APL+Win and the R Statistics & Graphics Toolkit
©APLNext, LLC. All Rights Reserved.
3/24/2014 - Page 1

Contents

Summary

- The statistics and graphics features of R are easily accessible from APL+Win using the □cse system function and the R.Net interface to R.

- What is R?

R is a statistics and graphics toolkit. Its origin is the S language developed at Bell Laboratories. R provides a wide variety of statistics and graphics tools. R is available as no-cost, open source software under the terms of the Free Software Foundation's GNU General Public License. R is in wide use by many enterprises both public and private.

- What is R.Net?

R.NET is a .Net interface to the R toolkit. R.Net is composed of .Net assemblies which enable programs written using a .NET Framework programming language, e.g. C#, to collaborate with the R toolkit. R.Net is available as no-cost, open source software under the terms of its BSD license.

- What is the APL+Win CSE?

The APL+Win □cse system function is an interface to the APLNext C# Script Engine (CSE). Using the □cse system function, APL+Win can execute CSE scripts, employing the R.Net assemblies, to use the R toolkit.

- Why interface APL+Win and R?

Certainly any feature of R could be duplicated in APL+Win, assuming that the appropriate time and technical expertise are available to implement the required feature.   R is used by many scientific, research and commercial entities worldwide so it incorporates tested tools for statistics and graphics. From the economic and practical points-of-view, using R for what it can do is a rational decision.

Prerequisites

To run the examples in this document the following components must be installed on the target workstation:

- APL+Win version 14.0.04+, available from APL2000, has its own installer.
- Microsoft .Net Framework 4.5, available from Microsoft at no cost, has its own installer.
- The APLNext C# Script Engine v1.0.102+ (CSE), included in the APL+Win subscription, has its own installer which should be 'run as administrator'.

- R toolkit, available from many sources (e.g. <u>US NIH)</u>, has its own installer which should be 'run as administrator'. Review the <u>installation instructions</u> and the <u>R FAQ</u> for this component. The physical path at which the 32-bit version of the R toolkit has been installed is required to run the examples in this document.

- The .Net assemblies (RDotnet.dll and RDotNet.NativeLibrary.dll) for the R.Net R toolkit which are included in the .zip file containing this document or can be obtained from the <u>R.Net project website</u>. It may be necessary to 'Unblock' these assemblies in their Windows Explorer Properties dialog so that they can be loaded by the CSE.

- The APL+Win 'R Stat&Graph.w3' workspace, which accompanies this document, contains the APL+Win functions used in the examples.

After the above components are installed:

- The examples in this document can be run starting from an APL+Win programmer session.

- The examples in this document are for demonstration purposes only.

- The examples use specific versions of R and R.Net, so modifications to the APL+Win functions may be necessary if subsequent versions of these components are used.

- The location of the APL+Win interpreter executable and the R.Net assemblies using for the examples may need to be modified for your environment. The examples assume that the current folder, i.e. □chdir'', contains the required components.

- In the current implementation of R only one instance of the R engine can be created by an application during an application session.

- To run the examples in this document it is necessary create an instance of the R engine at the start of the APL+Win session (by running the APL+Win 'InitREngine' function) and close it at the end of the APL+Win session (by running the APL+Win 'CloseREngine' function).

Object Model of APL+Win and the R Statistics & Graphics Toolkit

R.Net is a .Net (C#) interface to the R Statistics and Graphics toolkit. From APL+Win the □cse system function creates an instance of the R.Net object so that APL+Win can use the R toolkit with an object-oriented object model.
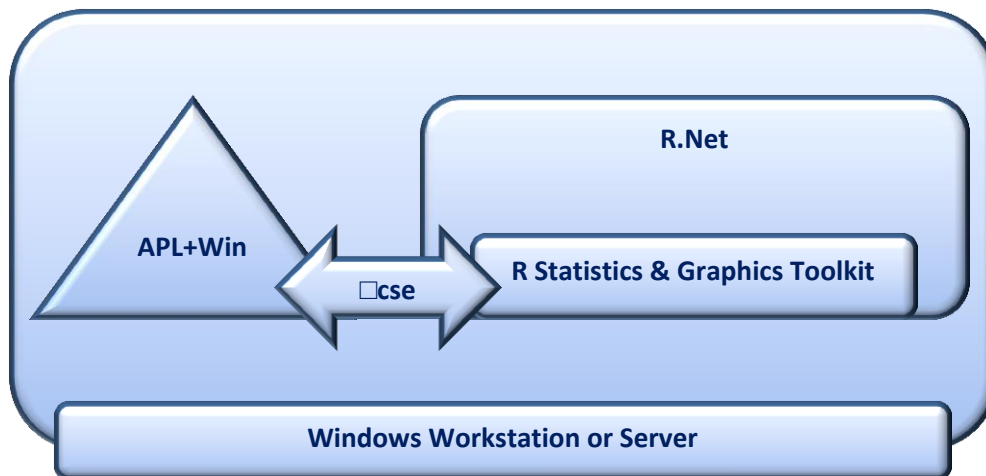
Shared APL+Win Functions Used in the Examples

To initialize an instance of the CSE (with □cself set to 'RNet') and create an instance of the R.Net engine (called 'rEng'), the APL+Win 'InitREngine' function is used. The installation path of the R software is required in this function and the R.Net assemblies are assumed to be in the current folder. The default installation path of the R software is illustrated.

```
InitREngine;sink
:TRY *
 □cself←'RNet'
 sink←□cself □cse 'Init' 'System'
 sink←□cse 'ExecStmt' 'using System;'
 sink←□cse 'returnonerror' 1
 ⍝ ↑Express C# exceptions as APL+Win exceptions

 sink←□cse 'LoadAssemblyByName' 'System.Core'
 sink←□cse 'LoadAssemblyByName' 'System.Data'
 sink←□cse 'LoadAssemblyByName' 'System.Data.DataSetExtensions'
 sink←□cse 'LoadAssemblyByName' 'System.Numerics'
 sink←□cse 'LoadAssemblyByName' 'System.Xml'
 sink←□cse 'LoadAssemblyByName' 'System.Xml.Linq'
 sink←□cse 'LoadAssembly' 'RDotNet.dll'
 sink←□cse 'LoadAssembly' 'RDotNet.NativeLibrary.dll'
 ⍝ ↑.Net assemblies required for this project
 ⍝ ↑ They are assumed to be in the current folder
```

⊙   Alternatively they could be put into the .Net GAC

⊙   and loaded with the ☐cse 'LoadAssemblyByName' method

sink←☐cse 'ExecStmt' 'using System.Collections.Generic;'
sink←☐cse 'ExecStmt' 'using System.Linq;'
sink←☐cse 'ExecStmt' 'using System.Text;'
sink←☐cse 'ExecStmt' 'using System.IO;'
sink←☐cse 'ExecStmt' 'using System.Data;'
sink←☐cse 'ExecStmt' 'using RDotNet;'

⊙ ↑Establish namespace shortcuts

sink←☐cse 'ExecStmt' 'var envPath = Environment.GetEnvironmentVariable("PATH");'
sink←☐cse 'ExecStmt' 'string rBinPath = @"C:\Program Files\R\R-3.0.2\bin\i386";'
sink←☐cse 'ExecStmt' 'Environment.SetEnvironmentVariable("PATH", envPath + Path.PathSeparator + rBinPath);'

⊙ ↑Requires elevated privledges

⊙ ↑R requires that a Win OS Path be defined
⊙ ↑X86 version selected so that R can be used from APL+Win (Win32)

sink←⎕cse 'ExecStmt' 'REngine rEng =REngine.CreateInstance("APLNextR");'
sink←⎕cse 'ExecStmt' 'rEng.Initialize();'
⊙ ↑Create an instance of RDOTNOT.REngine class called rEng
:CATCHALL
 ⎕error ('InitREngine failed: ',,⎕dm)
:ENDTRY

To close the R.Net engine instance and the CSE instance, the APL+Win 'CloseREngine' function is used.

```
CloseREngine;sink
:TRY
 :IF 0<1↑ρ⎕cself ⎕cse 'self'
  sink←⎕cse 'ExecStmt' 'rEng.Close();'
  sink←'C' ⎕cse 'Close'
 :ENDIF
:CATCHALL
:ENDTRY
```

Creating, Saving and Displaying an R-plot in APL+Win

This example illustrates saving an R-plot to an image file in the bitmap format and displaying that image on an APL+Win □wi form. Run the APL+Win 'ShowRPlotInAPLWinForm' function to see the results. This function uses the REngine instance to create an R-plot of the Gauss probability density as a bitmap file and uses that bitmap file to display the R-plot image in an APL+Win form.

Instead of directly computing the Gauss probability density function for a selection of values and using the R 'plot(,)' function, it would also be possible to use the R-function 'rnorm(#vals, mean, stddev)' to generate samples of a Gauss-distributed random variable and then use the R 'hist()' function to generate a similar chart.

```
ShowRPlotInAPLWinForm;F;P;sink
⍝APLNext 20131229
:TRY *
 sink←⎕cse 'ExecStmt' 'double[] x = new double[200];'
 sink←⎕cse 'SetValue' 'x' (0.04×(⍳200)-100)
 ⍝ ↑Create x-axis elements for the plot using APL+Win
 ⍝ ↑Set the value of the C# variable 'x' from APL+Win
 sink←⎕cse 'ExecStmt' 'rEng.SetSymbol("x", rEng.CreateNumericVector(x));'
 ⍝ ↑Set R symbol "x"

 sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"y <- 1/sqrt(2*pi)*exp(-x^2/2)");'
 ⍝ ↑Using R, compute the Gauss probability density function for each x-axis
element
 ⍝  and assign the results to y

 sink←⎕cse 'ExecStmt' 'string chartFnm = @"chartFnm.bmp";'
 sink←⎕cse 'ExecStmt' 'File.Delete(chartFnm);'
```

```
 sink←⎕cse                                                              'ExecStmt'
'rEng.SetSymbol("chartFnm",rEng.CreateCharacterVector(new
string[]{chartFnm}));'
 sink←⎕cse 'ExecStmt' 'rEng.Evaluate("bmp(chartFnm)");'
 ⍝ ↑Set the saved chart bitmap destination

 sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"plot(x,y,type=""l"")");'
 ⍝ ↑Create the line plot using R
 sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"dev.off()");'
 ⍝ ↑Write the chart to the target bitmap file

 F←'F'⎕wi 'Create' 'Form' ('size' 40 80) ('caption' 'R-plot in APL+Win Form')
 P←'F.P'⎕wi 'Create' 'Picture' ('where' 0 0 40 80) ('imagesize' 38 78) ('style' 4)
 P⎕wi 'bitmap' 'chartFnm.bmp'
 sink←F⎕wi 'Wait'
 ⍝ ↑Display the chart in a modal APL+Win form

:CATCH
```
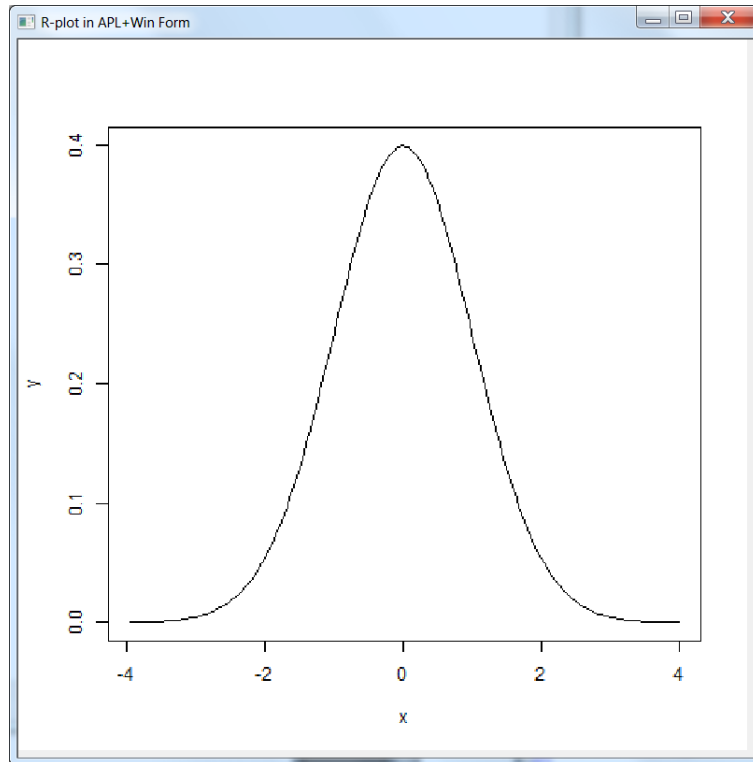
```
 'ShowRPlotInAPLWinForm failed: ',,⎕dm


:ENDTRY
```

To keep this example simple, the format of the illustrated plot is elementary. With some study of the R documentation for plotting and graphics much more sophisticated charts can be prepared.

R-Plot illustrating the Central Limit Theorem

In this example two R-plots are placed in the same chart. The first is a histogram of sample averages and the second is a line chart of the same averages to illustrate the anticipated effect of the Central Limit Theorem.

```
CentralLimitChart;F;P;sink;trialVals
 ⍝APLNext 20140222
:TRY *
 sink←⎕cse 'ExecStmt' 'string chartFnm = @"chartFnm.bmp";'
 sink←⎕cse 'ExecStmt' 'File.Delete(chartFnm);'
 sink←⎕cse                                          'ExecStmt'
'rEng.SetSymbol("chartFnm",rEng.CreateCharacterVector(new
string[]{chartFnm}));'
 sink←⎕cse 'ExecStmt' 'rEng.Evaluate("bmp(chartFnm)");'
 ⍝ ↑Set the saved chart bitmap destination

 trialVals←1E¯4×16 1000⍴?¨(×/16 1000)⍴×/16 1000
 ⍝ ↑In a production environment, trialVals would come from a simulation or
other experiment involving trials of a model
 sink←⎕cse 'ExecStmt' 'double[,] trialVals = new double[16,1000];'
 ⍝ ↑Create C# matrix to receive APL+Win values
 sink←⎕cse 'SetValue' 'trialVals' trialVals
```

◎ ↑Set C# matrix with the APL+Win values
sink←⎕cse 'ExecStmt' 'var rmt = rEng.CreateNumericMatrix(trialVals);'
◎ ↑Create an R.Net variable containing the APL+Win values
sink←⎕cse 'ExecStmt' 'rEng.SetSymbol("rmt",rmt);'
◎ ↑Create a R symbol containing the APL+Win values

sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"mns <- cbind(rmt[1,],apply(rmt[1:4,], 2, mean),apply(rmt[1:16,], 2, mean))");'
◎ ↑Compute the means of samples of 1, 4, 16
sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"meds <- cbind(rmt[1,],apply(rmt[1:4,], 2, median),apply(rmt[1:16,], 2, median))");'
◎ ↑Compute the medians of samples of 1, 4, 16
sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"hist(mns[,3], main = ""Means of samples of size 16"",xlab = ""Size 16 means"", las = 1, col = ""darkred"", breaks = 50, prob = TRUE)");'
sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"lines(density(mns[,3]), col = ""blue"")");'
◎ ↑Define the R-plot type, axis labels, colors, etc.

```
sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"dev.off()");'
 ⍝ ↑Write the chart to the target bitmap file

F←'F'⎕wi 'Create' 'Form' ('size' 40 80) ('caption' 'R-plot in APL+Win Form')
P←'F.P'⎕wi 'Create' 'Picture' ('where' 0 0 40 80) ('size' 30 60) ('style' 0)
P⎕wi 'bitmap' 'chartFnm.bmp'
sink←F⎕wi 'Wait'
 ⍝ ↑Display the chart in a modal APL+Win form

:CATCH
 'CentralLimitChart failed: ',,⎕dm
:ENDTRY
```
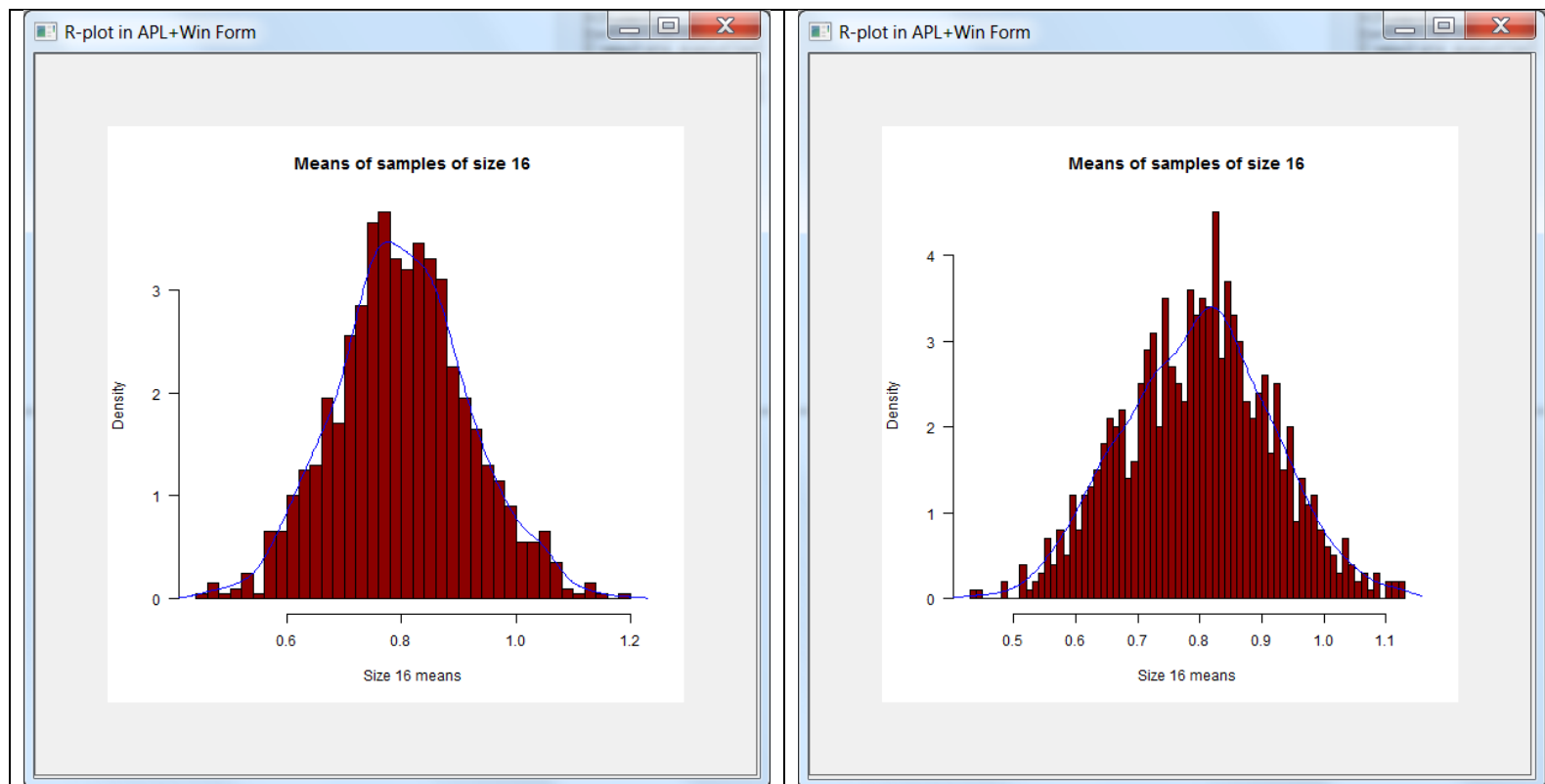
When the APL+Win 'CentralLimitChart' function is run, an APL+Win form is displayed containing the JPEG-format image containing the R-plot. Since the 'trial' data used in this example is based on the APL+Win 'deal' function, running this function a few times gives a graphic representation of the APL+Win random number generator results.

R-Plot of Beta Distribution

The beta distribution has two parameters α and β which provide flexibility to specify an asymmetrical probability distribution with mean $1+(1/(\beta/\alpha))$. The APL+Win 'BetaDistChart' function has a three element right argument (#values α β). This function uses the R-function 'rbeta(#vals, α, β)' to generate n sample values of a beta-distributed random variable and then plots these values using the R-function 'hist()'.

BetaDistChart X;F;P;sink;trialVals;n;a;b
⍝APLNext 20140226
⍝ X[1]: #vals
⍝ X[2]: alpha
⍝ X[3]: beta

(n a b)←X
:TRY *
 sink←☐cse 'ExecStmt' 'string chartFnm = @"chartFnm.bmp";'
 sink←☐cse 'ExecStmt' 'File.Delete(chartFnm);'
 sink←☐cse 'ExecStmt'
'rEng.SetSymbol("chartFnm",rEng.CreateCharacterVector(new
string[]{chartFnm}));'
 sink←☐cse 'ExecStmt' 'rEng.Evaluate("bmp(chartFnm)");'
 ⍝ ↑Set the saved chart bitmap destination

 sink←☐cse 'ExecStmt' 'double[] n = new double[1];'

```
sink←☐cse 'SetValue' 'n[0]' n
sink←☐cse 'ExecStmt' 'rEng.SetSymbol("n",rEng.CreateNumericVector(n));'

sink←☐cse 'ExecStmt' 'double[] a = new double[1];'
sink←☐cse 'SetValue' 'a[0]' a
sink←☐cse 'ExecStmt' 'rEng.SetSymbol("a",rEng.CreateNumericVector(a));'

sink←☐cse 'ExecStmt' 'double[] b = new double[1];'
sink←☐cse 'SetValue' 'b[0]' b
sink←☐cse 'ExecStmt' 'rEng.SetSymbol("b",rEng.CreateNumericVector(b));'


sink←☐cse 'ExecStmt' 'rEng.Evaluate(@"b <- rbeta(n,a,b)");'
sink←☐cse 'ExecStmt' 'rEng.Evaluate(@"hist(b, prob = TRUE, breaks=50, col =
""darkred"")");'
⍝ ↑Define the R-plot type, axis labels, colors, etc.


sink←☐cse 'ExecStmt' 'rEng.Evaluate(@"dev.off()");'
```

⍝ ↑Write the chart to the target bitmap file

F←'F'⎕wi 'Create' 'Form' ('size' 40 80) ('caption' 'R-plot in APL+Win Form')
P←'F.P'⎕wi 'Create' 'Picture' ('where' 0 0 40 80) ('size' 30 60) ('style' 0)
P⎕wi 'bitmap' 'chartFnm.bmp'
sink←F⎕wi 'Wait'
⍝ ↑Display the chart in a modal APL+Win form
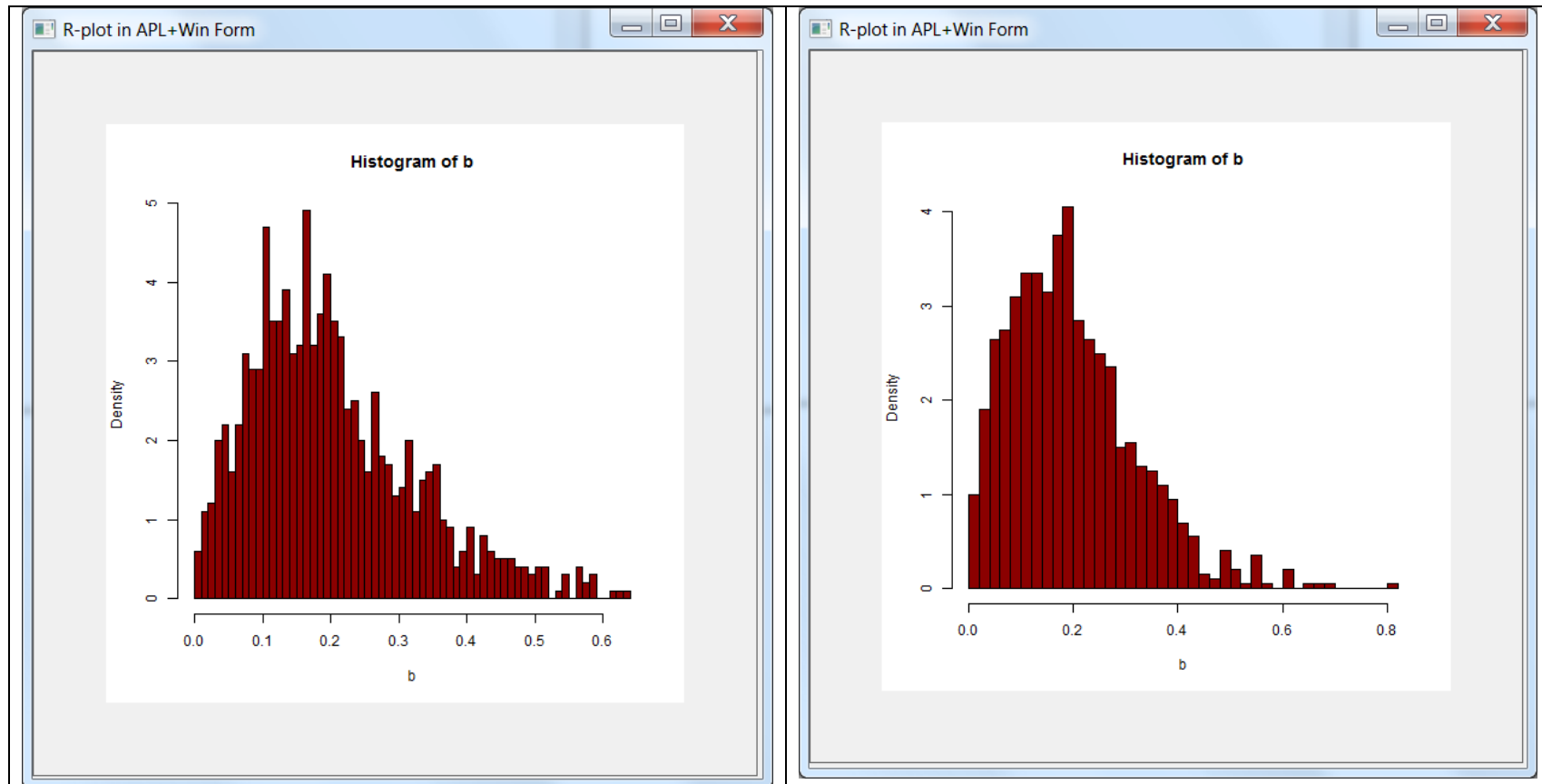
:CATCH
'CentralLimitChart failed: ',,⎕dm
:ENDTRY

Sample R-plots (mean 0.2) using a right argument of 1000 2 8 for the APL+Win 'BetaDIstChart' function.

Using R for *ad hoc* Calculations in APL+Win

This example illustrates creating an APL+Win GUI form application which requests user input, responds to the user's button click event and uses the R Engine to perform an *ad hoc* calculation. The APL+Win 'UseRToCalculateAdHoc' function defines the APL+Win GUI form with Edit controls for input and output Edit and a Button control, subscribed to the Button click event with the 'UseRToCalculateAdHocEH' function and waits for user input.

```
UseRToCalculateAdHoc;F;LI;EE;EI;BN;LO;EO;sink
⍝APLNext 20131215
:TRY

 F←'F'⎕wi 'Create' 'Form' ('size' 20 80) ('caption' 'Use R for ad hoc calculation')
 LI←'F.LI'⎕wi 'Create' 'Label' ('where' 1 0) ('caption' 'Enter Radius: ')
 EI←'F.EI'⎕wi 'Create' 'Edit' ('where' 1 20 2 20) ('text' '1')
 BN←'F.BN'⎕wi 'Create' 'Button' ('where' 4 2 2 25) ('caption' 'Calculate using
APL+Win and R')
 BN⎕wi 'onClick' 'UseRToCalculateAdHocEH'
 LO←'F.LO'⎕wi 'Create' 'Label' ('where' 8 0) ('caption' 'Area of Circle: ')
 EO←'F.EO'⎕wi 'Create' 'Edit' ('where' 8 20 2 20)
 EE←'F.EE'⎕wi 'Create' 'Edit' ('where' 12 10 5 50) ('style' 4 16)
 sink←F ⎕wi 'Wait'
 ⍝ ↑Define an APL+Win GUI form, set the button click event handler, wait for
user
 sink←F ⎕wi 'Close'
```

```
:CATCHALL
 'UseRToCalculateAdHoc failed: ',,⎕dm
:ENDTRY
```

The 'UseRToCalculateAdHocEH' event handler function obtains the user input, passes it to the R.Net engine instance, requests the calculation from the R Engine, gets the result into APL+Win and displays the result on the APL+Win GUI form.

UseRToCalculateAdHocEH;radius;area
⍝APLNext 20131215

:TRY
 'F.BN'☐wi 'enabled' 0
 ⍝ ↑Disable user input during event handler function execution
 radius←1↑☐FI 'F.EI'☐wi 'text'
 ⍝ ↑Get the user input from the APL+WIN form

 sink←☐cse 'ExecStmt' 'double radius = 0;'
 sink←☐cse 'SetValue' 'radius' (1⊃radius,0.5)
 ⍝ ↑Set C# input variable from the APL+Win variable coerced to double in APL+Win
 sink←☐cse   'ExecStmt'   'rEng.SetSymbol("radius", rEng.CreateNumericVector(new double[]{radius}));'
 ⍝ ↑Set R symbol "radius"

```
 sink←⎕cse 'ExecStmt' 'rEng.Evaluate(@"area <- pi*(radius^2)");'
 ⍝ ↑Using R, compute the area of the circle and assign the value to the R-
symbol "area"


 area←⎕cse 'GetValue' 'rEng.GetSymbol("area").AsNumeric().First()'
 ⍝ ↑Get the result from R


 'F.EO'⎕wi 'text' (⍕ area)
 ⍝ ↑Display result in APL+Win form


:CATCH
 'F.EE'⎕wi 'text' ('Error: ',,⎕dm)


:FINALLY
 'F.BN'⎕wi 'enabled' 1
:ENDTRY
```
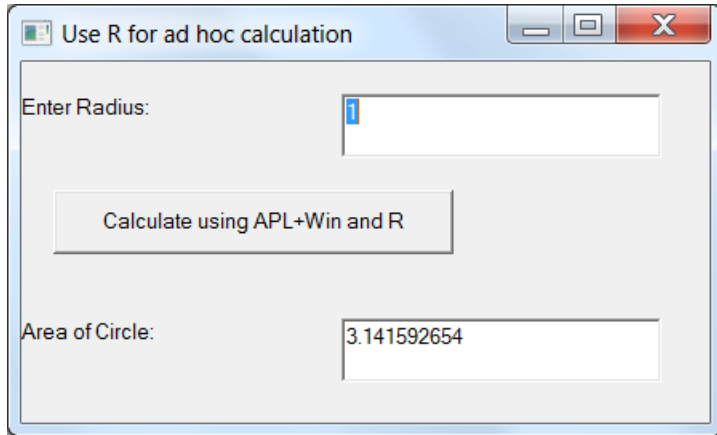
After the 'UseRToCalclulateAdHoc' function is run, the user enters a radius value and clicks the 'Calculate...' button:

Creating and Executing a user-defined function in R with a scalar argument and result

Sometimes it is convenient to create a user-defined function in R so that it can be executed several times during the APL+Win session.

- Use the R.Net engine to evaluate a valid R function definition and cast it as an R.Net 'Function'

- Create the arguments for the function as an R.Net 'SymbolicExpression[]'

- Use the R.Net 'Invoke' method to execute the R.Net function

RUserDefinedFunction

☐ cse 'ExecStmt' 'var Add_10 = rEng.Evaluate(@"Add_10 <-function(x){x = x + 10;return(x);}").AsFunction();'

☐ cse 'ExecStmt' 'var I = rEng.CreateIntegerVector(new Int32[] { 100 });'

☐ cse 'ExecStmt' 'var se = new SymbolicExpression[] { I };'

☐ cse 'ExecStmt' 'var res = Add_10.Invoke(se).AsInteger();'

☐ cse 'ExecStmt' 'Int32 r = res.First();'
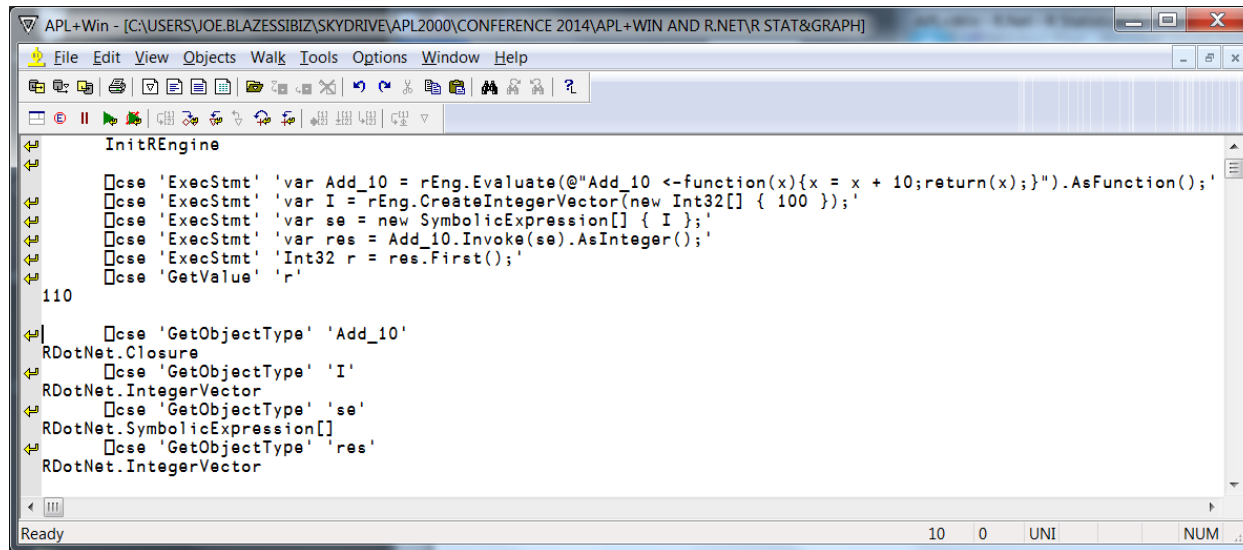
☐ cse 'GetValue' 'r'

☐ cse 'GetObjectType' 'Add_10'
☐ cse 'GetObjectType' 'I'

⎕cse 'GetObjectType' 'se'
⎕cse 'GetObjectType' 'res'

When the APL+Win 'RUserDefinedFunction' function is run the 'Add_10 (100)' R-function is executed and the result captured in APL+Win.

Creating an Executing a user-defined function in R with an array argument and result

In this example the user-defined R function will return a vector of values based upon arguments provided by APL+Win. Because the C# 'for{…}' control structure is used, a CSE script is created and run using the CSE 'Exec' method. The R-function 'runif' is used to generate n sample values of the uniform distribution over the interval [min, max].

```
Z←RUserDefinedFunctionArrayResult X;S;N;MI;MX
 ⍝ N : #Uniform distribution values to return
 ⍝ MI: Minimum value of the range of uniform distribution values
 ⍝ MX: Maximum value of the range of uniform distribution values

(N MI MX)←X
X←⍬
☐cse 'ExecStmt' 'Int32 N;'
☐cse 'SetValue' 'N' N
☐cse 'ExecStmt' 'Int32 MI;'
☐cse 'SetValue' 'MI' MI
☐cse 'ExecStmt' 'Int32 MX;'
☐cse 'SetValue' 'MX' MX

S←⊂'var    runif   =    rEng.Evaluate(@"RUNIF<-function(n,mn,mx){res    =
runif(n,min=mn,max=mx);return(res);}").AsFunction();'
S←S,⊂'var nArg = rEng.CreateIntegerVector(new Int32[]{N});'
```

```
S←S,⊂'var mnArg = rEng.CreateIntegerVector(new Int32[]{MI});'
S←S,⊂'var mxArg = rEng.CreateIntegerVector(new Int32[]{MX});'
S←S,⊂'var se = new SymbolicExpression[] { nArg, mnArg, mxArg };'
S←S,⊂'var res = runif.Invoke(se).AsNumeric();'
S←S,⊂'double[] dRes = new double[res.Count()];'
S←S,⊂'for (Int32 I = 0;I<res.Count();I++)'
S←S,⊂'{'
S←S,⊂'  dRes[I] = res[I];'
S←S,⊂'}'

S←⊃S
⎕cse 'Exec' S
Z←⎕cse 'GetValue' 'dRes'
```

Here is the CSE script which this APL+Win function creates:

```
var      runif     =      rEng.Evaluate(@"RUNIF<-function(n,mn,mx){res      =
runif(n,min=mn,max=mx);return(res);}").AsFunction();
var nArg = rEng.CreateIntegerVector(new Int32[]{N});
```

```
var mnArg = rEng.CreateIntegerVector(new Int32[]{MI});
var mxArg = rEng.CreateIntegerVector(new Int32[]{MX});
var se = new SymbolicExpression[] { nArg, mnArg, mxArg };
var res = runif.Invoke(se).AsNumeric();
double[] dRes = new double[res.Count()];
for (Int32 I = 0;I<res.Count();I++)
{
  dRes[I] = res[I];
}
```

When the APL+Win 'RUserDefinedFunctionArrayResult' function is run three times, 10 pseudo-random values of the uniform distribution over the interval [1, 5] are provided by the R engine for each execution of the function:

Programming Styles using R, R.Net Toolkit and APL+Win

The ensemble of R, the R.Net toolkit and APL+Win supports several programming styles. The R.Net <u>toolkit documentation</u> describes the first three of these programming styles. For simplicity the R cos() trigonometric function used, even though it is available in APL+Win and APL+Win merely displays the values obtained from R. In a production application R functions which are not available or convenient in C# or APL+Win would be used and the results obtained from R would be further manipulated by C# or APL+Win.

In the examples in this section it is important to note the methods by which APL+Win values are assigned to R.Net and R variables and analogously how the values of R.Net and R variables are assigned to APL+Win variables.

R variables are associated with symbols (e.g. text variable names). R variables have simple types analogous to vectors (rank 1) and matrices (rank 2).

The R.Net toolkit for R provides .Net object types such as Function, NumericVector, NumericMatrix (rank 2), CharacterVector, CharacterMatrix (rank 2), which correspond to the R variable types.   The R.Net tool kit for R also provides methods to get and set the value of R variables such as CreateNumericVector(double[]),   GetSymbol(string).AsNumeric(), SetSymbol(string, NumericVector).

The R.Net object types can be coerced into arrays of .Net value types such as double[], double[,], bool[].   For example the R.Net NumericVector can be converted in C# to a double[] using C# generic extension methods, i.e. NumericVector.ToArray<double>().

The APL+Win ⎕cse system function 'GetValue' and 'SetValue' methods can be used to get and set the value of .Net variables that have been derived from R.Net variables using APL+Win variable values.

R.Net style #1

This style uses R expressions to perform calculations and prepare results exclusively in R. The advantage of this style is that R sample code can generally be used directly.

The APL+Win 'RNetStyle1' function illustrates this style:

```
RNetStyle1;x;y
⍝APLNext 20131229
:TRY
'Style #1 - Do as much as possible in R'
□cse 'ExecStmt' 'rEng.Evaluate(@"x <- (0:12) * pi / 12");'
⍝ ↑Create the R symbol 'x' containing the values at which
⍝  the function will be evaluated

□cse 'ExecStmt' 'rEng.Evaluate(@"y <- cos(x)");'
⍝ ↑Evaluate the cos() function at desired values and
⍝  assign the results to the R symbol 'y'

x←□cse 'GetValue' 'rEng.Evaluate("x").AsNumeric().ToArray<double>();'
y←□cse 'GetValue' 'rEng.Evaluate("y").AsNumeric().ToArray<double>();'
x,[1.5]y
⍝ ↑Get the R-calculated values into APL+Win
```
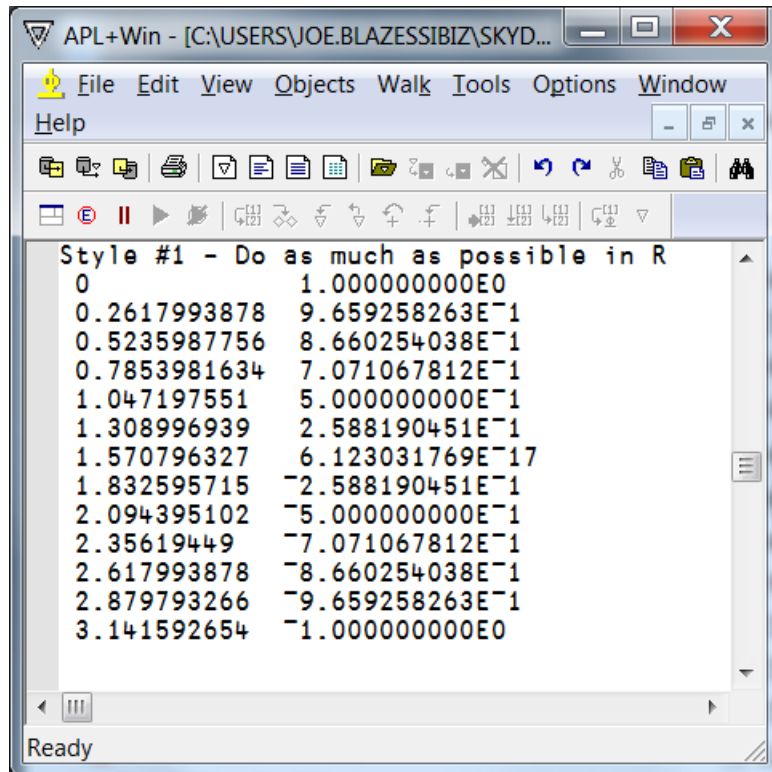
```
:CATCH
 'RNetStyle1: ',,⎕dm
:ENDTRY
```

When this function is executed, the results are available in APL+Win


```
 APL+Win - [C:\USERS\JOE.BLAZESSIBIZ\SKYD...
 File  Edit  View  Objects  Walk  Tools  Options  Window
 Help

Style #1 - Do as much as possible in R
 0                1.000000000E0
 0.2617993878     9.659258263E¯1
 0.5235987756     8.660254038E¯1
 0.7853981634     7.071067812E¯1
 1.047197551      5.000000000E¯1
 1.308996939      2.588190451E¯1
 1.570796327      6.123031769E¯17
 1.832595715     ¯2.588190451E¯1
 2.094395102     ¯5.000000000E¯1
 2.35619449      ¯7.071067812E¯1
 2.617993878     ¯8.660254038E¯1
 2.879793266     ¯9.659258263E¯1
 3.141592654     ¯1.000000000E0

Ready
```

R.Net Style #2

This style uses R expressions to perform calculations and prepare results which can only be done in R and otherwise uses C# expressions supported by the R.Net toolkit for R whenever that is more convenient or familiar.

The APL+Win 'RNetStyle2' function illustrates this style:

RNetStyle2;x;y

⍟APLNext 20131229

:TRY

'Style #2 - Mostly R Expressions, but use C# whenever convenient or familiar'

☐cse 'ExecStmt' 'var x = rEng.Evaluate(@"x <- (0:12) * pi / 12").AsNumeric();'

⍟ ↑Create the C# variable x containing the values at which the function will

⍟ be evaluated. The variable x has R.Net.NumericVector type.

☐cse 'ExecStmt' 'rEng.SetSymbol("x", x);'

⍟ ↑Set the value of the R symbol "x" from the value of the R.Net C# yariable x

☐cse 'ExecStmt' 'var y = rEng.Evaluate("cos(x)").AsNumeric();'

⍟ ↑Create the R.Net C# variable 'y' containing the results of the cos() function

⍟ at the desired values. The variable x has R.Net.NumericVector type.

x←☐cse 'GetValue' 'x.ToArray<double>();'

y←☐cse 'GetValue' 'y.ToArray<double>();'

```
x,[1.5]y
⍝ ↑Get the R-calculated values into APL+Win

:CATCH
 'RNetStyle2: ',,⎕dm
:ENDTRY
```

When the APL+Win 'RNetStyle2' function is executed the same results as the APL+Win 'RNetStyle1' function are produced.

R.Net Style #3

This style increases the abstraction of R syntax using more of the R.Net toolkit features. This style has the advantage that the resulting C# methods can be re-used and composed with other methods in the same .Net project. This style also continues to use C# whenever it is more convenient or familiar.

The APL+Win 'RNetStyle3' function illustrates this style:

RNetStyle3;x;y

⍝APLNext 20131229

:TRY

'Style #3 - C# Abstraction of R syntax using the R.Net toolkit'

☐cse 'ExecStmt' 'var x = rEng.CreateNumericVector(Enumerable.Range(0, 13).Select(i => i * Math.PI / 12).ToArray());'

⍝ ↑Create the C# variable (R.Net.NumericVector type) x containing the values at which the function will be evaluated.

☐cse 'ExecStmt' 'var cos = rEng.GetSymbol("cos").AsFunction();'

⍝ ↑Create the C# variable (R.Net.Function type) cos which encapsulates the R cos() function.

☐cse 'ExecStmt' 'var y = cos.Invoke(new[] { x }).AsNumeric();'

⍝ ↑Create the C# variable (R.Net.NumericVector type) containing the results of the evaluation by using the

⍝ R.Net.Invoke() method on the R.Net.Function variable cos.

```
x←□cse 'GetValue' 'x.ToArray<double>();'
y←□cse 'GetValue' 'y.ToArray<double>();'
x,[1.5]y
⊙ ↑Get the R-calculated values into APL+Win

:CATCH
 'RNetStyle3: ',,□dm
:ENDTRY
```

When the APL+Win 'RNetStyle3' function is executed the same results as the
APL+Win 'RNetStyle1' or 'RNetStyle2' functions are produced.

R.Net and APL+Win Style #4

This programming style is similar to style #3 except that APL+Win is used whenever it is more convenient or familiar.

The APL+Win 'RNetAndAPLStyle4' function illustrates this style:

```
RNetAndAPLStyle4;x;y
⍝APLNext 20131231
⎕io←1
:TRY
'Style #4 - Use APL+Win whenever more convenient or familiar'
⎕cse 'ExecStmt' 'double[] dV;'
x←○(¯1+⍳13)÷12 ⍝Use APL+Win to generate the required values
⎕cse 'SetValue' 'dV' x ⍝Set the C# variable dV using the APL+Win values
⎕cse 'ExecStmt' 'var x = rEng.CreateNumericVector(dV);'
⍝ ↑Create the C# variable (R.Net.NumericVector type) x containing the values
at which the function will be evaluated.

⎕cse 'ExecStmt' 'var cos = rEng.GetSymbol("cos").AsFunction();'
⍝ ↑Create the C# variable (R.Net.Function type) cos which encapsulates the R
cos() function.

y←⎕cse 'GetValue' 'cos.Invoke(new[] { x }).AsNumeric().ToArray<double>();'
```

⍝ ↑Get the R-calculated values into APL+Win using the R.Net.Invoke() method on the R.Net.Function variable cos.

x,[1.5]y ⍝Display or use the values in APL+Win

:CATCH
 'RNetStyle4: ',,⎕dm
:ENDTRY

When this function is executed the same results are obtained as the functions illustrating the other styles.

Utility functions in the APL+Win 'R Stat& Graph.w3' workspace

The APL+Win 'R Stat&Graph.w3' workspace contains the APL+Win 'Plot' function which makes 2-D R-plots easy by separating the generation of values from the mechanics of creating an R-plot.

```
Plot X;xyVals;xLAB;yLAB;mainLAB
⍝ xyVals[;1]: x-axis values
⍝ xyVals[;2]: y-axis values
⍝ xLAB     : text x-axis label
⍝ yLAB     : text y-axis label
⍝ mainLAB   : chart title

(xyVals xLAB yLAB mainLAB)←X
X←θ
sink←⎕cse 'ExecStmt' 'string chartFnm = @"chartFnm.bmp";'
sink←⎕cse 'ExecStmt' 'File.Delete(chartFnm);'
sink←⎕cse                                               'ExecStmt'
```

```
'rEng.SetSymbol("chartFnm",rEng.CreateCharacterVector(new
string[]{chartFnm}));'
sink←⎕cse 'ExecStmt' 'rEng.Evaluate("bmp(chartFnm)");'
⍝ ↑Set the saved chart bitmap destination

sink←⎕cse 'ExecStmt' 'double[,] xyVals = new double[,]{};'
sink←⎕cse 'SetValue' 'xyVals' xyVals
sink←⎕cse                                                    'ExecStmt'
'rEng.SetSymbol("xyVals",rEng.CreateNumericMatrix(xyVals));'

sink←⎕cse 'ExecStmt' 'string xLAB = "";'
sink←⎕cse 'SetValue' 'xLAB' xLAB
sink←⎕cse                                                    'ExecStmt'
'rEng.SetSymbol("xLAB",rEng.CreateCharacterVector(new string[]{xLAB}));'

sink←⎕cse 'ExecStmt' 'string yLAB = "";'
sink←⎕cse 'SetValue' 'yLAB' yLAB
sink←⎕cse                                                    'ExecStmt'
```

```
'rEng.SetSymbol("yLAB",rEng.CreateCharacterVector(new string[]{yLAB}));'

sink←☐cse 'ExecStmt' 'string mainLAB = "";'
sink←☐cse 'SetValue' 'mainLAB' mainLAB
sink←☐cse                                                        'ExecStmt'
'rEng.SetSymbol("mainLAB",rEng.CreateCharacterVector(new
string[]{mainLAB}));'

sink←☐cse  'ExecStmt'  'rEng.Evaluate(@"plot(xyVals[,c(1)],xyVals[,c(2)],  col  =
""darkred"", xlab=xLAB, ylab=yLAB, main=mainLAB)");'
sink←☐cse 'ExecStmt' 'rEng.Evaluate(@"dev.off()");'

F←'F'☐wi 'Create' 'Form' ('size' 40 80) ('caption' 'R-plot in APL+Win Form')
P←'F.P'☐wi 'Create' 'Picture' ('where' 0 0 40 80) ('size' 30 60) ('style' 0)
P☐wi 'bitmap' 'chartFnm.bmp'
sink←F☐wi 'Wait'
```

The APL+Win 'Gauss' and 'Unif' functions can be used to generate pseudo-random samples from the normal and uniform probability distributions respectively.

```
Z←Gauss X;n;m;s;sink
⍝ X[1]: #estimates desired
⍝ X[2]: mean of Gauss distribution
⍝ X[3]: standard deviation of Gauss distribution
⍝ Z  : Vector of estimates

(n m s)←X
sink←□cse 'ExecStmt' 'double mGauss = 0;'
sink←□cse 'SetValue' 'mGauss' m
sink←□cse 'ExecStmt'
'rEng.SetSymbol("mGauss",rEng.CreateNumericVector(new
double[]{mGauss}));'


sink←□cse 'ExecStmt' 'double sGauss = 0;'
```

```
sink←□cse 'SetValue' 'sGauss' s
sink←□cse 'ExecStmt'
'rEng.SetSymbol("sGauss",rEng.CreateNumericVector(new double[]{sGauss}));'

sink←□cse 'ExecStmt' 'Int32 nGauss = 0;'
sink←□cse 'SetValue' 'nGauss' n
sink←□cse 'ExecStmt'
'rEng.SetSymbol("nGauss",rEng.CreateIntegerVector(new Int32[]{nGauss}));'

Z←□cse 'GetValue'
'rEng.Evaluate("rnorm(nGauss,mGauss,sGauss)").AsNumeric().Take(nGauss).To
Array();'
```

Z←Unif X;n;max;min;sink

⍝ X[1]: #estimates desired

⍝ X[2]: minimum value of the range of estimates

⍝ X[3]: maximum value of the range of estimates

⍝ Z : Vector of estimates

(n min max)←X

sink←☐cse 'ExecStmt' 'Int32[] nUnif = new Int32[1];'

sink←☐cse 'SetValue' 'nUnif[0]' n

sink←☐cse 'ExecStmt'
'rEng.SetSymbol("nUnif",rEng.CreateIntegerVector(nUnif));'

sink←☐cse 'ExecStmt' 'double[] minUnif = new double[1];'

sink←☐cse 'SetValue' 'minUnif[0]' min

sink←☐cse 'ExecStmt'
'rEng.SetSymbol("minUnif",rEng.CreateNumericVector(minUnif));'

sink←☐cse 'ExecStmt' 'double[] maxUnif = new double[1];'

sink←⎕cse 'SetValue' 'maxUnif[0]' max
sink←⎕cse 'ExecStmt'
'rEng.SetSymbol("maxUnif",rEng.CreateNumericVector(maxUnif));'

Z←⎕cse 'GetValue' 'rEng.Evaluate("runif(nUnif, min=minUnif, max =
maxUnif)").AsNumeric().Take((Int32)(nUnif[0])).ToArray();'

R For Actuaries

A few interesting links about the use of R in actuarial mathematics:

- http://toolkit.pbworks.com/f/R%20Examples%20for%20Actuaries%20v0.1-1.pdf
- http://toolkit.pbworks.com/w/page/22358245/R%20Examples%20For%20Actuaries
- http://www.r-bloggers.com/sampling-for-monte-carlo-simulations-with-r/
- http://www.springer.com/statistics/computational+statistics/book/978-1-4419-1575-7

Resources

- APL2000 Consulting Services

- R-Project.org

- Comprehensive R Archive Network

- R.Net

- Beta distribution in R

  - ○

- Rclr

- Some examples:

  - http://psychwire.wordpress.com/2011/06/25/importing-and-displaying-a-data-frame-with-c-and-r-net/

  - http://www.codeproject.com/Articles/25819/The-R-Statistical-Language-and-C-NET-Foundations

  - http://msenux.redwoods.edu/math/R/normal.php

  - http://www.ats.ucla.edu/stat/r/library/lecture_graphing_r.htm

- http://rtutorialseries.blogspot.com
- http://www.stat.berkeley.edu/classes/s133/saving.html
- https://github.com/jmp75/rdotnet-onboarding
- http://csg.sph.umich.edu/docs/R/graphics-1.pdf
- Bing or Google: "plotting in R stat"