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 $e^{\text{intercept}}$.

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, an 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, an 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;

Statement
`    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

x	*y*
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
```

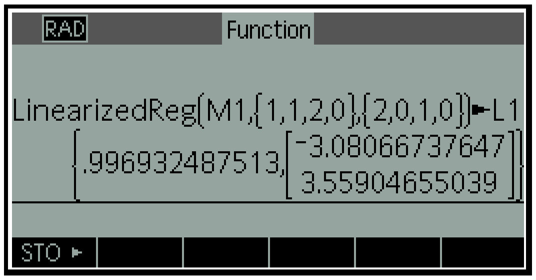
The first argument of calling function **LinearizedReg** is the matrix M1 which contains the (x, y) data points. The second argument is the list {1,1,2,0}. This list tells the function **LinearizedReg** that values for variable x are in the first column of matrix M1. The remaining list elements tell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



```
RAD          Function

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

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, an value to shift the values of x, an 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, an value to shift the values of z, an 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, an value to shift the values of y, an 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);`

Statement
`// 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`
`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`

Statement
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 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:

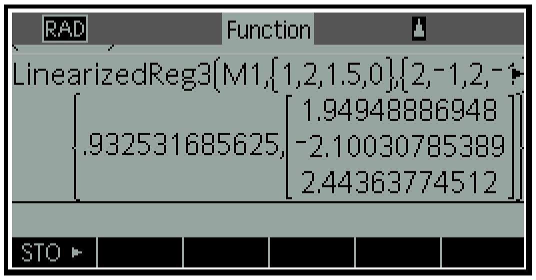
z = 1/(2*z – 1)

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►
   [                [ 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+1) + 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. The values of the regression coefficients a, b, and c are 1.949488869, – 2.100307854 , and 2.443637745, respectively.

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 the 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 store the transformed values in **MatX**. Otherwise, it stores the transformed values in **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`

Statement
`lstDimT:=SIZE(TrnfMat);`
`NumTrnf:=lstDimT(1);`
`// create the data matrices`
`MatX:=MAKEMAT(1,NumRowsX,MaxTerms+1);`
`VectY:=MAKEMAT(1,NumRowsX,1);`
`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`

Statement
`  RegCoeff:=LSQ(MatX,VectY);`
`  // calculate ymean`
`  Sum1:=0;`
`  FOR i FROM 1 TO NumRowsX DO`
`    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

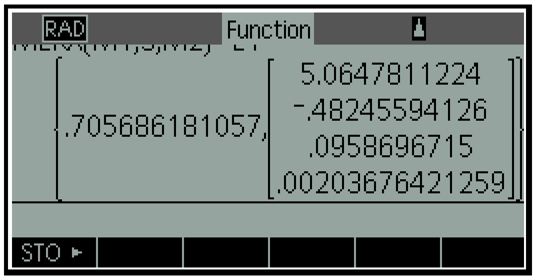
Select Variable	*Scale Value*	*Shift Value*	*Power*	*Target Matrix*	*Target Matrix Column*
2	3	5	0	1	3
3	5	–2	2	1	4

Table 7 – Transformations matrix.

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

`MLRX(M1,3,M3)→L1`

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



```
{.705686181057,[[5.0647811224
                 -.48245594126
                 .0958696715
                 .00203676421259]]}
```

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

Select Variable	*Scale Value*	*Shift Value*	*Power*	*Target Matrix*	*Target Matrix Column*
4	1	0	0	0	0

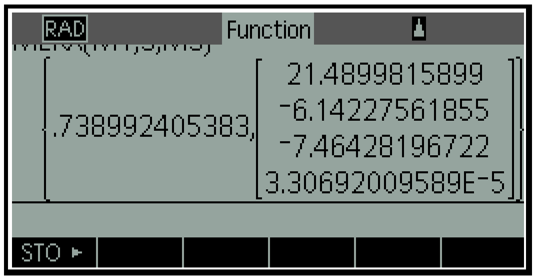
Select Variable	*Scale Value*	*Shift Value*	*Power*	*Target Matrix*	*Target Matrix Column*
1	5	–2	1	1	2
1	2	1	–1	1	2
2	3	5	0	1	3
1	1	1	–1	1	3
1	1	1	1	1	4
2	1	3	1	1	4
3	5	–2	2	1	4

Table 8 – The second transformations matrix.

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**.



```
MLRX(M1,3,M3)
{.738992405383,[[21.4899815899]
                [-6.14227561855]
                [-7.46428196722]
                [3.30692009589E-5]]}
```

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).