NLREG COM Interface

Phillip H. Sherrod
6430 Annandale Cove
Brentwood, TN 37027-6313 USA
phil.sherrod@sandh.com

www.nlreg.com

Contents

Introduction

The NLREG COM (Component Object Model) library is designed to make it easy for production applications to call on NLREG as an "engine" to perform nonlinear regressions. Entry points are provided to pass in a NLREG source program, check for compile errors, pass in data values, perform the regression and fetch computed parameter and statistic values. The NLREG COM library also can compute predicted values of the dependent variable once a function has been fitted.

Because of the standardization of the COM interface, it is easy to call NLREG from programs written in Visual Basic, Visual C++, VBA, Excel, Access, ASP and other languages. The NLREG COM library is designed to run as an in-process DLL for speed of execution.

Registering the NLREG COM Library

Before you can use the NLREG COM library you must register it so that Windows is aware that it exists and knows where it is located. To do this, get to the command prompt in Windows and then change directory to the directory where NLREGCOM.DLL resides. Then issue the following command:

```
REGSVR32 NLREGCOM.DLL
```

A window will be displayed confirming that NLREGCOM.DLL has been registered.

Data Types Used With the NLREG COM Methods

The descriptions of the NLREG methods show the recommended data types for the input and output arguments. For example, the SetDataValue method has three arguments:

SetDataValue(*inRow* As Long, *inCol* As Long, *inValue* As Double)

The recommended type for the *inRow* and *inCol* arguments is Long and the recommended type for the *inValue* argument is Double.

However, internally NLREG accepts all of the incoming arguments as VARIANT types and converts them to the appropriate types before using them. So all of the following calls to SetDavaValue will work:

SetDataValue(1, 2, 4.0)
SetDataValue(1.0, 2.0, 4)
SetDataValue("1", 2, "4.5")

Output values from NLREG are returned as VARIANTs with the appropriate type specified. The application language will convert them into the type required to match the variables you specify to receive them.

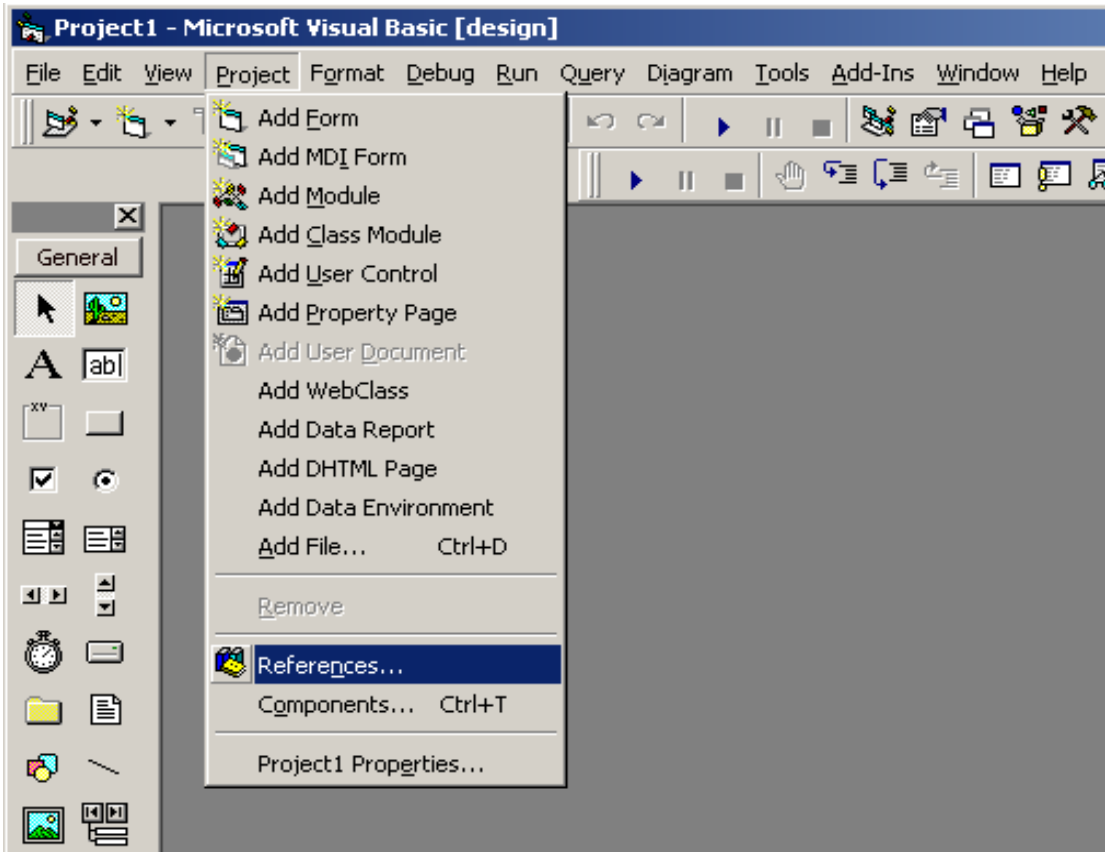
This dynamic data type conversion makes it easy to call NLREG methods from languages such as ASP which do not have data type declarations.

Calling NLREG from Visual Basic

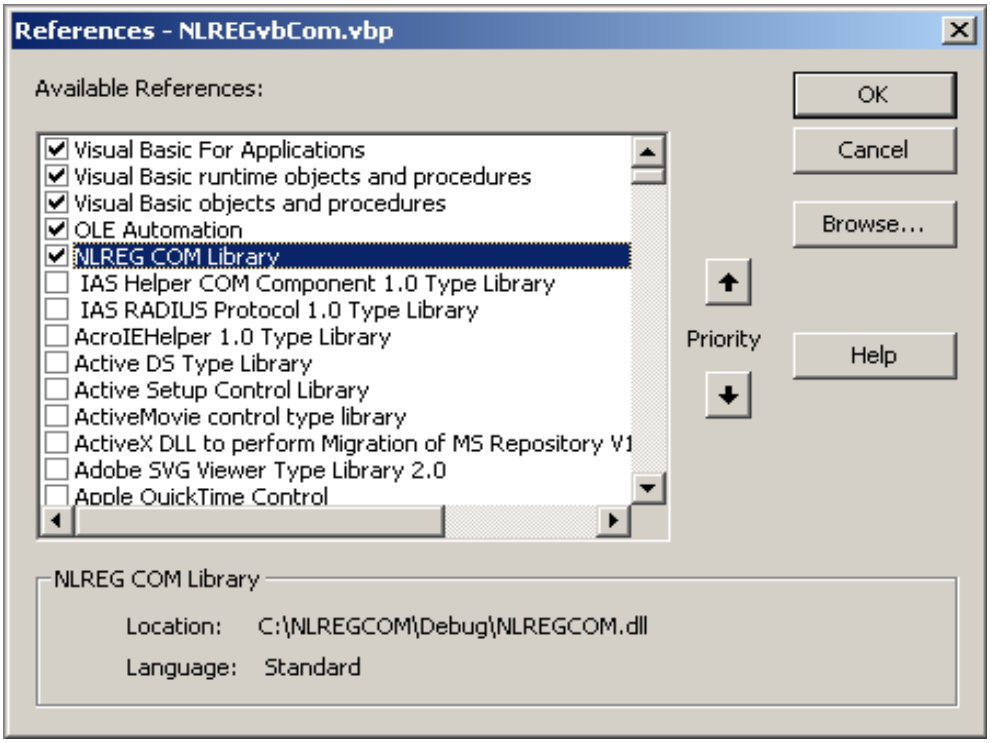
It is easy to call NLREG methods from Visual Basic because Visual Basic has excellent support for COM objects.

Referencing NLREG from Visual Basic

The first step in using NLREG with a Visual Basic program is to tell Visual Basic to reference NLREG. To do this, start Visual Basic, then click "Project" on its main menu, then click "References…" on the drop-down menu.



A popup screen will appear with a list of the registered COM objects. Search through the list and check the box next to "NLREG COM Library". Then click "OK" to exit from the references screen.



If the "NLREG COM Library" entry doesn't appear in the references list, then the NLREG COM library has not been properly registered, and you should perform the registration procedure as described above.

Once you have established a reference to NLREG within Visual Basic, you can add declarations and method calls to your Basic programs.

Declaring the NLREG COM object in a Visual Basic program

Each Visual Basic procedure that calls a NLREG method must contain the following declarations:

```
Dim nlreg As NLREGCOMLib.NLREG
Set nlreg = New NLREGCOMLib.NLREG
```

This declares a variable named "nlreg" to reference the NLREG COM library. Once you do that, you can then reference NLREG methods by typing "nlreg.*methodname*()". (You do not have to name the variable "nlreg", but it is recommended that you do so to make it clear that the methods are part of the NLREG COM library.)

Visual Basic Example Program

```
Private Sub Command1_Click()
'
'  Reference the NLREG COM library
'
Dim nlreg As NLREGCOMLib.nlreg
Set nlreg = New NLREGCOMLib.nlreg
'
'  Misc. data declarations.
'
Dim ProgramSource As String
Dim Report As String
Dim VarName As String
Dim Status As Long
Dim NumVar As Long
Dim Xindex, Yindex As Long
Dim Aindex, Bindex As Long
Dim Avalue, Bvalue As Double
Dim Index As Long
Dim PredictedY, Rsquared As Double
'
'  Initialize the NLREG library.
'
nlreg.Initialize (0)
'
'  Compile the NLREG program.
'  Note: If the program is in a file, use CompileFile().
'
ProgramSource = "Variables x,y; Parameters a,b; Function y=a+b*x;
data;"
Status = nlreg.Compile(ProgramSource)
If (Status <> 1) Then
    ' Some sort of compile error occurred.
    Report = nlreg.GetAnalysisReport(0)
    Stop
End If
'
'  Check information about the variables.
'
NumVar = nlreg.NumInputVariables()
For Index = 0 To NumVar - 1 Step 1
    VarName = nlreg.InputVariableName(Index)
Next
'
'  Get index numbers for the input variables.
'
Xindex = nlreg.InputVariableFind("x")
Yindex = nlreg.InputVariableFind("y")
```

```
'
'  Provide a set of x,y data values.
'
Status = nlreg.SetDataRows(5)
Status = nlreg.SetDataValue(0, Xindex, 1#)
Status = nlreg.SetDataValue(0, Yindex, 3#)
Status = nlreg.SetDataValue(1, Xindex, 2#)
Status = nlreg.SetDataValue(1, Yindex, 5#)
Status = nlreg.SetDataValue(2, Xindex, 3#)
Status = nlreg.SetDataValue(2, Yindex, 7.1)
Status = nlreg.SetDataValue(3, Xindex, 4#)
Status = nlreg.SetDataValue(3, Yindex, 8.5)
Status = nlreg.SetDataValue(4, Xindex, 5#)
Status = nlreg.SetDataValue(4, Yindex, 11.2)
'
'  Perform the regression analysis.
'
Status = nlreg.Compute()
If (Status < 0) Then
    ' Some sort of error occurred during the analysis.
    Report = nlreg.GetAnalysisReport(0)
    Stop
End If
'
'  Get the report from the analysis.
'
Report = nlreg.GetAnalysisReport(0)
'
'  Get the computed values of the 'a' and 'b' parameters.
'
Aindex = nlreg.ParameterFind("a")
Bindex = nlreg.ParameterFind("b")
Avalue = nlreg.ParameterValue(Aindex, 0)
Bvalue = nlreg.ParameterValue(Bindex, 0)
'
'  Use the fitted function to predict some y values given x values.
'
Status = nlreg.SetEvaluationValue(Xindex, 5#)
PredictedY = nlreg.EvaluateFunction(0)
Status = nlreg.SetEvaluationValue(Xindex, 6.5)
PredictedY = nlreg.EvaluateFunction(0)
'
'  Get the proportion of variance explained (R^2) for the model.
'
Rsquared = nlreg.GetStatistic(6)
'
'  End of program
'
End Sub
```

Calling NLREG from ASP Scripts

It is easy to call NLREG methods from ASP (Active Server Page) scripts. You must declare an object to reference the NLREG COM library. This is done by using the following statement:

```
set nlreg = server.createobject("nlregcom.nlreg")
```

After doing that, you can invoke NLREG methods by specifying nlreg.*method*() in the script. Note, unlike Visual Basic, there is no way to set up a "Reference" to the NLREG COM library in ASP scripts. Because of this, if you mistype the name of a method or specify an incorrect number of arguments, the error will not be detected until the ASP page attempts to execute the statement with the error.

Here is a complete ASP script that compiles a NLREG program stored in the file c:\test\test.nlr, computes the regression and then displays the names of the two parameters and their computed values:

```
<HTML>
<%
        set nlreg = server.createobject("nlregcom.nlreg")
        nlreg.Initialize(0)
        ival = nlreg.CompileFile("c:\test\test.nlr")
        response.write "Compile result = " & ival & "<p>"
        nlreg.Compute
        result = nlreg.ParameterName(0) & " = " & nlreg.ParameterValue(0,0) & "<p>"
        response.write result
        result = nlreg.ParameterName(1) & " = " & nlreg.ParameterValue(1,0) & "<p>"
        response.write result
%>
</HTML>
```

Methods in the NLREG COM Library

Initialize() -- Initialize for a new analysis

Function Initialize (*inDummy* As Long) As Long

The Initialize()method resets NLREG for a new program. It should be called before starting a new analysis involving a new source program. You should not call it if you are changing the data values and re-computing using the same source program.

The *inDummy* argument is currently not used but is reserved for future use. You should specify a value of 0 (zero) for it. The method always returns the value 0.

Compile() -- Compile a NLREG program

Function Compile (*inSource* As String) As Long

The Compile() method accepts a NLREG source program, compiles it and returns a status code indicating if there were any compilation errors.

Arguments:

inSource = A string containing the NLREG source program. As usual, source statements are separated by semicolon (';') characters. You may place carriage-return and line-feed characters between the statements, but it is not necessary. See the *NLREG – Nonlinear Regression Analysis Program* manual for detailed information about writing NLREG programs.

There are three ways to provide the data values for the program:

1. You can pass the data records as part of the source program. In this case, the source program should end with a DATA statement of the form:

   DATA;

   and the data records should immediately follow it. You may separate the data records either with carriage-return/line-feed characters or by semicolons.

2. You can place the data in an external file. In this case the ending statement must be:

   DATA "*filename*";

Where *filename* is the full specification of the file with the data records. Note: you should specify the directory along with the file name (for example, "c:\work\test.dat").

3. You can pass in the data values using the SetDataValue() method described below. In this case, the program must end with the following statement:

   DATA;

   and no data records should follow it.

Returned value:

The value 1 is returned by Compile() if the compilation was successful. A value of 0 (zero) is returned if there were compile errors. You can use the GetAnalysisReport() method to obtain a listing of the compile errors.

CompileFile() -- Compile a NLREG program from a file

Function CompileFile (*inFilespec* As String) As Long

The CompileFile() method reads a NLREG source program from a file, compiles it and returns a status code indicating if there were any compile errors. The operation of this method is identical to Compile() except that you specify the name of a file containing the NLREG program rather than passing the source program in as string variable.

Arguments:

inFilespec = A string containing the specification for the file containing the NLREG source program. You should specify the directory as well as the file name (for example, "c:\work\test.nlr").

See the description of the Compile() method for information about returned values and methods for specifying data values.

GetAnalysisReport() -- Get compiler and analysis report

Function GetAnalysisReport (*inDummy* As Long) As String

This method returns a string that is either (a) a list of the compile errors if Compile() returned value of 0, or (b) a listing of the results of the NLREG analysis run if the compilation was successful and Compute() was called.

The *inDummy* argument is reserved for future use. You should specify 0 (zero) for its value.

Compute() -- Perform a regression analysis

Function Compute () As Long

The Compute() method is called after Compile() or CompileFile() to perform the regression analysis. If you are running multiple analyses using the same NLREG source program and different sets of data values, you can call Compile() once and then call Compute() repeatedly after providing each set of data values.

The following values are returned by Compute():

6 = Both parameter and relative function convergence
5 = Parameter convergence
4 = Relative function convergence
3 = Absolute function convergence
2 = Singular convergence. (Possibly mutually dependent parameters)
1 = False convergence. (Answers may not be correct.)
-1 = Function did not converge before iteration limit reached
-2 = Function cannot be computed at starting parameter values
-3 = Bad parameter values
-4 = Jacobian could not be computed
-5 = Singular matrix. Possibly mutually dependent parameters
-6 = Function did not converge before max allowed function calls
-7 = There are fewer data observations than parameters.

NumInputVariables() -- Get number of input variables

Function NumInputVariables () As Long

This method returns a count of the number of input variables that were specified in the NLREG program that was last compiled. For example, if the program contained the declaration:

Variables x,y,z;

Then NumInputVariables() would return the value 3.

InputVariableName() -- Get the name of an input variable

Function InputVariableName (*inIndex* As Long) As String

The InputVariableName() method returns the name of an input variable. The *inIndex* argument is a 0-based index to select which variable name you want. The variables are numbered in the order in which they are specified on VARIABLES statements in the NLREG program. When used with the NumInputVariables() method, you can easily write a program to determine how many input variables were specified and display the names of each variable.

InputVariableFind() -- Get the index of an input variable

Function InputVariableFind (*inVariableName* As String) As Long

The InputVariableFind() method looks up the name of an input variable (i.e., a variable declared using the VARIABLES statement) and returns the index number of the variable. The variables are numbered based on the order that they are specified on VARIABLE statements in the NLREG program; the first variable has an index number of 0 (zero). If the specified variable name cannot be found, an index number of -1 is returned.

InputVariableValue() -- Get a statistic for an input variable

Function InputVariableValue (*inIndex* As Long, *inValtype* As Long) As Double

The InputVariableValue() method returns a statistic for an input variable. The *inIndex* argument is a 0-based index to select which variable you are getting values for. The variables are numbered in the order that they are declared on VARIABLE statements in the NLREG program. You can use the InputVariableFind() method to get the index number for a specific variable.

You must invoke the Compute() method before using InputVariableValue().

The *InValtype* argument selects which type of value you want for the variable. The following values may be specified for *InValtype*:

0 (VARVAL_MIN) = Minimum value of the variable
1 (VARVAL_MAX) = Maximum value of the variable
2 (VARVAL_MEAN) = Mean value of the variable
3 (VARVAL_STDEV) = Standard deviation of the variable

NumParameters() -- Get number of computed parameters

Function NumParameters () As Long

This method returns a count of the number of parameters that were specified in the NLREG program. Parameters are declared using the PARAMETERS statement(s) in NLREG programs.

ParameterName() -- Get the name of a computed parameter

Function ParameterName (*inIndex* As Long) As String

The ParameterName() method returns the name of a parameter. The *inIndex* argument is a 0-based index to select which parameter name you want. The parameters are numbered based on the order that they are specified on PARAMETER statements in the NLREG program.

ParameterFind() -- Get the index of a computed parameter

Function ParameterFind (*inParameterName* As String) As Long

The ParameterFind() method looks up the name of a parameter and returns the index number of the parameter. The parameters are numbered based on the order that they are specified on PARAMETER statements in the NLREG program; the first parameter has an index number of 0 (zero). If the specified parameter name cannot be found, an index number of -1 is returned.

ParameterValue() -- Get the computed value for a parameter

Function ParameterValue (*inIndex* As Long, *inValtype* As Long) As Double

The ParameterValue() method returns a value for a computed parameter. The *inIndex* argument is a 0-based index to select which parameter you are getting values for. The parameters are numbered in the order that they are declared on PARAMETER statements in the NLREG program. You can use the ParameterFind() method to get the index number for a specific parameter.

You must invoke the Compute() method before using ParameterValue().

The *InValtype* argument selects which type of value you want for the parameter. The following values may be specified for *InValtype*:

0 (PARVAL_ESTIMATE) = Computed estimate for the parameter
1 (PARVAL_STDERR) = Standard error of the estimate
2 (PARVAL_TVALUE) = t value for the computed parameter value
3 (PARVAL_PROBT) = Probability of the t value (Prob(t))
4 (PARVAL_CONFLOW) = Lower value of the confidence interval
5 (PARVAL_CONFHI) = Upper value of the confidence interval
6 (PARVAL_INITIAL) = Initial value that was specified for the parameter

SetParameterValue() -- Set value for a parameter

Function SetParameterValue (*inIndex* As Long, *inValtype* As Long, *inValue* As Double) As Long

The SetParameterValue() method sets a value for a parameter.

There are two situations where this function is useful:

1. You can set the initial (starting) value of a parameter prior to calling Compute() to fit the model to the data. In this case, SetParameterValue() should be called after Compile() or CompileFile() and before Compute().

2. You can set the value for a parameter before calling EvaluateFunction() if you want to use a specified parameter rather than the computed parameter value when evaluating the value of a function. In this case, SetParameterValue() must be called after Compute() and before EvaluateFunction().

The *inIndex* argument is a 0-based index to select which parameter you are setting values for. The parameters are numbered in the order that they are declared on PARAMETER statements in the NLREG program.

The *InValtype* argument selects which type of value you are setting for the parameter. The following values may be specified for *InValtype*:

The *InValue* argument is the value to which the parameter is to be set.

0 (PARSET_INITIAL) = Initial value assigned to parameter at beginning of computation

If the method executes successfully, it returns a value of 1. If an error occurs while executing the method, it returns a value of 0. If an error occurs, check the parameter index and the value type index.

SetDataRows() and SetDataValue() -- Set input data values

Function SetDataRows (*inNumrows* As Long) As Long

Function SetDataValue (*inRow* As Long, *inCol* As Long, *inValue* As Double) As Long

There are three methods for specifying sets of data values for an analysis:

1. You can end your NLREG source program with a Data; statement and specify the data records starting with the next line of the program.
2. You can end your NLREG source program with a Data "*filename*"; statement and put the data in an external file.
3. You can use the SetDataRows() and SetDataValue() methods to specify each element of the data array.

The SetDataRows() method specifies how many rows (observation records) of data there are. It must be called after you call the Compile() method and before you call SetDataValue().

After you have called SetDataRows() to specify how many rows of data there are, you can call SetDataValue() to provide a value for each element of the data array. The *inRow* argument specifies the row index; it must be in the range from 0 to *numrows*-1. The *inCol* argument specifies the column index; it must be in the range from 0 to *numvar*-1 where *numvar* is the number of input variables in the program. The *inValue* argument is the data value to be set in the (*row,col*) element of the data array.

The following values are returned by SetDataValue():

0 (RVSDA_OK) = Success
4 (RVSDA_NOMEMORY) = Unable to allocate memory for the array
5 (RVSDA_NOPROGRAM) = No program has been specified yet
6 (RVSDA_BADROW) = Invalid row index number
7 (RVSDA_BADCOL) = Invalid column index number

If you call Compile() to compile a new NLREG program, you must specify a new set of data values before calling Compute().

Here is a sample code fragment showing how SetDataRows() and SetDataValue() might be used to set the data values for 2 variables and 4 rows:

```
Sub Setdata_Click()
Dim result As Long
‘  Specify that there are 4 data rows.
nlreg.SetDataRows (4)
‘  Specify the data value for 2 rows and 2 columns (variables).
result = nlreg.SetDataValue(0, 0, 1.1)
result = nlreg.SetDataValue(0, 1, 2.1)
result = nlreg.SetDataValue(1, 0, 2.2)
result = nlreg.SetDataValue(1, 1, 3.9)
result = nlreg.SetDataValue(2, 0, 3.1)
result = nlreg.SetDataValue(2, 1, 6.2)
result = nlreg.SetDataValue(3, 0, 4.3)
result = nlreg.SetDataValue(3, 1, 7.8)
```

NumDataRows() – Get number of data rows (observations)

Function NumDataRows (*inDummy* As Long) As Long

The NumDataRows() method returns the number of data rows (observations).

GetDataValue() -- Get the value of an input variable

Function GetDataValue (*inRow* As Long, *inCol* As Long) As Double

The GetDataValue() method returns the value of a specified row and column position in the data matrix. The *inRow* argument specifies the row (observation) number and must range from 0 to *numrows*-1. The *inCol* argument specifies which input variable the data value corresponds to and must be in the range 0 to *numvar*-1. Variables are numbered in the order that they are specified on VARIABLE statements in the NLREG program. GetDataValue() is the compliment of the SetDataValue() method; it retrieves a value from the data matrix whereas SetDataValue() sets a value in the data matrix.

GetStatistic() -- Get a computed statistic value

Function GetStatistic (*inValtype* As Long) As Double

This method can be called after Compute() to obtain the values of statistics computed for the function.

The *inValtype* argument specifies which statistic value is to be returned by the function. The following values for *inValtype* may be specified:

0 (GSX_NUMOBS) = Number of observations
1 (GSX_ITERATIONS) = Number of iterations required to obtain convergence
2 (GSX_RESIDUAL) = Sum of squared deviations (residuals)
3 (GSX_SUMDEV) = Sum of deviations
4 (GSX_MAXDEV) = Maximum deviation for any observation
5 (GSX_STDERR) = Standard error of the estimate
6 (GSX_RSQUARED) = Proportion of variance explained (R^2)
7 (GSX_ADJRSQUARED) = Adjusted coefficient of multiple determination
8 (GSX_DURBINWATSON) = Value of Durbin-Watson test for autocorrelation
9 (GSX_RUNTIME) = Number of seconds to perform analysis
10 (GSX_NLREGVERSION) = Version of NLREG being used
11 (GSX_AVRDEV) = Average deviations for the observations

GetCorrelationValue() -- Get a correlation value

Function GetCorrelationValue (*inRow* As Long, *inCol* As Long) As Double

If the Correlate statement is included in the NLREG program, NLREG will compute a correlation matrix for the input variables. If you specify the Correlate statement in the program without any variable names, like this:

```
variables x,y,z;
correlate;
```

then all of the variables (x, y and z in this example) will be included in the correlation matrix and there will be a row and a column in the matrix for each one. The order of the rows and columns corresponds to the order in which the variables are declared.

If you specify a set of variables with the Correlate statement, like this:

```
variables x,y,z;
correlate x,y;
```

Then only the specified variables are included in the correlation matrix, the matrix has a row and column for each variable and the order of the rows and columns corresponds to the order that the variables are specified with the Correlate statement.

Since the correlation of a variable with itself is always 1.00, the diagonal elements of the correlation matrix will always have the value 1.00.

The GetCorrelationValue() method retrieves a value from a specified row and column of the correlation matrix. The *inRow* argument specifies the row number and it must range from 0 up to *numvariables*-1 where *numvariables* is the number of variables in the

correlation matrix. The *inCol* argument is the column number index; it has the same range as *inRow* since the matrix is square.

Here is a program fragment that obtains the correlation matrix and writes its values to a file:

```
Dim nlreg As NLREGCOMLib.nlreg
Set nlreg = New NLREGCOMLib.nlreg
Dim rowcol As Long
Dim row As Long
Dim col As Long
…
Open "correlation.txt" For Output As #1   ' Open file for output.
For row = 0 To rowcol - 1
   For col = 0 To rowcol - 1
      Print #1, Format(nlreg.GetCorrelationValue(row, col), "   ###.####");
   Next
   Print #1,
Next
Close #1
```

GetCovarianceValue() -- Get a covariance value

Function GetCovarianceValue (*inRow* As Long, *inCol* As Long) As Double

If the Covariance statement is included in the NLREG program, NLREG will compute a covariance matrix for the computed parameter values. There will be a row and a column in the matrix for each parameter. The order of the rows and columns corresponds to the order in which the parameters are declared.

For example, if the NLREG program contains the following declarations:

```
parameters a,b,c;
covariance;
```

Then the covariance matrix will have three rows and three columns.

The *inCol* argument to GetCovarianceValue() specifies the column of the covariance matrix and the *inRow* argument specifies the row. They must be in the range 0 to *numparam*-1 where *numparam* is the number of parameters.

Here is a program fragment that obtains the covariance matrix and writes its values to a file:

```
Dim nlreg As NLREGCOMLib.nlreg
Set nlreg = New NLREGCOMLib.nlreg
Dim row As Long
Dim col As Long
…
Open "covariance.txt" For Output As #1   ' Open file for output.
For row = 0 To rowcol - 1
   For col = 0 To rowcol - 1
      Print #1, Format(nlreg.GetCovarianceValue(row, col), "   ###.####");
   Next
   Print #1,
Next
Close #1
```

SetEvaluationValue() and EvaluateFunction() -- Evaluate the function

Function SetEvaluationValue (*inIndex* As Long, *inValue* As Double) As Long
Function EvaluateFunction (*inDummy* As Long) As Double

The EvaluateFunction() method evaluates the current NLREG program function with a specified set of input variable values and the computed parameter values. It must be called after Compute() has successfully completed the analysis. What EvaluateFunction() does is "plug in" a set of input variable values and determine what the corresponding value of the function is using the parameter values computed by the regression analysis.

There are a number of situations where EvaluateFunction() can be useful. For example, if you want to compare the actual values of the dependent variable with the estimates generated by the fitted function, iterate through each observation, get the values of the independent variables and use SetEvaluationValue() to set the values, then call EvaluateFunction() to compute the estimated value of the function. If you want to generate a plot of the fitted function, iterate in small steps across the range of dependent variable values you want to plot and call EvaluateFunction() to get the function value for the plot.

The SetEvaluationValue() method must be called to set the values of the input variables before calling EvaluateFunction(). The *inIndex* argument to SetEvaluationValue() is a 0-based index that specifies which variable is having its value set, and the *inValue* argument specifies the value to be set. The value returned by SetEvaluationValue() is 0 if the operation is successful; a non-zero error code is returned if an error occurs.

SetEvaluationValue() must be called once for each input variable. The index numbers for the variables correspond to the order in which the variables were declared on the variable statement(s) in the NLREG program. Note: One of the input variables is the dependent

variable (i.e., the variable on the left side of the equal sign in the function). Since the purpose of the EvaluateFunction() method is to compute the value of the dependent variable, there is no need to call SetEvaluationValue() to set the value of the independent variable. However, you still must count the dependent variable when determining the variable index value to specify for *inIndex*.

The following values are returned by SetEvaluationValue():

0 (RVSDA_OK) = Success
12 (RVSDA_BADVAR) = Invalid variable index specified for *inIndex.*

Here is an example code fragment that evaluates the function at 101 points varying the value of the independent variable from 0 to 100:

```
Dim nlreg As NLREGCOMLib.nlreg
Set nlreg = New NLREGCOMLib.nlreg
Dim funval As Double
Dim i As Long
Dim result As Long
…
‘  Compute the regression.
result = nlreg.Compute()
‘  Evaluate the function over a range 0 to 100 for variable 1.
For i = 0 To 100
        result = nlreg.SetEvaluationValue(1,i)
        funval = nlreg.EvaluateFunction(0)
        …
Next
```

Index