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. As newer machines appeared on the market, they supported more advanced regression calculations. The HP 39gII follows this trend very well. It support matrix/vector calculations and also 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 features. 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 coefficient of a general multiple regression model and the coefficient of determination.

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, 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 expression:

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

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 of 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 effort 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, and 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 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**.

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.

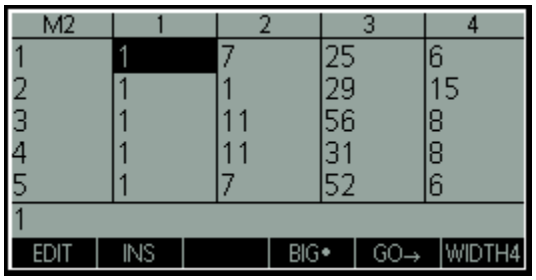

Fig. 2 – The values in matrix M2.

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

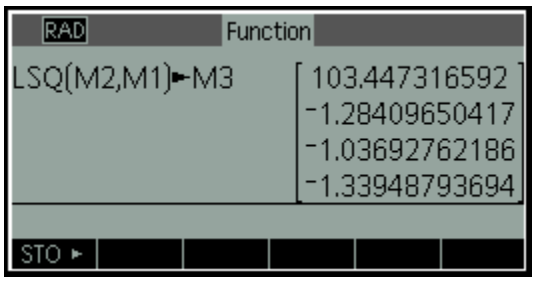

Fig. 3 – The results of executing function LSQ.

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

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

I have presented you with a simple example of using the function **LSQ**. The example works well because it uses the variables without performing any mathematical transformation on them. Moreover, 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 fraction of the variance in variable y that is explained by the regression model we obtain. When the coefficient of determination is 1, its maximum 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, 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 $\hat{y}_i$ 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. 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;**

Statement
`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**.
- Calculate the mean for the y values.
- Calculate 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 multiplication expression **MatX*RegCoeff**. The second **FOR** loop calculates the sums of squared differences between $\widehat{y_i}$ and $\bar{y}$ and 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)**:

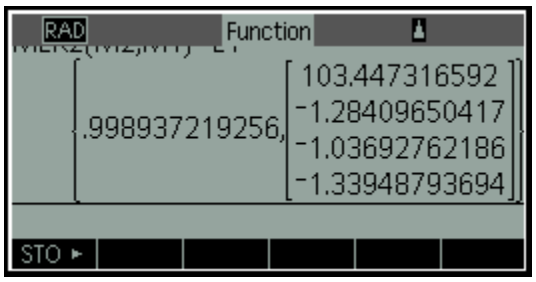

Fig. 4 – The results of executing function MLR2.

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.

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 a quadratic and cubic polynomials, respectively.

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`

Statement
`  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

x	*y*
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 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 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**.

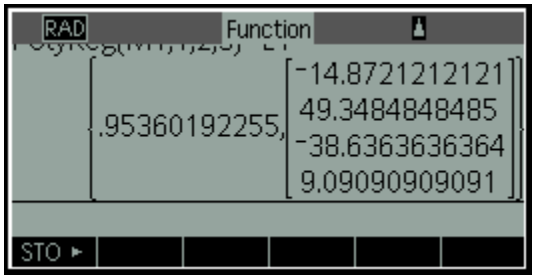

Fig. 5 – The results of executing function PolyReg.

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 matrix **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.

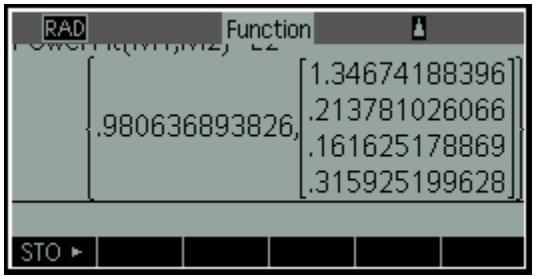

Fig. 6 – The results of executing function PowerFit.

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 do so. If not go back to step 3. In most cases, this step goes without any problems.
12. 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.
13. Connect your PC to the physical HP 39gII calculator using a USB cable.
14. Select and copy the function from the emulator to the calculator by using the SEND command (available in Prgm mode).
15. 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. 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).

