arbor. He worked for a few years in the field of water treatment before focusing for 17 years on writing programming books and articles. Later he worked in corporate technical documentation. He is a big fan of HP calculators and collects many vintage models. His hobbies also include traveling, music, movies (especially French movies), chemistry, cosmology, Jungian psychology, mythology, statistics, and math. As a former PPC and CHHU member, Namir enjoys attending the HHC conferences. *Email me at: nshammas@aol.com*

From The Editor

HP Solve #30 page 86

From The Editor – Issue 30

It is January and a new year is starting. Change is part of most plans for a better 2013. Electronics enthusiasts immediately think of the Consumer Electronics Show and HP is thinking about teachers.

Education is a primary application for computational capability and certainly calculators play an important role as you may read about in the lead article for this issue.

HP Solve will undergo a change in the applications support of calculators with its major emphasizes switching from that of a technical newsletter to that of an educational newsletter. Grades 6 – 20 will be the operational level of content for teachers of science, technology, engineering, and mathematics – STEM. The April issue will reflect this change with a new editor.

Correction for issue 29 – Customer Corner interview with Gary Tenzer. He said that he visited HP in 1972 when they were in Corvallis. Actually, in 1972 they were still in Cupertino and moved to Corvallis a year or so later.

Here is the content of this issue

S08 – Regular Columns

- **From the editor.**
- **One Minute Marvels.**
- **HP User Community News – Felix's Bibliography, A Little Bit of HP History**
- **Identifying a Voyager Series Calculator, Part II.**

That is it for my last issue. I hope you enjoy it.

X < > Y,

Richard

Email me at: hpsolve@hp.com or rjnelsoncf@cox.net

HP 48 One Minute Marvels No. 17 – DOW Addition

One Minute Marvels, OMMs, are short, efficient, unusual, and fun HP 48 programs that may be entered into your machine in a minute or less. These programs were developed on the HP 48, but they will usually run on the HP 49 and HP 50 as well. Note the HP48 byte count is for the program only.

Day of Week routine text addition

In issue # 26, page 7, OMM No. 13 listed two routines to determine the Day of week, DOW given a date. I was reading about a mathematical "discovery" of Quaterions that occurred on October 16, 1843. The day of the week was also given as Monday. I reached for my calculator to check the DOW with **'dow2'** and I realized that the actual day in text was not provided so I wrote a simple routine to provide it. The news article DOW was correct.

I am reproducing the two routines of OMM No. 13 for reader convenience followed by the conversion routine, **'dow3'**, to convert the output number to a day.

Input a standard date in mm.ddyyyy format and **'dow1'** (system flag –42 clear) returns a three-letter day.

'dow1' << 0 TSTR 1 3 SUB >>

5 commands, 22.5 Bytes, # A8D0h. Timing: 8.211999 ⇒ "SAT" in 26.2_ms.

Joseph K. Horn suggests using DDAYS and a known date to calculate the day of week. The known date is a Sunday (year 3,000) and is selected to have the day and month the same so the system flag –42 setting doesn't matter. He had to "hunt" for a date that met these requirements. Given a date in mm.ddyyyy format, **'dow2'** returns a number between 0 (Sunday) and 6 (Saturday). Example: HHC 2011date 9.242011, **'dow2'** returns 6 (Saturday).

'dow2' << 2.023 SWAP DDAYS 7 MOD >>

5 commands, 30.5 bytes, #B181h. Timing: 1 ⇒0 in 7.17_ms.

Two different techniques are used to return the day of week given a date. Both use five commands, but one (**'dow2'**) is 3.7 times faster. **'dow3'** adds a day number conversion so you don't have to remember that Sunday is zero. You may simply add it to the end of **'dow2'** Note the **'dow3'** is fast enough that the extreme speed difference between the two DOW routines (**'dow1' 'dow2')** is still valid.

'dow3' << 1 + { "SUN" "MON" "TUE" "WED" "THR" "FRI" "SAT" } SWAP GET >>

5 commands, 81 bytes, #44DEh. Timing: 1 ⇒ "MION" in 16.5_ms

How **'dow3'** works. This very straight forward routine assumes a digit 0 to 6 on the stack. It doesn't need to check for valid inputs. **'dow2'** leaves a digit on the stack that has one added to make the input 1 to 7 instead of 0 to 6. The third command is a list containing the abbreviated days of the week – you may make these whatever you wish. The input for the get command is the position number of the item in the list so the stack levels 1 & 2 must be exchanged with SWAP before GET is executed.

Adding **'dow3'** to **'dow2'** makes the program complete and more practical because of infrequent use. When you do need DOW, however, it is very nice because calendars covering the 8,419 years that **'dow2'** covers are usually not conveniently available.

HP User Community News

Calculator Bibliography As described in the HHC 2012 Report in issue # 29, pages 13 & 14 Felix Gross (Germany) is working on the most complete bibliography ever compiled of Calculator articles. If an ***HP Solve*** reader is looking for any calculator information he should start with Felix's bibliography. Presently it is nearly 300 pages and it covers all worldwide publications. If you have an obscure publication you would like to have added to the bibliography – or check on one – you may contact Felix at: Felix Gross (felix.gross@alumni.ethz.ch)

CES 2013
The Consumer Electronics Show, CES, was January 8-12 this year. CES is the largest show of its kind and some observers feel that it has gotten too big. This was 50th show and I have attended nearly every one since the beginning when there were two shows each year. Some of the largest companies did not exhibit this year because they felt that other venues were more effective in reaching the intended audience. There wasn't a single major calculator exhibitor this year for the first time. The NCTM conference in April in Denver is where calculators will be the star.

Some of HP's loyal users, however, still gathered for dinner during CES as shown in the photo below.

Jake Schwartz Cherry Hill, NJ; Sam Kim Las Vegas, NV; Brian Walsh Chicago, IL; Richard Nelson Mesa, AZ; Chris Owen Wichita, KS; Jo Vandale Belgium

Ed note: The article reproduced below was written in early 1999 based on the HP49g introduction of May 21, 1999. Time passes so quickly that even though you were there if you don't make a few notes the historical details are easy to forget.

A Little Bit of HP Calculator History

Jake Schwartz

Roughly in 1993 at the time when the HP48GX was released, the calculator R&D was in the process of being transferred from Corvallis to Singapore. At this point it seemed that the Singapore people wished to include the Corvallis folks in their efforts, and thus they used a combined team of developers from both locations to design the HP38G. An attempt was made to crack the high-school market with this machine, and my understanding was that only a handful of HP individuals were assigned the task of educating the teachers about the merits of the 38. This paled in comparison to the literally hundreds of TI folks who had been doing the same thing for a handful of years. As a result, the 38 seemed to go virtually unnoticed in comparison to the massive TI effort which has succeeded in making their graphing models ubiquitous in the school market. Meanwhile, the HP calculator zealots wrote the HP38G off as a failed attempt to be like the others and we continued playing with our HP48G-series toys.

In 1995, the U.S. HP Handheld User's Conference was held in Minneapolis, and HP Singapore's Kheng Joo Khaw (then head of palmtops and calculators) was there to speak on current HP affairs. He mainly was talking about the advent of the Windows CE units, and how HP had decided that they should use a "standard" windowing O.S. in order to make any headway in the growing handheld market (This was despite a successful run with the DOS-based HP 95/100/200LX models). When we asked him about calculators and what would happen next, he indicated that calculators were not their current focus.
One year later, at the 1996 HP Handheld Conference in Anaheim, Khaw was there again, speaking on palmtops and we asked again about the destiny of calculators. And again, he said that they felt that calculators didn't need to be updated at this time, and when they did, there would be some activity in that area. He even went as far to say that the HP48G-series was already too complicated and that it intimidated people, so he saw no need to extend the high end. We felt at that time that effectively, calculators were as good as dead at HP. (I shot the videotapes from these conferences if anyone is interested.)

In June of 1997, our Philadelphia Area HP Handheld Club (still going since 1978) hosted HP's Eric Vogel (who worked on handheld products since 1976). He had recently switched from Corvallis to an instrumentation group which developed the handheld Logic Dart. At the end of his enlightening 4-hour presentation, we asked him about his thoughts on whether there might be another top-of-the-line calculator from Hewlett-Packard. Eric was convinced that the calculator world had seen its heyday and it was time to move on, considering what could be done on any PC on anybody's desk these days. It was sort of a bittersweet moment...we understood his point of view but didn't really want to believe it.

The Summer went by and the British HPCC group hosted their Fifteenth Anniversary conference in London in September of 1997, also denoting the 25th birthday of the HP35. Something different happened there...it was announced that a new calculator team would be forming officially on November 1st in Australia. Apparently the Singapore group really *wasn't* going to do anything, so this new band of developers asked and got the go-ahead to take the effort over. It was precisely at this event when the Australians first met the Meta Kernel team and also saw demonstrations of MK, Erable and ALG48 for the very first time. Obviously this weekend made a significant impression. Following these demos, Richard Nelson asked for a show of hands as to how many people would be willing to buy an HP48 which contained these tools built into ROM. Of course, the vast majority voted in the affirmative.

It was realized then that the ACO group would be on their own, having to prove themselves without the benefit of any other related existing products which already represented a stream of income for them.

Back at home, we began to speculate on how the new group (consisting mainly of people who were not from the old Corvallis team) could make their mark. I speculated that a "ready", "set", "go" approach might allow them to ease their way into the industry. The HP48G+ was their first product - and truly, this could represent ACO's "dipping its toe into the water". Then the HP6S - a completely new but entry-level unit - followed. And now, they have announced their intentions of releasing the HP49G (True, over the past quarter-century HP has always said that they don't talk about unannounced products, so this is a major departure). With all the firmware overhauling which was described in Jean-Yves Avenard's descriptive posting, this is pretty amazing, considering that the team has only been together a year and a half!

Knowing that ACO absolutely MUST generate revenue in order to stay in business, it makes sense that they would chase a larger market in which they previously haven't had a significant presence. In addition, I think that the message which we tried to impose on the ACO people in London - that these calculators still matter very much to professionals as well as students - did sink in. Just like actors who must first work day-to-day jobs to make ends meet at the beginnings of their careers, so too must ACO put cash into its coffers in order to afford the freedom to develop the kind of machines we would prefer to use. Let's hope that the 49G is a stepping stone to The Next Big Thing.

Jake Schwartz

Identifying a Voyager Series Calculator – Part II

In issue # 29 I gave a procedure for identifying the model number of the five original voyager series of calculators if the HP logo was missing. I mentioned that I often had to help people identify the model number while speaking over the telephone.

The procedure I gave was less suitable for a verbal process.

The simplest way to go through the procedure verbally is to answer a series of four questions regarding the upper right corner divide key as shown in Fig. 1. There are gold/yellow and blue notations above and on the sloping surface of the key. The question flow chart asks about the color and text of those notations.

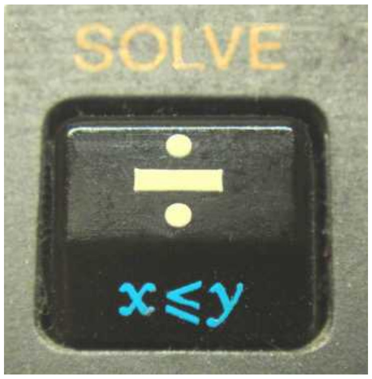


Fig. 1 – Voyager ÷ key, which model is it?

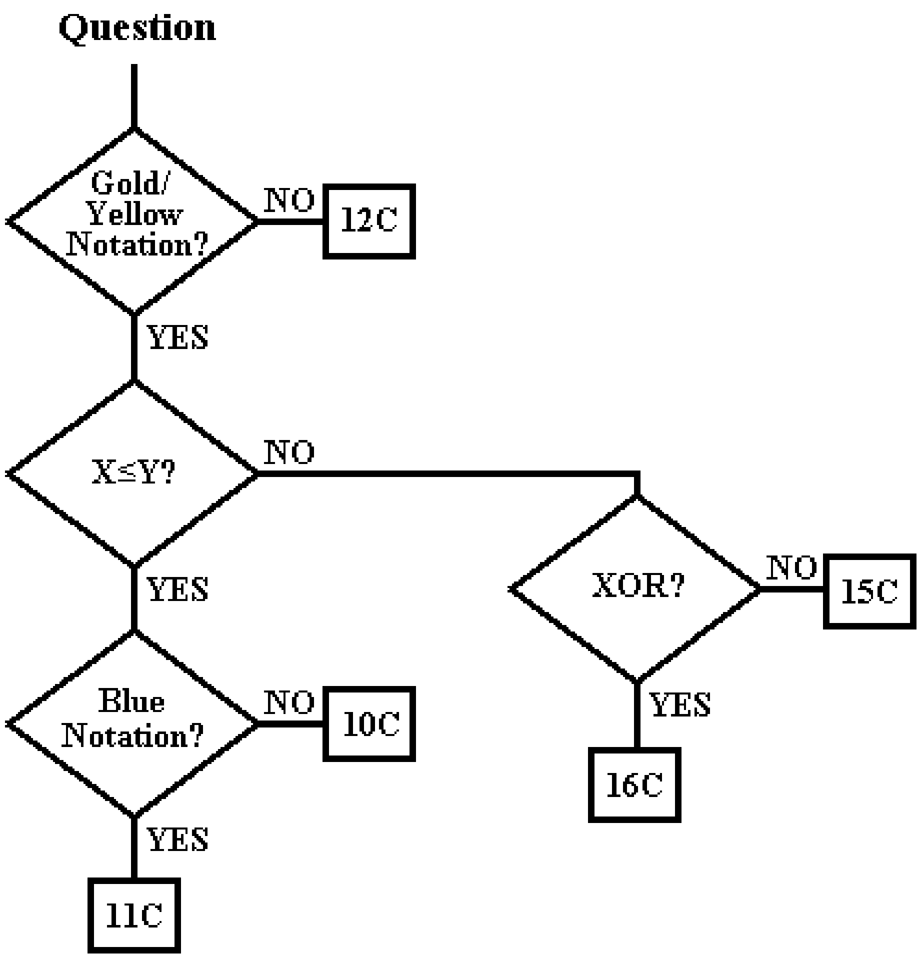
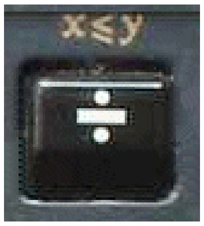


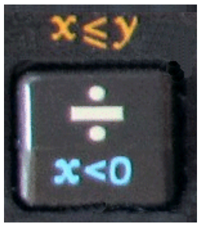
Fig. 2 – Question flow chart to determine model number if HP logo is missing.

Here is a challenge for the reader. Can you make a simpler flow chart of less than four questions? Refer to the five key notations shown below.

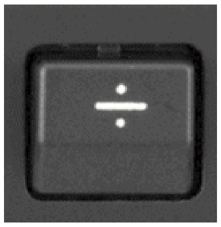
If you are able to visually compare the calculator with the images below you can just ask one question. Which photo matches your calculator divide key? Verbally you could also ask the user to just tell you what the notations are and you pick the correct photo.



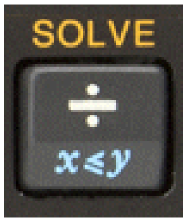
HP-10C



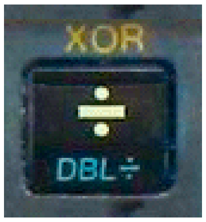
HP-11C



HP-12C



HP-15C



HP-16C

About the Editor

Richard J. Nelson, a long time HP Calculator enthusiast, was editor and publisher of *HP-65 Notes*, *The PPC Journal*, *The PPC Calculator Journal*, and the *CHHU Chronicle.* He has also had articles published in *HP65 Key Note* and *HP Key Notes*. As an Electronics Engineer turned technical writer Richard has published hundreds of articles discussing all aspects of HP Calculators. His work may be found on the Internet and the HCC websites at: hhuc.us . He proposed and published the PPC ROM and actively contributed to the UK HPCC book, *RCL 20*. His primary calculator interest is the User Interface. Richard may be reached at: rjnelsoncf@cox.net