# Chi-Squared Example Excel Add-in

## Introduction

Earlier we wrote a simple console C# example application that used a small C++/CLI wrapper to calculate the Chi-Squared distribution using the boost library. The output was then sent to an Excel sheet. The advantage was that we could combine functionality available in boost (C++) with the ease of programming in C#.

We decided to convert this example into an Excel COM add-in. Making it as COM add-in has as advantage that it runs inside Excel and thus can create the Excel sheet much faster as there are no out-of-process calls needed anymore. And the COM add-in still has the advantage that you can use fast C++ code.

Instead of making the COM add-in in C++/ATL, we decided to make the COM add-in in C# using the “Shared Add-in” project wizard. Creating a COM add-in in C++ is difficult and error-prone. Using the C# “Shared Add-in” project wizard is much easier.

The wizard generates a *Connect* class that implements the *IExtensibility2* interface required for COM add-ins. The setup project that is also generated by the wizard, handles registering the add-in in the registry.

In the *OnConnection()* method of the generated *Connect* class, we add code to create a menu item that calls our callback function (In .NET this is handled by a delegate). In the *OnDisconnection()* method we add code that removes the menu item we created in the *OnConnection()* method. The menu item is in Excel 2007 available in the “Add-In” ribbon menu.

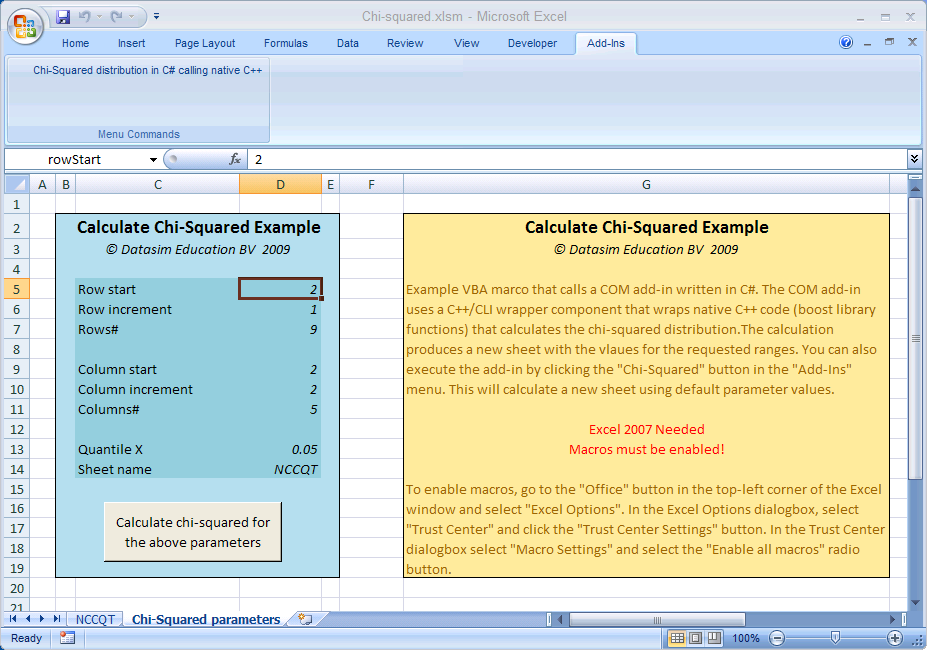
In the callback function we added to the C# add-in we use the same C++/CLI wrapper to calculate the Chi-Squared distribution using the boost library. Then the results are written by the C# add-in directly to a new Excel sheet.

To finish this example, we wanted to create an Excel sheet were you enter all the required parameters. Then a button on the Excel sheet calls our COM add-in function. Thus our add-in must also expose a function callable by an Excel VBA macro. We do this by creating a COM interface and let the generated *Connect* class implement that interface. The function in the interface is now visible from Excel VBA.

So the VBA macro behind the button on the sheet passes the proper cell values to a COM add-in written in C#. The C# COM add-in then uses a tiny wrapper written in C++/CLI to call native C++ functionality.

Below you will find the code of the *Connect* class. Note that it uses calls to general utility code that is not included in this document.

## Excel sheet calling the COM add-in



## Connect class & IConnect interface

namespace CsExcelAddIn

{

using System;

using System.Collections.Generic;

using Extensibility;

using System.Runtime.InteropServices;

using Office=Microsoft.Office.Core;

using Excel=Microsoft.Office.Interop.Excel;

using Wrapper;

/// <summary>

/// Interface that defines methods that are callable from VBA.

/// </summary>

[GuidAttribute("E24AA65F-8416-4d98-8627-4AC83756209A")]

public interface IConnect

{

/// <summary>

/// Create a new sheet with the Chi-Squared calculation.

/// </summary>

/// <param name="rowStart">The row start value.</param>

/// <param name="rowIncrement">The row increment value.</param>

/// <param name="rows">The number of rows.</param>

/// <param name="columnStart">The column start value.</param>

/// <param name="columnIncrement">The column increment.</param>

/// <param name="columns">The number of columns.</param>

/// <param name="quantileX">The quantile x-value.</param>

/// <param name="sheetName">The name of the sheet to create.</param>

void CreateChiSquaredSheet(double rowStart, double rowIncrement,

int rows, double columnStart, double columnIncrement, int columns,

double quantileX, string sheetName);

}

/// <summary>

/// The object for implementing an Add-in.

/// </summary>

[GuidAttribute("32BBE261-6FE4-4A11-940E-4B864AE0D121"),

ProgId("CsExcelAddIn.Connect")]

public class Connect: Object, Extensibility.IDTExtensibility2, IConnect

{

// Variables

private Excel.Application m\_xl;

private Office.COMAddIn m\_cai;

private Office.CommandBarButton m\_menuItem;

// Constants for menu.

private const string c\_menuName="Tools";

private const string c\_menuItemCaption="Chi-Squared distribution";

private const string c\_menuItemKey="ChiSquared";

/// <summary>

/// Implements the constructor for the Add-in object.

/// Place your initialization code within this method.

/// </summary>

public Connect() {}

/// <summary>

/// Implements OnConnection method of the IDTExtensibility2 interface.

/// Receives notification that the Add-in is being loaded.

/// </summary>

/// <param term='application'>Root object of the host app.</param>

/// <param term='connectMode'>How is the Add-in being loaded.</param>

/// <param term='addInInst'>Object representing this Add-in.</param>

public void OnConnection(object application,

Extensibility.ext\_ConnectMode connectMode,

object addInInst, ref System.Array custom)

{

// Store reference to the Excel application

// Exit if host application is not Excel

m\_xl = application as Excel.Application;

if (m\_xl==null) return;

// If addInInst is the same object as myself then I was loaded as

// Automation add-in instead of COM add-in.

if (addInInst!=this)

{

// Attach myself to the add-in object

// In that way I can call functions of this object from

// VBA using the add-ins collection

m\_cai=addInInst as Office.COMAddIn;

m\_cai.Object=this;

// Install the menu item and event handler

m\_menuItem=AddInUtils.AddMenuItem(m\_xl, m\_cai, c\_menuName,

c\_menuItemCaption, c\_menuItemKey);

m\_menuItem.Click+=new Office.\_CommandBarButtonEvents\_ClickEventHandler(ChiSquaredClick);

}

}

/// <summary>

/// Implements OnDisconnection method of the IDTExtensibility2 interface.

/// Receives notification that the Add-in is being unloaded.

/// </summary>

/// <param term='disconnectMode'>How is the Add-in unloaded.</param>

/// <param term='custom'>Array of params (host app specific).</param>

public void OnDisconnection(Extensibility.ext\_DisconnectMode

disconnectMode, ref System.Array custom)

{

// Remove the menu

AddInUtils.RemoveMenuItem(m\_xl, disconnectMode, c\_menuName,

c\_menuItemCaption);

}

/// <summary>

/// Implements OnAddInsUpdate method of the IDTExtensibility2 interface.

/// Receives notification that the collection of Add-ins has changed.

/// </summary>

/// <param term='custom'>Array of params (host app specific).</param>

public void OnAddInsUpdate(ref System.Array custom)

{

}

/// <summary>

/// Implements OnStartupComplete method of IDTExtensibility2 interface.

/// Receives notification that host application has completed loading.

/// </summary>

/// <param term='custom'>Array of params (host app specific).</param>

public void OnStartupComplete(ref System.Array custom)

{

}

/// <summary>

/// Implements OnBeginShutdown method of the IDTExtensibility2 interface.

/// Receives notification that the host application is being unloaded.

/// </summary>

/// <param term='custom'>Array of params (host app specific).</param>

public void OnBeginShutdown(ref System.Array custom)

{

}

/// <summary>

/// The event handler called when there was a click on our menu item.

/// </summary>

/// <param name="button">The button that genereted the event.</param>

/// <param name="CancelDefault"></param>

private void ChiSquaredClick(Office.CommandBarButton button,

ref bool CancelDefault)

{

// Call the create function with default values

(this as IConnect).CreateChiSquaredSheet(2.0, 1.0, 9, 2.0, 2.0, 5,

0.05, "NCCQT");

}

/// <summary>

/// Create a new sheet with the Chi-Squared calculation.

/// </summary>

/// <param name="rowStart">The row start value.</param>

/// <param name="rowIncrement">The row increment value.</param>

/// <param name="rows">The number of rows.</param>

/// <param name="columnStart">The column start value.</param>

/// <param name="columnIncrement">The column increment.</param>

/// <param name="columns">The number of columns.</param>

/// <param name="quantileX">The quantile x-value.</param>

/// <param name="sheetName">The name of the sheet to create.</param>

void IConnect.CreateChiSquaredSheet(double rowStart, double rowIncrement,

int rows, double columnStart, double columnIncrement, int columns,

double quantileX, string sheetName)

{

// Create row indices

VectorCollectionGenerator<double> dofRows=

new VectorCollectionGenerator<double>();

dofRows.Start=rowStart;

dofRows.Increment=rowIncrement;

dofRows.Size=rows;

Set<double> dofSet=SetCreator.CreateSet(dofRows);

// Create column indices

VectorCollectionGenerator<double> nonCentralParameterColumns=

new VectorCollectionGenerator<double>();

nonCentralParameterColumns.Start=columnStart;

nonCentralParameterColumns.Increment=columnIncrement;

nonCentralParameterColumns.Size=columns;

Set<double> nonCentralParameterSet=

SetCreator.CreateSet(nonCentralParameterColumns);

double r1=dofRows.Start;

double c1=nonCentralParameterColumns.Start;

int nRows=dofRows.Size;

int nColumns=nonCentralParameterColumns.Size;

double incrementRow=dofRows.Increment;

double incrementColumn=nonCentralParameterColumns.Increment;

double cs;

// Make the matrix and fill it

NumericMatrix<double> mat=new NumericMatrix<double>(nRows, nColumns);

for (int r=mat.MinRowIndex; r<=mat.MaxRowIndex; ++r)

{

c1=nonCentralParameterColumns.Start;

for (int c=mat.MinColumnIndex; c<=mat.MaxColumnIndex; ++c)

{

cs=BoostMath.Quantile(new ChiSquaredDistribution(r1), quantileX);

mat[r, c]=

BoostMath.Cdf(new NonCentralChiSquaredDistribution(r1, c1), cs);

c1+=incrementColumn;

}

r1+=incrementRow;

}

// Make it into an associative matrix and send it to Excel

AssocMatrix<double, double, double> myAssocMat=

new AssocMatrix<double, double, double>(dofSet,

nonCentralParameterSet, mat);

AddInUtils.ExportAssocMatrixToExcel(m\_xl, myAssocMat, sheetName);

}

}

}

## Excel sheet VBA macro

' Calculate a new Chi-Squared sheet

Sub CalculateButton\_Click()

On Error Resume Next

' Variable for the current worksheet with the parameters

Dim ws As Excel.Worksheet

' Variable to our COM Add-in.

Dim o As CsExcelAddIn.IConnect

' Variable for the Excel COM Add-in object

Dim ai As COMAddIn

' Get the Add-in object for my Excel COM add-in

ai = Application.COMAddIns.Item("CsExcelAddIn.Connect")

If (Err.Number <> 0) Then

MsgBox("Chi-squared COM add-in not installed")

Exit Sub

End If

' Check if our add-in it is connected (loaded)

If ai.Connect = False Then

MsgBox("Chi-squared COM add-in not loaded")

Exit Sub

End If

' Get the active sheet (the one that called this function)

ws = ThisWorkbook.ActiveSheet

' Get the Chi-Squared add-in object

o = ai.Object

' Call the create Chi-Squared sheet function

Call o.CreateChiSquaredSheet(

ws.Range("rowStart"), ws.Range("rowIncrement"), ws.Range("rows"),

ws.Range("columnStart"), ws.Range("columnIncrement"), ws.Range("columns"),

ws.Range("quantileX"), ws.Range("sheetName"))

End Sub