Excel Translator™

User Guide

Copyright © 2006. Ultimate Risk Solutions

Excel Translator™ User's Guide

Contents

1 ABOUT *EXCEL TRANSLATOR*™ 1

1.1. Benefits 1

2 SYSTEM REQUIREMENTS 4

Minimum Configuration 4

3 INSTALLATION 5

4 *EXCEL TRANSLATOR*™ CONCEPTS 6

5 USING *EXCEL TRANSLATOR* ™ 9

5.1 Defining Input and Output Variables 9
5.2 Using VBA Macros in Excel Models 9
5.3 Using External Add-Ins in Excel Models 10
5.4 Translating Excel Spreadsheets 11
5.5 Using Translated DLL in a VBA Macro – Examples and Sample Files 11
5.6 Using Translated DLL in C++ Program – Examples and Sample Files 14
5.7 Using External DLL Function Calls in Spreadsheet Models 15
5.8 Testing Translated DLL 16

6 *URS MODEL BUILDER* AND *RESULT VIEWER* 19

7 USING URS STATISTICAL DISTRIBUTION AND MATH LIBRARY 21

8. STATISTICAL DISTRIBUTIONS OF *EXCEL TRANSLATOR*™ 28

8.1 Continuous Distributions 28
8.2 Discrete Distributions 36

1 About *Excel Translator™*

When used in Microsoft Windows environment, *Excel Translator*™ compiles Excel spreadsheet files into Windows machine code Dynamic Link Libraries (DLLs). The translated DLL preserves the entire logic of the calculations of an Excel spreadsheet, but executes (calculates) significantly faster than the original Excel spreadsheet.

Such DLLs can be used by other programs for fast dynamic processing. Such programs can be written in any programming language utilizing the DLL standard, such as Microsoft Visual Basic or C, C++, C#, or other.

1.1. Benefits

The technology behind the *Excel Translator*™ is pending US patent and offers many important benefits:

1) *Fast execution:* The fast execution speed of the model converted with the Excel Translator™ into machine code is just one among many important advantages offered by this product.
2) *Saving time and money:* Instead of exerting a significant effort in programming and software engineering, the owner of Excel Translator™ software translates any Excel-based model, with all of its VBA and associated DLL's, at a click of a mouse button. By investing into Excel Translator™, its users save not only the trouble and time of programming, but the associated expenses of program development – often quite significant.
3) *Hiding the proprietary algorithms* from prying eyes of competitors: Thus produced machine code conveniently hides the inner workings of the model, protecting its proprietary nature – an important feature in many applications in finance, engineering and science.
4) *Preserving integrity of the model:* model developers do not need to worry that users would inadvertently modify its algorithm or "break" the model while entering data.
5) *Expanded Math Library:* In addition, Excel Translator offers its unique Statistical Distribution and Math Library, which expands on Excel's rich development environment and enables creation of sophisticated financial, statistical and other models.
6) *Model Builder*: Model Builder is a component which facilitates building sophisticated models out of previously created DLLs, using them effectively as building blocks to create complex models, while using a light, elegant architecture. Model Builder is a component of the full developer license of Excel Translator™ or it can be can be purchased separately.

A typical application of the *Excel Translator*™ is a complex Excel model that needs to be recalculated many times, perhaps tens of thousands times, each time with a different set of input parameters. For each set of input parameters, the entire model is recalculated to produce corresponding output parameters. The combined effect of a translated DLL running much faster than its Excel prototype, plus a large number of required iterations, may result in a dramatically improved speed of model's calculation.

Excel Translator™ produces code which delivers the same numeric results as the original Excel model, only orders of magnitude faster. Usually, the more complicated the Excel model, the more time it takes to calculate, the more dramatic is the increase in speed demonstrated by the machine code produced with Excel Translator.

There are many reasons why organizations and institutions convert the original or the prototype Excel model into a program. Such conversion is generally a tedious, lengthy and expensive programming project. Excel Translator ™ achieves the same results in seconds!

Without *Excel Translator,* modelers usually employ either or both of the following two options:

1. They write specialized software for their models in various programming languages, such as C/C++ or Visual Basic. This option offers an advantage in calculation speed, but its substantial drawbacks are:

    - *Rigidity of the program code,*
    - *High expense of its creation and maintenance,*
    - *Lengthy, tedious and laborious reprogramming effort every time the structure of the model changes.*

2. They create Excel spreadsheets to process the calculations of the models. The advantage of this approach is the powerful and rich modeling environment offered by Microsoft Excel with commercially available and proprietary Plug-Ins, and the flexibility and ease of changing the model structure. However, the speed of calculations is unacceptably slow, especially for complex models, and protection of the proprietary logic is weak.

Excel Translator ™ from *Ultimate Risk Solutions,* offers the best of both of these two approaches. It combines the high speed of execution offered only by executable programs, as well as the flexibility and ease-of-development offered by powerful Excel environment. In addition, Excel Translator™ extends Excel's capabilities by offering its users its own rich set of statistical distribution functions which far expand the capabilities of Microsoft Excel in this area (see the "Statistical Distributions in Excel Translator™" section of this manual for more information).

Excel Translator ™ comes with *Model Builder*, an auxiliary application that conveniently assembles and runs previously translated DLL components. Thus, it facilitates creation of complex models, built out of previously created or simpler ones, allowing you to use models as building blocks for other models.

Excel Translator ™ is an embodiment of the concept of "programming without programmers." Instead of using a tiger team of programming experts working hard (sometimes for weeks or months, or even longer) to convert an Excel model into machine code, with Excel Translator ™ you can achieve the same results, instantly.

2 System Requirements

Excel Translator is capable of compiling very large Excel spreadsheets. Naturally, it performs better on faster hardware, but it is capable of running on a relatively small system.

Minimum Configuration

Processor:	300 megahertz Pentium
RAM:	128 megabytes
Operating System:	Windows (95/98/2000/NT/XP)
Available disk storage:	20 megabytes

CD-ROM, CD-RW or CD/DVD drive

3 Installation

To install *Excel Translator™*, insert the *Excel Translator* CD into the CD drive, or download the installation kit from www.ExcelTranslator.com into a folder on your computer, for example, **C:\ET**.

If the Setup window does not appear automatically, use the following steps to start it:

- Left-click on the **Start** menu.
- Choose the **Run**… item.
- In the Open: field, type in the drive letter of the CD drive, colon, and the name of *Excel Translator* installation file, or specify the path to the installation file, for example:

  D:\ExcelTranslator-1-6-0(36).exe or

  C:\ET\ExcelTranslator-1-6-0(36).exe, if you downloaded Excel Translator™ distribution into **C:\ET.**

- Click **OK** and follow the instructions

Note that in order to produce Microsoft Windows DLLs, which is Excel Translator's standard functionality, Excel Translator™ must use a C++ compiler. Excel Translator™ uses a freely distributed Microsoft's Visual C++ compiler that can be downloaded from Microsoft website. Your computer should have a fast Internet connection in order to download and install Microsoft C++ compiler. You can also install a full function Excel Translator™ on a computer with no Internet connection, in which case you'd have to install a C++ compiler using a CD, flash disk or other distribution media.

Excel Translator™ installer wizard program helps you download and install a C++ compiler from Microsoft.

During the installation of Excel Translator™, the installation program asks if you wish to install a free Microsoft C++ compiler, and if you answer YES, the installation proceeds to the appropriate Micrsoft.com download page and helps you install the latest version of **Microsoft Visual C++ Express.** The current release of Excel Translator™ requires **Visual C++ 2005 Express** (or higher) version. It also supports an older Visual C++ 2003 compiler. Note that Microsoft's installation wizard and website may use names, such as "Visual C++ Express Edition," "Visual C++ Express," and "Visual C++ 2005 Express" interchangeably. They all refer to the same freely distributed C++ compiler from Microsoft.

If you need to use Excel Translator™ to generate C++ code in order, for example, to port the code to a different hardware or software platform, you may choose a different compiler suitable for that task.

4 *Excel Translator™* Concepts

In order to use *Excel Translator™*, certain cells in the Excel spreadsheet containing your model need to be identified as “Input Cells” or “Input Variables”. Certain other cells need to be defined as “Output Cells” or “Output Variables”. All remaining cells become “Logic Cells” or “Calculation Cells”, they contain values or Excel formulas. There are no restrictions on the number of Logic Cells or sheets that you can utilize in your Excel model (other than those restrictions imposed by Microsoft Excel.)

You can use the user-defined functions calling VBA macros in your logic cells. You can also use the functions from external add-ins, as well as from external DLLs declared in VBA modules.

The input cells are the “entry points” and the output cells are the “exit points” in your model as shown on Fig. 1 below.

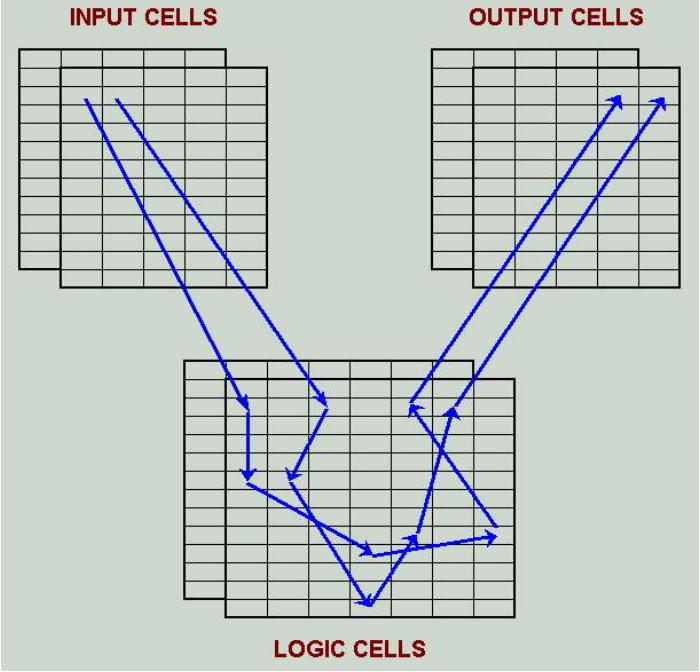

Fig. 1. Input / Output Cells and Logic Cells

After you define the input and output variables, you are ready to translate.
Excel Translator™ will compile your Excel spreadsheet into a DLL. Thus produced DLL will perform all the calculations defined in your Excel model and with the same numeric results, only now the calculations will be performed by the fast executing machine code.

The spreadsheet’s input cells become Input Parameters in the DLL and the output cells become Output Parameters. The DLL is a single function that:

- Takes the values of Input Parameters
- Runs all calculations specified in your Excel model
- Produces the values of Output Parameters

This function can be called in a fast repeating mode, for example 100,000 times (iterations) with different values of input parameters. Each time it will return a different set of calculated output values.

Input values can be produced by any external application written in VBA, C++, or any programming language which can utilize dynamic link libraries (DLLs). *Model Builder* supplied with *Excel Translator*™ is one of such external applications. Model Builder saves you the trouble of having to program an application whose function is to feed input parameters into the model.

Output values can be accumulated in the same external application to be subsequently analyzed. The DLL calculations execute much faster than the original Excel model. The speed of the program produced by *Excel Translator* ™ rivals that produced by expert programmers, making the time and expense of programming and the corresponding software development cycle unnecessary and obsolete.

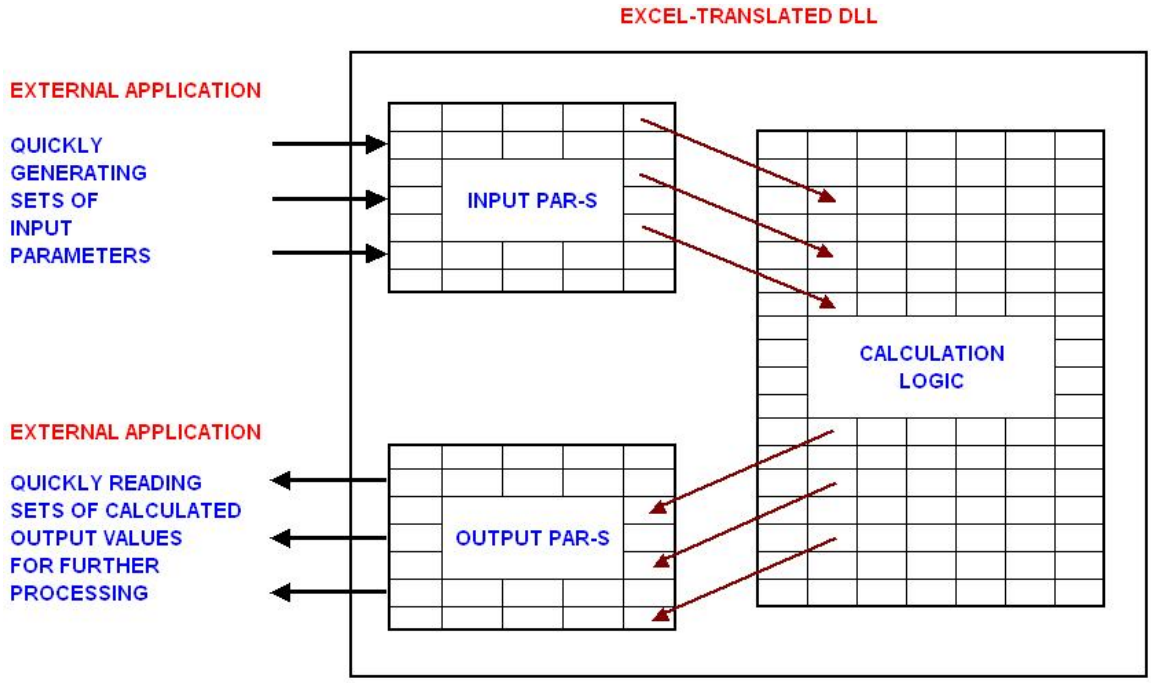

Fig. 2. Translated Model (DLL) and Feeding / Collection of Input / Output Parameters

NOTE: The original Excel model can contain any number of Excel formulas, user-defined functions calling VBA macros, functions from external Add-Ins, such as *Analysis Toolpack*, and/or functions from

additional external DLLs, which are declared in VBA modules and used in the Excel model.

5 Using *Excel Translator* ™

5.1 Defining Input and Output Variables

Open Excel spreadsheet containing your model. From Excel's main menu, select

Tools…Excel Translator…Define Input/Output.

On the first Wizard page, **Define Input Variables**, click the **Add** button to add each input variable. Then on the **Add Input Variable** dialog, specify the name and select the range of cells for that variable. You can subsequently edit or remove input variables. Click **Next** when you finished editing your specifications for input variables. On the subsequent Wizard page, **Define Output Variables**, create output variables in a similar way. Click **Finish** when done with defining output variables.

NOTE: *Excel Translator* creates a VBA module called **URS_VariableDefinition** in your spreadsheet that contains all the information about input and output variables. Do not edit this VBA module !

Also note that each input and output variable can be defined either as a single cell or as a range of cells (array or matrix).

5.2 Using VBA Macros in Excel Models

Generally, you can write your own user-defined functions and subroutines (macros) in Excel VBA as you normally would. *Excel Translator* ™ allows you to use VBA macros in your model. These functions and subroutines become part of the translated DLL after your Excel file is compiled by *Excel Translator*™.

Excel Translator's output is a machine code, which quite naturally cannot deal with Excel objects or program code which communicates with Excel. Indeed, the translated DLL is not using Excel, so such code in it would be meaningless. While there are virtually no limitations on how a model is created, please keep in mind the following rules:

- The lines of VBA code inside user-defined functions and/or subroutines that write data back to Excel will be ignored by *Excel Translator*™. Avoid writing such code or make sure that such code does not, in any way, affect your end result.

- *Excel Translator* ™ will ignore the lines of VBA code that use Excel objects, their methods and/or properties, therefore such code should also be avoided. The exceptions to this rule are:

- RANGE object when used as an argument of a function. For example, the following function will be valid for translation:

```
Function MyFunction(rMyInputRange As Range) As Double
   Dim i As Integer, dValue As Double

   For i = 1 To rMyInputRange.Count

      dValue = rMyInputRange(i)
      '
      ' other programming code
      '
   Next i
End Function
```

- RANGE object when used to read data from Excel's cells. For example, the following function will be translated:

```
Function MyFunction() As Double
   Dim i As Integer, adValues(10) As Double

   For i = 1 To 10

      adValues(i) = Sheets("Sheet1").Range("A1").Offset(i, 0).Value
      '
      ' other programming code
      '
   Next i
End Function
```

5.3 Using External Add-Ins in Excel Models

The functions from the external add-ins implemented as XLL files and registered in Excel, may be used in your Excel models. *Excel Translator*™ compiles (translates) such functions requiring no extra effort on your part.

Likewise, *Excel Translator*™ understands functions from COM Add-Ins (sometimes called Automation Add-Ins) translating them without any extra effort on your part.

NOTE: Avoid using external add-in's functions that make no sense in the context of a compiled model: functions that call methods or properties of Excel objects. *Excel Translator™* will ignore such calls to Excel objects and translating such functions may lead to an error.

5.4 Translating Excel Spreadsheets

You can translate an Excel file in three different ways:

- By selecting **Tools…Excel Translator…Translate** item from Excel's main menu
- By selecting **Excel Translator** item from **Start→ Programs→ Excel Translator** group and specifying the Excel file(s) you want to translate
- By right clicking on the Excel file in *Windows Explorer* and selecting the **Translate** menu item.

NOTE: The **Excel Translator** item will appear under **Tools** menu in Excel after you install *Excel Translator™*. If the **Excel Translator** item is not there, select **Tools…Add-Ins** and make sure that **Excel Translator** is on the list of add-ins and that it's checked. If **Excel Translator** is not on the list of add-ins, click on the **Browse** button and find the **ExcelTranslator.xla** file in "**C:\Program Files\Ultimate Risk Solutions\Excel Translator\AddIn**" folder (note that you may have chosen a different folder during the installation).

5.5 Using Translated DLL in a VBA Macro – Examples and Sample Files

The sample files for using translated DLLs from VBA code are installed into **C:\Program Files\Ultimate Risk Solutions\Excel Translator\Samples\VBA** folder. These files represent examples, which are intentionally simplified to be instructional while demonstrating how to use the product effectively. The real models you create can be as complex as you need them to be, but the process of defining input/output, translating the file, and running translated DLL from external application will still be the same, as demonstrated by the examples in the sample files.

The **SimpleModel.xls** file contains a simple Excel model that calculates the volume and the area of a parallelepiped. There is one input variable there, which is a (1 x 3) range where the elements of the vector are the lengths of the sides of a parallelepiped, and two output variables – volume and area, both single cells.

NOTE: The purpose of this example is to show how to define the input and output variables in your Excel model, how to translate Excel spreadsheet containing your model, and how to use translated DLL from VBA code in another Excel file.

Select **Tools…Excel Translator…Define Input/Output** from Excel's main menu to see how the input and output variables are defined in that spreadsheet. Then translate **SimpleModel.xls** by selecting **Tools…Excel Translator…Translate**. *Excel Translator™* creates the following files as a result of translation:

- **SimpleModel.dll** - a dynamic link library containing all calculations defined in the original Excel model;
- **SimpleModel.utd** - an auxiliary file used by the translated DLL;
- **SimpleModel.Excel.test** - an auxiliary file used during DLL testing.

NOTE: **DLL, UTD** and **EXCEL.TEST** files must always be kept in the same folder.

The **SimpleModelUser.xls** file shows you how to use the DLL translated from **SimpleModel.xls**. The **SimpleModelUser.xls** file has a VBA macro that calculates the values of parallelepiped's volume and area for different lengths of sides A, B, and C. You can control the number of iterations by changing the values on the **Data** sheet in **SimpleModelUser.xls.**

Look at the VBA code in **SimpleModelUser.xls**. Using translated DLL is easy. First, you need to reference *Excel Translator* in the Microsoft Visual Basic environment. In order to do that, select **Tools…References** from the main menu, click on the **Browse** button, and find the **ExcelTranslator.xla** file in "**C:\Program Files\Ultimate Risk Solutions\Excel Translator\AddIn**" folder (note that you may have chosen a different folder during the installation).

Secondly, at the beginning of your VBA code you need to create a `DLL` object by adding the following line:

```
Set oDLL = URS_CreateObject("<drive:path\translated_dll_file_name>")
```

The `DLL` object has the following properties:

`InputVariables` - a collection of `Variable` objects
`OutputVariables` - a collection of `Variable` objects

Both `InputVariables` and `OutputVariables` collections have the following methods:

`Count` - returns the number of variables in the collection

`Item(<Index>) or Item("<Name>")` - returns specific `Variable` from the collection by index or by name

NOTE: Item index in `InputVariables` and `OutputVariables` collections is one-based (starts from 1).

`Variable` object has the following properties:

`Name` - read-only property
`Value` - read-write property that can contain a one- or two-dimensional array or a single value

The `DLL` object also has the `Calculate` method that is called after the values of Input Variables are set. The `Calculate` method recalculates the entire DLL and after that the values of Output Variables can be retrieved.

SimpleModelUser.xls sets the values of Input Variables as follows (see `CalculateOutput`() function):

```
adInputValues(1) = m_adValuesSideA(i)
adInputValues(2) = m_adValuesSideB(j)
adInputValues(3) = m_adValuesSideC(k)

oDLL.InputVariables("Sides").Value = adInputValues
```

Then the DLL is recalculated:

```
oDLL.Calculate
```

After that, the calculated values of Output Variables are retrieved from the DLL:

```
m_aOutputRows(nIteration).dVolume = oDLL.OutputVariables("Volume").Value
m_aOutputRows(nIteration).dArea   = oDLL.OutputVariables("Area").Value
```

In **SimpleModelUser.xls**, click Ctrl+R to run the VBA macro that creates different values of input parameters, for each such set of values calls `oDLL.Calculate` to calculate the output values and then writes the input and the output values into the cells on the **Output** sheet.

5.6 Using Translated DLL in C++ Program – Examples and Sample Files

The sample Visual C++ project can be found in **C:\Program Files\Ultimate Risk Solutions\URS Translator\Samples\C++\SimpleModelUser** folder. The project name is **SimpleModel.dsw.** It is a simple C++ application that does exactly what the above VBA module does – calculates the volume and the area of a parallelepiped.

All code of your interest is located in **OnOK()** function of **CSimpleModelDlg** class. First, the DLL should be loaded using the following command:

```
HMODULE hDLL = LoadLibrary( sDllFileName );
```

where `sDllFileName` is the full name of the translated DLL file, including drive and path.

Then the pointer, `pModule`, to a translated DLL object should be obtained using `GetTranslatedObject` function, which is the function exportable from the translated DLL:

```
typedef IURSTranslatedObject* (*PFN_GetTranslatedObject)();

PFN_GetTranslatedObject pfnGetTranslatedObject =
(PFN_GetTranslatedObject)GetProcAddress(hDLL, _T("GetTranslatedObject"));

. . . . . .

IURSModule* pModule = dynamic_cast< IURSModule* >
                              ((*pfnGetTranslatedObject)());
```

Note that `IURSTranslatedObject` and `IURSModule` classes are declared in the header files, **IURSTranslatedObject.h** and **URSModuleInterface.h**. Both header files are included at the beginning of **CSimpleModelDlg.cpp**. The headers are located in **C:\Program Files\Ultimate Risk Solutions\URS Translator\Include** folder.

Then, in our example, we declare the input variables, `mInput`, that will be used to pass the input to the DLL, and the output variable, `mOutput`, that will be used to retrieve the output from the DLL.

```
IURSModule::TMatrix mInput, mOutput;
```

Typedef `TMatrix` is declared in **URSModuleInterface.h** as follows:

```
typedef vector< vector< double > > TMatrix;
```

After that, we set the input variable(s):

```
pModule->SetInputValue( 0, mInput );
```

recalculate DLL:

```
pModule->CalculateFull();
```

and retrieve the output:

```
pModule->GetOutputValue( 0, mOutput );
dVolume = mOutput[ 0 ][ 0 ];

pModule->GetOutputValue( 1, mOutput );
dArea   = mOutput[ 0 ][ 0 ];
```

NOTE: Unlike VBA, all arrays in C++ are zero-based.

At the end, we need to free our DLL as follows:

```
FreeLibrary ( hDLL );
```

Build and execute the application. From **SimpleModelUser** dialog select **SimpleModel.dll**, enter an initial value, step, and the number of values for A, B, and C sides of parallelepiped and click **Run**. The volume and area for each set of inputs will be calculated and shown in the grid of the dialog.

5.7 Using External DLL Function Calls in Spreadsheet Models

Sometimes you may want to write your own functions in programming languages, such as C or C++, place those functions in your own DLL, and use that DLL in your Excel spreadsheet model. You may also want to utilize certain functions from existing DLLs written by other people. *Excel Translator™* understands such function calls to external DLLs.

All you need to do is to declare those external DLL functions in the VBA module of your Excel spreadsheet, which you normally do anyway in order to enable Excel to understand such DLL-based functions.

The example of such use of external DLLs is given in **C:\Program Files\Ultimate Risk Solutions\Excel Translator\Samples\ExternalDLL** folder. The external DLL, which is called **ExternalDLL.dll,** was created in C++. The C++ project with the code for creating the DLL is provided in the **ExternalDLL** sub-folder of the above folder.

There you can see that **ExternalDLL.dll** has a single function, `ExternalDLL_Calculate`, which takes four parameters. First parameter is a string that indicates whether the volume or the area of a parallelepiped needs to be calculated. Three other parameters are the side lengths of the parallelepiped. The function returns either volume or area.

NOTE: You can pass string, numeric, and Boolean parameters to external DLL functions.

The **SimpleModel.xls** now calls this DLL function to calculate volume and area. Look at cells **D7** and **D8** on the **Logic** sheet. The formulas in those cells use `CalculateValue()` function. That function is declared in the VBA module of **SimpleModel.xls** as follows:

```
Public Declare Function CalculateValue Lib _
    "C:\Program Files\Ultimate Risk Solutions\URS Translator\
    Samples\ExternalDLL\ExternalDLL.dll" _
    Alias "ExternalDLL_Calculate" (ByVal sType As String, _
    ByVal A As Double, ByVal b As Double, ByVal c As Double) As Double
```

Note that in the above declaration, `CalculateValue` is used as an alias for the real dll function `ExternalDLL_Calculate`.

NOTE: You don't need to use aliases, call real DLL functions if it's more convenient for you.

After you translate **SimpleModel.xls**, open **SimpleModelUser.xls** and run its macro. You will see that the macro correctly calculates the output values while calling translated **SimpleModel.dll**, which now uses external DLL function calls.

5.8 Testing Translated DLL

The purpose of DLL testing is to ensure that the Excel spreadsheet and the DLL translated from that spreadsheet perform the same calculations and return the same output values given the same input parameters.

The *URS DLL Tester* utility is accessible from **Start→Program Files→Excel Translator.** When you run *URS DLL Tester*, it will ask you to select the translated DLL you want to test and the Excel file from which that DLL was created.

After you select those two files, you are ready to test. Simply enter the values of your input parameters and click the **Test DLL** button. The *URS DLL Tester* works as follows:

- It starts Excel
- It populates the input area of the DLL, as well as the input cells of the original Excel spreadsheet with the same values of input parameters
- It runs DLL calculations and Excel calculations
- It goes through every Excel cell that impacts the output values and compares the value produced by Excel's formula with the corresponding value produced by the DLL
- If there are mismatches between the values calculated by Excel and by the DLL, they will be displayed on the Discrepancy dialog that *EXCEL DLL Tester* shows at the end of each test.

The Discrepancy dialog lists the Excel address of each cell for which the discrepancy was found, its Excel value, and its corresponding DLL value. If you double-click on the item in the list-box on the dialog, *the DLL Tester* will reposition the cursor directly in the cell that produced the selected discrepancy.

The most likely cause of "discrepancies" is when a new version of the model is compared to an old DLL, produced by translating an older version of the model. When such a new spreadsheet is not retranslated, a comparison with the old DLL yields discrepancies.

If there is a discrepancy in one cell, then all cells dependent on that cell will also show discrepancies. That is why the Discrepancy dialog gives you the option of either viewing only the cells with the sources of errors or viewing all cells with the errors. You can also set a watch on specific cells and those cells will be shown in the list-box.

NOTE: You can also test your translated DLL from *Model Builder* or from any other external application. With *Model Builder*, you have an advantage of being able to run DLL testing in automatic mode whereby the test will be performed iteratively until the first discrepancy is found.

If you are running your translated model from other VBA application, you can run DLL testing either at each individual iteration or in automatic mode. Review VBA code in **SimpleModelUser.xls** as an example.

At the beginning (see **RunModel()** function), the path and name for both DLL and Excel files are declared. Then the DLL object is created and DLL testing enabled:

```
m_sDll = " C:\Program Files\Ultimate Risk Solutions\Excel Translator
           \Samples\VBA\SimpleModel.dll"
```

```
m_sXls = " C:\Program Files\Ultimate Risk Solutions\Excel Translator
           \Samples\VBA\SimpleModel.xls"
' Create object from translated DLL
Set oDLL = EXCEL_CreateObject(m_sDll)

' Initialize dll tester
Call oDLL.EnableDllTesting(Application, m_sXls)
```

Then at each iteration (see **CalculateOutput()** function), the DLL testing can be performed:

```
' 0 - no testing
' 1 - "usual" mode
' 2 - "auto step-through" mode
bContinue = oDLL.TestDll(1)

If False = bContinue Then Exit Sub
```

If the parameter passed into the **TestDll()** function is 0, the function does nothing and simply returns `true`, so that the program can continue executing. If the parameter is 1, the Discrepancy dialog is shown. If the user chooses to continue testing by closing the Discrepancy dialog, the function returns `true`. If the user decides to stop testing, the function returns `false`.

If the **TestDll()** function parameter is 2, the program performs DLL testing automatically and returns `true` if there are no discrepancies. If, at any iteration, one or more discrepancies are found, the *DLL Tester* pauses to display the Discrepancy dialog. Then, depending on the user's action – continue testing (by closing the dialog) or stop testing – the function returns `true` or `false`.

At the end of your VBA application (see **RunModel()** function), you always need to finish DLL testing and destroy the DLL object created at the beginning:

```
' stop dll tester
Call oDLL.FinishDllTesting
Set oDLL = Nothing
```

6 *URS Model Builder* and *Result Viewer*

URS Model Builder is an auxiliary application supplied with *Excel Translator™* that makes working with Excel-translated components easy and convenient. It allows you to build complicated models using some previously translated models as building blocks. Thus, for example, when you have built the models for several asset classes, such as bonds and stocks, you can combine them to model the behavior of a portfolio comprised of such assets.

Start *URS Model Builder*, then create new project (**File…New Project**). A **Project** contains **Components** folder to which you can add Excel-translated components on right click. When you add an Excel-translated component, you will be prompted to upload its DLL. You can also specify the path\name of the original Excel file that was compiled to produce that DLL and enable DLL testing.

NOTE: *URS Model Builder* quickly runs calculations of one or more Excel-translated components multiple times, as specified by the number of **iterations**. At each iteration, *URS Model Builder* produces the values of all input variables, runs calculations of all Excel translated components in a project, and processes the values of the specified output variables.

Excel-translated component's input and output variables are displayed on *URS Model Builder*'s project tree. The values of input variables can be produced, at each iteration, by several different means which you can see if you right click on any input variable on a project tree. The input variable can be:

- Set as Constant
- Simulated Stochastically from a statistical distribution (variety of distributions are available)
- Read Sequentially from a tab-delimited text file or from an array of values
- Set Equal to Other Variable, whether to the output variable from other component or to the input variable from any component
- Calculated as Dependent on other variables.

You can build a sophisticated model that will execute fast by creating Excel-translated components and connecting them together in *URS Model Builder* by setting input variables of one component equal to the output variables of other components. Thus you can create a sophisticated model with nothing else but the models, which you created previously.

Besides specifying how you want to produce the values of the input variables, before you run your model, you also need to specify how you want to process the values of the output variables. Right click on any output variable and select **Accumulate Statistics**, in which

case the empirical distribution (histogram) for that variable will be produced at the end of all iterations.

You can also select the **Save All Iterations** item, in which case each single iteration will save the corresponding value of that output variable into a binary Results Storage (**RST**) file. The name and location of the **RST** file is specified in Iteration Options (select **Iterations…Options** from the main menu) on the **Results** page.

After you set the **Accumulate Statistics** and/or **Save All Iterations** flags on the output variables of your interest, set the number of iterations in Iteration Options and then run the model by choosing **Iterations…Run** from the main menu. At the end of a model's run, the empirical distributions (histograms) for all variables marked with the **Accumulate Statistics** flag will be produced. The values calculated at each iteration for all variables marked with the **Save All Iterations** flag will be saved in the specified **RST** file.

The **RST** file can then be viewed with the *Result Viewer*, an auxiliary application supplied with *URS Model Builder*. The *Result Viewer* can read RST files, produce reports from their data, copy/paste that data to Excel or other spreadsheet applications or export it to other files or databases.

The *Result Viewer* can be invoked either from *URS Model Builder* by selecting **Iterations…Result Viewer** from the main menu or by selecting appropriate item from **Start→Programs→Excel Translator** group.

Create new report in the *Result Viewer*, select the **Iteration Table** report, find the relevant **RST** file and specify which variables you want to list in the iteration table. The *Result Viewer* will show the table of values produced in each iteration for each specified variable.

You can copy this table (**Edit…Copy Entire Table**) and paste it to Excel. You can also export the table (**File…Export**) to tab-delimited text (**TXT**) file, to comma delimited (**CSV**) file, or to *Microsoft Access* (**MDB**) database.

7 Using URS Statistical Distribution and Math Library

If you work with statistical distributions, which is often the case in financial and stochastic models, in addition to Excel's functions, *Excel Translator*™ offers a variety of distributions available from the URS Distribution and Math Library, which you can use in your Excel model. The URS Distribution and Math Library comes with *Excel Translator*™ and is available in Excel as URS functions. All URS functions can be found in the **User-Defined** section when you select **Insert…Function** from Excel's Main Menu or when you click on the function toolbar.

URS distribution functions include the functions that generate random values from the distributions and functions that calculate various distribution properties, such as mean, standard deviation, moments, or distribution percentiles. URS math functions include random number generation, calculation of lower and upper triangular matrices, and generation of the sets of correlated standard normal random numbers.

NOTE: Excel models that you can translate with *Excel Translator*™ and run in fast repeating mode can include stochastic elements.

The complete list of URS functions available for use in Excel with *Excel Translator*™, along with the description of each function and its parameters, is presented below. Distribution formulas are provided in Section 8 of this Guide.

The examples of how to use URS functions in your Excel model or in a VBA macro are given in the **UsingURSFunctionsInExcel.xls** and **UsingURSFunctionsInVBA.xls** files located in **C:\Program Files\Ultimate Risk Solutions\Excel Translator\Samples\URSFunctions** folder**.** The URS functions are shown on the **Output** sheet.

Select **Tools…Customize** from Excel's Main Menu, then select the **Commands** tab, and choose **Tools** item from the list. Add the **Calculate Full** button to one of Excel's toolbars. Now, when you click the **Calculate Full** button, you will see different random values generated in cells **I18:I21** of the **Output** sheet.

URS_DIST_GetCdfPercent

Purpose: Calculates Cdf percent from value X

Parameters:

X	-	value
SType	-	text string, which is a distribution type (see list of available types below)
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum

BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_DIST_GetCV

Purpose: Calculates coefficient of variation of the distribution

Parameters:

SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_DIST_GetDensityValue

Purpose: Calculates Pdf or Pmf function for value X

Parameters:

X	-	value
SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_DIST_GetDeviate

Purpose: Generates random value (deviate) from a distribution

Parameters:

SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a

mass point) at maximum, default is FALSE

URS_DIST_GetInverseCdf

Purpose: Calculates inverse Cdf from Cdf percent

Parameters:

Percent	-	Cdf percent
SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_DIST_GetMean

Purpose: Calculates distribution mean

Parameters:

SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_DIST_GetMoment

Purpose: Calculates specified moment of a distribution

Parameters:

Moment	-	integer, which is a moment number
SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_DIST_GetStDev

Purpose: Calculates distribution standard deviation

Parameters:

SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_DIST_GetVariance

Purpose: Calculates distribution variance

Parameters:

SType	-	text string, which is a distribution type
RParameters	-	Excel range where you entered distribution parameters
DValueMin	-	distribution minimum
DValueMax	-	distribution maximum
BMassOnMin	-	optional parameter, if TRUE distribution is truncated (has a mass point) at minimum, default is FALSE
BMassOnMax	-	optional parameter, if TRUE distribution is truncated (has a mass point) at maximum, default is FALSE

URS_MATH_GetRandomNumber

Purpose: Generates random number from 0 to 1.

Parameters: None

URS_DIST_GetDeviateStandardNormal

Purpose: Calculates random value from Standard Normal distribution.

Parameters: None

URS_DIST_GetDeviateChiSquare

Purpose: Calculates random value from Chi-Square distribution.

Parameters:
nDegreeOfFreedom - degree of freedom (integer value)

URS_DIST_GetDeviateStudentT

Purpose: Calculates random value from Student-t distribution.

Parameters:
nDegreeOfFreedom - degree of freedom (integer value)

URS_MATH_GetLowerMatrix

Purpose: Calculates lower triangular matrix from correlation matrix using Choleski decomposition. Used in array formulas.

Parameters:
RCorrelationMatrix - Excel range that contains the correlation matrix. All diagonal elements in a correlation matrix should equal 1 and symmetrical elements should be equal. Number of rows should equal number of columns.

URS_MATH_GetUpperMatrix

Purpose: Calculates upper triangular matrix from correlation matrix using Choleski decomposition. Used in array formulas.

Parameters:
RCorrelationMatrix - Excel range that contains the correlation matrix. All diagonal elements in a correlation matrix should equal 1 and symmetrical elements should be equal. Number of rows should equal number of columns.

URS_MATH_GetCorrelatedStNormalDeviates

Purpose: Generates an array of correlated standard normal deviates. Use in array formulas.

Parameters:
RCorrelationMatrix - Excel range that contains the correlation matrix. All diagonal elements in a correlation matrix should equal 1 and symmetrical elements should be equal. Number of rows should equal number of columns.

URS_MATH_GetCorrelatedDeviates

Purpose: Generates an array of deviates from the distributions you specify correlated via Normal Copula. Use in array formulas.

Parameters:
RCorrelationMatrix - Excel range that contains the correlation matrix. All diagonal elements in a correlation matrix should equal 1 and symmetrical elements should be equal. Number of rows should equal number of columns.

RDistributionParameterMatrix - Excel range that contains the matrix of distribution parameters. Number of rows in the matrix must be equal to the number of rows (columns) in the correlation matrix. Rows in distribution parameter matrix should contain information about the distributions to be correlated in the following order:

- Column 1: Distribution type
- Columns 2 through N: Distribution parameters
- Column N+1: Distribution minimum
- Column N+2: Distribution maximum
- Column N+3 (Optional): Boolean that specifies if there is a mass point at minimum.
- Column N+4 (Optional): Boolean that specifies if there is a mass point at maximum.

The number of columns in the matrix depends on the greatest number of parameters in the distributions to be correlated and on using last optional Booleans.

List of available distribution types is given below:

Discrete distributions -

Bernoulli
Binomial
NegativeBinomial
Poisson

Continuous distributions –

BetaTransformed
Burr
BurrInverse
Chi-Square
Exponential
ExponentialInverse

Gamma
GammaInverse
GammaTransformed
GammaTransformedInverse
Gumbel
Loglogistic
Lognormal
Normal
Paralogistic
ParalogisticInverse
Pareto
ParetoGeneralized
ParetoInverse
ParetoSimple
StandardNormal
Student-T
Uniform
Weibull
WeibullInverse.

8. Statistical Distributions of *Excel Translator™*

The statistical distributions provided by *Excel Translator*™ are as described in *Loss Models: From Data To Decisions* by Stuart A. Klugman, Harry H. Panjer, and Gordon E. Willmot.

This section provides the Pdf/Pmf and Cdf formulas, as defined in the book, for each distribution used in *Excel Translator*™. In addition, for each continuous distribution, the moments of unlimited distribution and the moments of the distribution truncated from above are provided, also as given in the *Loss Models* book.

Excel Translator™ (also known as URS Translator, when packaged with URS flagship product, *The Risk Explorer™)* gives model designers the additional flexibility of using distributions, which can be limited (conditional) from below and from above, and also can be truncated from below and from above. Limited (conditional) distributions progress smoothly from a minimum to a maximum value, and truncated distributions have mass points, either at a minimum, at a maximum, or at both extremes. When you create limited or truncated distributions, *Excel Translator*™ automatically adjusts the formulas to reflect the distribution behavior.

- When you specify the minimum and/or maximum value for the distribution property page and do not use the mass point parameters, you create a *limited* distribution.
- If you set BMassOnMin or BMassOnMax parameters to TRUE, you create a *truncated* distribution at either the minimum or maximum.
- If you set both BMassOnMin and BMassOnMax parameters to TRUE, you create a distribution *truncated* at both ends.

8.1 Continuous Distributions

Beta Transformed

$$f(x)=\frac{\Gamma(\alpha+\tau)}{\Gamma(\alpha)\Gamma(\tau)}\frac{\gamma(x/\theta)^{\gamma\tau}}{x\left[1+(x/\theta)^{\gamma}\right]^{\alpha+\tau}}$$

$$F(x)=\beta(\tau,\alpha;u),\quad u=\frac{(x/\theta)^{\gamma}}{1+(x/\theta)^{\gamma}}$$

$$\mathrm{E}\left[X^{k}\right]=\frac{\theta^{k}\Gamma(\tau+k/\gamma)\Gamma(\alpha-k/\gamma)}{\Gamma(\alpha)\Gamma(\tau)},\quad -\tau\gamma<k<\alpha\gamma$$

$$\mathrm{E}\left[(X \wedge x)^k\right] = \frac{\theta^k \Gamma(\tau + k/\gamma)\Gamma(\alpha - k/\gamma)}{\Gamma(\alpha)\Gamma(\tau)} \beta(\tau + k/\gamma, \alpha - k/\gamma; u) + x^k\left[1 - F(x)\right], \quad k > -\tau\gamma$$

$$\text{mode} = \theta\left(\frac{\tau\gamma - 1}{\alpha\gamma + 1}\right)^{1/\gamma}, \quad \tau\gamma > 1, \ else\, 0$$

Burr

$$f(x) = \frac{\alpha\gamma(x/\theta)^\gamma}{x\left[1 + (x/\theta)^\gamma\right]^{\alpha+1}}$$

$$F(x) = 1 - u^\alpha, \qquad u = \frac{1}{1 + (x/\theta)^\gamma}$$

$$\mathrm{E}\left[X^k\right] = \frac{\theta^k \Gamma(1 + k/\gamma)\Gamma(\alpha - k/\gamma)}{\Gamma(\alpha)}, \quad -\gamma < k < \alpha\gamma$$

$$\mathrm{E}\left[(X \wedge x)^k\right] = \frac{\theta^k \Gamma(1 + k/\gamma)\Gamma(\alpha - k/\gamma)}{\Gamma(\alpha)} \beta(1 + k/\gamma, \alpha - k/\gamma; 1 - u) + x^k u^\alpha, \quad k > -\gamma$$

$$\text{mode} = \theta\left(\frac{\gamma - 1}{\alpha\gamma + 1}\right)^{1/\gamma}, \quad \gamma > 1, \ else\ 0$$

Burr Inverse

$$f(x) = \frac{\tau\gamma(x/\theta)^{\gamma\tau}}{x\left[1 + (x/\theta)^\gamma\right]^{\tau+1}}$$

$$F(x) = u^\tau, \quad u = \frac{(x/\theta)^\gamma}{1 + (x/\theta)^\gamma}$$

$$\mathrm{E}\left[X^k\right] = \frac{\theta^k \Gamma(\tau + k/\gamma)\Gamma(1 - k/\gamma)}{\Gamma(\tau)}, \quad -\tau\gamma < k < \gamma$$

$$\mathrm{E}\left[(X \wedge x)^k\right] = \frac{\theta^k \Gamma(\tau + k/\gamma)\Gamma(1 - k/\gamma)}{\Gamma(\tau)} \beta(\tau + k/\gamma, 1 - k/\gamma; u) + x^k\left[1 - u^\tau\right], \quad k > -\tau\gamma$$

$$\text{mode} = \theta\left(\frac{\tau\gamma - 1}{\gamma + 1}\right)^{1/\gamma}, \quad \tau\gamma > 1, \ else\ 0$$

Exponential

$$f(x)=\frac{e^{-x/\theta}}{\theta}$$
$$F(x)=1-e^{-x/\theta}$$
$$\mathrm{E}\left[X^k\right]=\theta^k\Gamma(k+1),\ k>-1$$
$$\mathrm{E}\left[X^k\right]=\theta^k k!, \text{ if } k \text{ is an integer}$$
$$\mathrm{E}\left[X\wedge x\right]=\theta\left(1-e^{-x/\theta}\right)$$
$$\mathrm{E}\left[(X\wedge x)^k\right]=\theta^k\Gamma(k+1)\Gamma(k+1;x/\theta)+x^k e^{-x/\theta},\ k>-1$$
$$=\theta^k k!\Gamma(k+1;x/\theta)+x^k e^{-x/\theta},\quad k \text{ an integer}$$
mode = 0

Exponential Inverse

$$f(x)=\frac{\theta e^{-\theta/x}}{x^2}$$
$$F(x)=e^{-\theta/x}$$
$$\mathrm{E}\left[X^k\right]=\theta^k\Gamma(1-k),\ k<1$$
$$\mathrm{E}\left[(X\wedge x)^k\right]=\theta^k\Gamma(1-k)G(1-k;\theta/x)+x^k\left(1-e^{-\theta/x}\right),\quad \text{all } k$$
mode= $\theta/2$

Gamma

$$f(x)=\frac{(x/\theta)^\alpha e^{-x/\theta}}{x\Gamma(\alpha)}$$
$$F(x)=\Gamma(\alpha;x/\theta)$$
$$\mathrm{E}\left[X^k\right]=\frac{\theta^k\Gamma(\alpha+k)}{\Gamma(\alpha)},\quad k>-\alpha$$
$$\mathrm{E}\left[X^k\right]=\theta^k(\alpha+k-1)\cdots\alpha,\quad \text{if } k \text{ is an integer}$$
$$\mathrm{E}\left[(X\wedge x)^k\right]=\frac{\theta^k\Gamma(\alpha+k)}{\Gamma(\alpha)}\Gamma(\alpha+k;x/\theta)+x^k\left[1-\Gamma(\alpha;x/\theta)\right],\ k>-\alpha$$
$$=\alpha(\alpha+1)\cdots(\alpha+k-1)\theta^k\Gamma(\alpha+k;x/\theta)+x^k\left[1-\Gamma(\alpha;x/\theta)\right],\quad k \text{ an integer}$$
mode= $\theta(\alpha-1),\quad \alpha>1,$ else 0

Gamma Inverse

$$f(x)=\frac{(\theta/x)^{\alpha}e^{-\theta/x}}{x\Gamma(\alpha)}$$

$$F(x)=1-\Gamma(\alpha;\theta/x)$$

$$\mathrm{E}\left[X^{k}\right]=\frac{\theta^{k}\Gamma(\alpha-k)}{\Gamma(\alpha)},\quad k<\alpha$$

$$\mathrm{E}\left[X^{k}\right]=\frac{\theta^{k}}{(\alpha-1)\cdots(\alpha-k)},\ \text{if k is an integer}$$

$$\mathrm{E}\left[(X\wedge x)^{k}\right]=\frac{\theta^{k}\Gamma(\alpha-k)}{\Gamma(\alpha)}\left[1-\Gamma(\alpha-k;\theta/x)\right]+x^{k}\Gamma(\alpha;\theta/x)\quad,\ \text{all } k$$

$$=\frac{\theta^{k}\Gamma(\alpha-k)}{\Gamma(\alpha)}G(\alpha-k;\theta/x)+x^{k}\Gamma(\alpha;\theta/x)$$

$$=\frac{\theta^{k}}{(\alpha-1)\cdots(\alpha-k)}G(\alpha-k;\theta/x)+x^{k}\Gamma(\alpha;\theta/x),\quad k\ \text{an integer}$$

$$\text{mode}=\theta/(\alpha+1)$$

Gamma Transformed

$$f(x)=\frac{\tau u^{\alpha}e^{-u}}{x\Gamma(\alpha)},\quad u=(x/\theta)^{\tau}$$

$$F(x)=\Gamma(\alpha;u)$$

$$\mathrm{E}\left[X^{k}\right]=\frac{\theta^{k}\Gamma(\alpha+k/\tau)}{\Gamma(\alpha)},\quad k>-\alpha\tau$$

$$\mathrm{E}\left[(X\wedge x)^{k}\right]=\frac{\theta^{k}\Gamma(\alpha+k/\tau)}{\Gamma(\alpha)}\Gamma(\alpha+k/\tau;u)+x^{k}\left[1-\Gamma(\alpha;u)\right],\quad k>-\alpha\tau$$

$$\text{mode}=\theta\left(\frac{\alpha\tau-1}{\tau}\right)^{1/\tau},\quad \alpha\tau>1,\ \textit{else}\ 0$$

Gamma Transformed Inverse

$$f(x)=\frac{\tau u^{\alpha}e^{-u}}{x\Gamma(\alpha)},\qquad u=(\theta/x)^{\tau}$$

$$F(x)=1-\Gamma(\alpha;u)$$

$$\mathrm{E}\left[X^{k}\right]=\frac{\theta^{k}\Gamma(\alpha-k/\tau)}{\Gamma(\alpha)},\qquad k<\alpha\tau$$

$$\mathrm{E}\left[(X \wedge x)^k\right] = \frac{\theta^k \Gamma(\alpha - k/\tau)}{\Gamma(\alpha)}\left[1 - \Gamma(\alpha - k/\tau; u)\right] + x^k \Gamma(\alpha; u)$$
$$= \frac{\theta^k \Gamma(\alpha - k/\tau)}{\Gamma(\alpha)} G(\alpha - k/\tau; u) + x^k \Gamma(\alpha; u)$$

$$\text{mode} = \theta\left(\frac{\tau}{\alpha\tau + 1}\right)^{1/\tau}$$

Loglogistic

$$f(x) = \frac{\gamma (x/\theta)^\gamma}{x\left[1 + (x/\theta)^\gamma\right]^2}$$
$$F(x) = u, \; u = \frac{(x/\theta)^\gamma}{1 + (x/\theta)^\gamma}$$
$$\mathrm{E}\left[X^k\right] = \theta^k \Gamma(1 + k/\gamma)\Gamma(1 - k/\gamma), \quad -\gamma < k < \gamma$$
$$\mathrm{E}\left[(X \wedge x)^k\right] = \theta^k \Gamma(1 + k/\gamma)\Gamma(1 - k/\gamma)\beta(1 + k/\gamma, 1 - k/\gamma; u) + x^k (1 - u), \quad k > -\gamma$$

$$\text{mode} = \theta\left(\frac{\gamma - 1}{\gamma + 1}\right)^{1/\gamma}, \qquad \gamma > 1, \quad \textit{else } 0$$

Lognormal

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-z^2/2\right) = \phi(z)/(\sigma x), \quad z = \frac{\log x - \mu}{\sigma}$$
$$F(x) = \Phi(z)$$
$$\mathrm{E}\left[X^k\right] = \exp\left(k\mu + k^2\sigma^2/2\right)$$
$$\mathrm{E}\left[(X \wedge x)^k\right] = \exp\left(k\mu + k^2\sigma^2/2\right)\Phi\left(\frac{\log x - \mu - k\sigma^2}{\sigma}\right) + x^k\left[1 - F(x)\right]$$
$$\text{mode} = \exp\left(\mu - \sigma^2\right)$$

Normal

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$
$$F(x) = \Phi\left(\frac{x - \mu}{\sigma}\right)$$

Paralogistic (this is a Burr distribution with $\gamma = \alpha$)

$$f(x) = \frac{\alpha^2 (x/\theta)^\alpha}{x\left[1+(x/\theta)^\alpha\right]^{\alpha+1}}$$

$$F(x) = 1-u^\alpha, \quad u = \frac{1}{1+(x/\theta)^\alpha}$$

$$\mathrm{E}\left[X^k\right] = \frac{\theta^k \Gamma(1+k/\alpha)\Gamma(\alpha-k/\alpha)}{\Gamma(\alpha)}, \qquad -\alpha < k < \alpha^2$$

$$\mathrm{E}\left[(X \wedge x)^k\right] = \frac{\theta^k \Gamma(1+k/\alpha)\Gamma(\alpha-k/\alpha)}{\Gamma(\alpha)}\beta(1+k/\alpha, \alpha-k/\alpha; 1-u) + x^k u^\alpha, \quad k > -\alpha$$

$$\text{mode} = \theta\left(\frac{\alpha-1}{\alpha^2+1}\right)^{1/\alpha}, \quad \alpha > 1, \quad \textit{else } 0$$

Paralogistic Inverse (this is an Inverse Burr distribution with $\gamma = \tau$)

$$f(x) = \frac{\tau^2 (x/\theta)^{\tau^2}}{x\left[1+(x/\theta)^\tau\right]^{\tau+1}}$$

$$F(x) = u^\tau, \qquad u = \frac{(x/\theta)^\tau}{1+(x/\theta)^\tau}$$

$$\mathrm{E}\left[X^k\right] = \frac{\theta^k \Gamma(\tau+k/\tau)\Gamma(1-k/\tau)}{\Gamma(\tau)}, \qquad -\tau^2 < k < \tau$$

$$\mathrm{E}\left[(X \wedge x)^k\right] = \frac{\theta^k \Gamma(\tau+k/\tau)\Gamma(1-k/\tau)}{\Gamma(\tau)}\beta(\tau+k/\tau, 1-k/\tau; u) + x^k\left[1-u^\tau\right], \quad k > -\tau^2$$

$$\text{mode} = \theta(\tau-1)^{1/\tau}, \qquad \tau > 1, \quad \textit{else } 0$$

Pareto

$$f(x) = \frac{\alpha\theta^\alpha}{(x+\theta)^{\alpha+1}}$$

$$F(x) = 1-\left(\frac{\theta}{x+\theta}\right)^\alpha$$

$$\mathrm{E}\left[X^k\right] = \frac{\theta^k \Gamma(k+1)\Gamma(\alpha-k)}{\Gamma(\alpha)}, \quad -1 < k < \alpha$$

$$\mathrm{E}\left[X^k\right]=\frac{\theta^k k!}{(\alpha-1)\cdots(\alpha-k)}, \quad \text{if } k \text{ is an integer}$$

$$\mathrm{E}\left[X\wedge x\right]=\frac{\theta}{\alpha-1}\left[1-\left(\frac{\theta}{x+\theta}\right)^{\alpha-1}\right], \quad \alpha\neq 1$$

$$\mathrm{E}\left[X\wedge x\right]=-\theta\log\left(\frac{\theta}{x+\theta}\right), \quad \alpha=1$$

$$\mathrm{E}\left[(X\wedge x)^k\right]=\frac{\theta^k\Gamma(k+1)\Gamma(\alpha-k)}{\Gamma(\alpha)}\beta\left[k+1,\alpha-k;x/(x+\theta)\right]+x^k\left(\frac{\theta}{x+\theta}\right)^{\alpha}, \quad \textit{all } k$$

mode = 0

Pareto Generalized

$$f(x)=\frac{\Gamma(\alpha+\tau)}{\Gamma(\alpha)\Gamma(\tau)}\cdot\frac{\theta^\alpha x^{\tau-1}}{(x+\theta)^{\alpha+\tau}}$$

$$F(x)=\beta(\tau,\alpha;u), \quad u=\frac{x}{x+\theta}$$

$$\mathrm{E}\left[X^k\right]=\frac{\theta^k\Gamma(\tau+k)\Gamma(\alpha-k)}{\Gamma(\alpha)\Gamma(\tau)}, \quad -\tau<k<\alpha$$

$$\mathrm{E}\left[X^k\right]=\frac{\theta^k\tau(\tau+1)\cdots(\tau+k-1)}{(\alpha-1)\cdots(\alpha-k)}, \quad \text{if k is an integer}$$

$$\mathrm{E}\left[(X\wedge x)^k\right]=\frac{\theta^k\Gamma(\tau+k)\Gamma(\alpha-k)}{\Gamma(\alpha)\Gamma(\tau)}\beta(\tau+k,\alpha-k;u)+x^k\left[1-F(x)\right], \quad k>-\tau$$

$$\text{mode}=\theta\frac{\tau-1}{\alpha+1}, \quad \tau>1, \ \textit{else}\ 0$$

Pareto Inverse

$$f(x)=\frac{\tau\theta x^{\tau-1}}{(x+\theta)^{\tau+1}}$$

$$F(x)=\left(\frac{x}{x+\theta}\right)^{\tau}$$

$$\mathrm{E}\left[X^k\right]=\frac{\theta^k\Gamma(\tau+k)\Gamma(1-k)}{\Gamma(\tau)}, \quad -\tau<k<1$$

$$\mathrm{E}\left[X^k\right]=\frac{\theta^k(-k)!}{(\tau-1)\cdots(\tau+k)}, \quad \text{if } k \text{ is a negative integer}$$

$$\mathrm{E}\left[(X\wedge x)^k\right]=\theta^k\tau\int_0^{x/(x+\theta)}y^{\tau+k-1}(1-y)^{-k}\,dy+x^k\left[1-\left(\frac{x}{x+\theta}\right)^{\tau}\right], \quad k>-\tau$$

$$\text{mode}=\theta\frac{\tau-1}{2}, \qquad \tau>1, \quad else\ \ 0$$

Pareto Simple [a, b]

$$f(x)=\alpha\frac{a^{\alpha}}{x^{\alpha+1}}\frac{1}{\left(1-\left(\frac{a}{b}\right)^{\alpha}\right)}$$

$$F(x)=\frac{1-\left(\frac{a}{x}\right)^{\alpha}}{1-\left(\frac{a}{b}\right)^{\alpha}}$$

$$\mathrm{E}\left(X^{k}\right)=\frac{\alpha\cdot a^{\alpha}}{k-\alpha}\left(b^{k-\alpha}-a^{k-\alpha}\right)\frac{1}{1-\left(\frac{a}{b}\right)^{\alpha}}$$

Uniform

$$f(x)=\begin{cases}\frac{1}{b-a} & ,a\le x\le b\\ 0 & \text{otherwise}\end{cases}$$

$$F(x)=\begin{cases}0 & ,\ x<a\\ \frac{x-a}{b-a} & ,a\le x\le b,\\ 1 & ,x>b\end{cases}$$

$$\mathrm{E}\left(X^{k}\right)=\frac{b^{k+1}-a^{k+1}}{(k+1)(b-a)}$$

Weibull

$$f(x)=\frac{\tau(x/\theta)^{\tau}e^{-(x/\theta)^{\tau}}}{x}$$

$$F(x)=1-e^{-(x/\theta)^{\tau}}$$

$$\mathrm{E}\left[X^{k}\right]=\theta^{k}\Gamma(1+k/\tau), \quad k>-\tau$$

$$\mathrm{E}\left[(X\wedge x)^{k}\right]=\theta^{k}\Gamma(1+k/\tau)\Gamma\left[1+k/\tau;(x/\theta)^{\tau}\right]+x^{k}e^{-(x/\theta)^{\tau}},\ \mathrm{k}>-\tau$$

$$\text{mode}=\theta\left(\frac{\tau-1}{\tau}\right)^{1/\tau},\ \tau>1,\ \text{else}\ \ 0$$

Weibull Inverse

$$f(x)=\frac{\tau(\theta/x)^{\tau}e^{-(\theta/x)^{\tau}}}{x}$$

$$F(x)=e^{-(\theta/x)^{\tau}}$$

$$\mathrm{E}\left[X^{k}\right]=\theta^{k}\Gamma(1-k/\tau),\ k<\tau$$

$$\mathrm{E}\left[(X\wedge x)^{k}\right]=\theta^{k}\Gamma(1-k/\tau)\left\{1-\Gamma\left[1-k/\tau;(\theta/x)^{\tau}\right]\right\}+x^{k}\left[1-e^{-(\theta/x)^{\tau}}\right],\quad \text{all } k$$

$$=\theta^{k}\Gamma(1-k/\tau)G\left[1-k/\tau;(\theta/x)^{\tau}\right]+x^{k}\left[1-e^{-(\theta/x)^{\tau}}\right]$$

$$\text{mode}=\theta\left(\frac{\tau}{\tau+1}\right)^{1/\tau}$$

Many of the above distributions use Incomplete Beta and Incomplete Gamma functions. The formulas for these functions are given below:

Incomplete beta

$$\beta(a,b,x)=\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\int_{0}^{x}t^{a-1}(1-t)^{b-1}dt,\qquad a>0,\ b>0,0<x<1$$

Incomplete gamma

$$\Gamma(\alpha;x)=\frac{1}{\Gamma(\alpha)}\int_{0}^{x}t^{\alpha-1}e^{-t}dt,\quad \alpha>0,\ x>0$$

$$\Gamma(\alpha)=\int_{0}^{\infty}t^{\alpha-1}e^{-t}dt,\qquad \alpha>0$$

when $a\le 0$ the integral does not exist. In that case, define

$$\Gamma(\alpha)G(\alpha;x)=\int_{x}^{\infty}t^{\alpha-1}e^{-t}dt,\qquad x>0$$

8.2 Discrete Distributions

In *URS Translator* the parameters used for discrete distributions, such as Poisson, Binomial, and Negative Binomial are always Mean and Variance-to-Mean ratio (for Poisson distributions, only the Mean parameter is relevant). The conversion formulas from Klugman's parameters to the Mean and Variance-to-Mean ratio are provided.

Poisson

$p_0 = e^{-\lambda}, \quad a = 0, \quad b = \lambda$

$$p_k = \frac{e^{-\lambda}\lambda^k}{k!}$$
$$\mathrm{E}[\mathrm{N}] = \lambda, \ Var[\mathrm{N}] = \lambda$$

Mean = λ

Binomial

$p_0 = (1-q)^m, \qquad a = -q/(1-q), \quad b = (m+1)q/(1-q), \quad 0 < q < 1$, m an integer

$$p_k = \binom{m}{k} q^k (1-q)^{m-k}, \quad k = 0,1,\ldots,m$$
$$\mathrm{E}[\mathrm{N}] = mq, \ Var[\mathrm{N}] = mq(1-q)$$

Mean = mq, Variance-to-Mean = $(1-q)$

Negative Binomial

$p_0 = (1+\beta)^{-r}, \ a = \beta/(1+\beta), \ b = (r-1)\beta/(1+\beta)$

$$p_k = \frac{r(r+1)\cdots(r+k-1)\beta^k}{k!(1+\beta)^{r+k}}$$

$E[N] = r\beta, \ Var[N] = r\beta(1+\beta)$

Mean = $r\beta$, Variance-to-Mean = $(1+\beta)$