Adding Numerical Functionality to LabVIEW Using the NAG Library

Jeremy Walton

The Numerical Algorithms Group
Oxford, UK

nag® Experts in numerical algorithms and HPC services

http://www.nag.co.uk/

Overview

- An introduction to NAG
  - NAG numerical libraries
- NAG and LabVIEW
  - how to call NAG routines from LabVIEW
    - NAG Library for .NET
    - NAG Fortran Library, NAG C Library
    - using a wrapper library
- Conclusions
  - finding out more

Overview

- An introduction to NAG
  - NAG numerical libraries
- NAG and LabVIEW
  - how to call NAG routines from LabVIEW
    - NAG Library for .NET
    - NAG Fortran Library, NAG C Library
    - using a wrapper library
- Conclusions
  - finding out more

NAG's products and users

- Products
  - Mathematical, statistical, data analysis components
    - NAG Numerical libraries
  - Fortran compiler and Windows IDE
  - HPC software engineering services
    - HECToR support
  - Consultancy work for bespoke application development
- Users
  - Academic researchers
  - Professional developers
  - Analysts / modelers

The NAG Numerical libraries

- Contain mathematical & statistical components
  - ~ 1700 of them
- Available on variety of different platforms
  - stringently tested
- Comprehensive documentation
- Used as building blocks by package builders
  - since 1971
  - gives reduced development time
  - allows you to concentrate on areas of expertise
  - interfaces to various environments

NAG Library Contents

- Root Finding
- Summation of Series
- Quadrature
- Ordinary Differential Equations
- Partial Differential Equations
- Numerical Differentiation
- Integral Equations
- Mesh Generation
- Interpolation
- Curve and Surface Fitting
- Optimization
- Special Function Approximation
- Linear Algebra
- Correlation & Regression Analysis
- Multivariate Methods
- Analysis of Variance
- Random Number Generators
- Univariate Estimation
- Nonparametric Statistics
- Smoothing in Statistics
- Contingency Table Analysis
- Survival Analysis
- Time Series Analysis
- Operations Research

The NAG Numerical libraries

- NAG Fortran Library
- NAG C Library
- NAG Library for .NET
- NAG Library for SMP & multicore
    - for symmetric multi-processor machines (OpenMP)
- NAG Parallel Library
    - for distributed memory parallel machines (MPI)
- NAG Toolbox for MATLAB

Other NAG library interfaces

- C
- C++
- C# / .NET
- Fortran
- Java
- Python
- Visual Basic
- CUDA
- OpenCL
- F#
- …

- Excel
- MATLAB
  - Octave, SciLab, Freemat…
- Maple
- Mathematica
- SciLab
- PowerBuilder
- LabVIEW
- R and S-Plus
- SAS
- Simfit
- …

Overview

- An introduction to NAG
  - NAG numerical libraries
- NAG and LabVIEW
  - how to call NAG routines from LabVIEW
    - NAG Library for .NET
    - NAG Fortran Library, NAG C Library
    - using a wrapper library
- Conclusions
  - finding out more

LabVIEW

- Platform and development environment
- Uses G, a visual programming language
- Variety of uses
  - data acquisition, instrument control, industrial automation
- Application = *virtual instrument* (VI). Builders
  - add *controls* and *indicators* to *front panel*
  - add *nodes* and *connections* to *block diagram*
- VIs can be embedded as subroutines in other VIs
- Can be extended by interfacing to external libraries

A LabVIEW user writes...

"The NAG numerical and statistical libraries have a long established reputation in academia and industry throughout the world. Their extensive use in a wide range of disciplines where accuracy and robustness are essential is testimony to their reliability.

As a LabVIEW programmer, having direct access to the same routines as the theoretical physicists and statisticians, working in different programming environments, is extremely reassuring and can save much time in software validation. This, coupled with breadth of the libraries (over 1,700 routines) and the depth of the documentation makes NAG an excellent choice for handling complex numerical analysis in LabVIEW."

Conway Langham, NPL

Using NAG in LabVIEW

- Supplement LabVIEW functionality w/ NAG routines
- Procedure to call routine depends on the library used
  - NAG Library for .NET
    - .NET assembly
    - uses CLR and .NET framework to manage assembly functions
  - NAG C Library / NAG Fortran Library
    - dynamic link library (DLL)

Overview

- An introduction to NAG
  - NAG numerical libraries
- NAG and LabVIEW
  - how to call NAG routines from LabVIEW
    - NAG Library for .NET
    - NAG Fortran Library, NAG C Library
    - using a wrapper library
- Conclusions
  - finding out more

Using NAG Library for .NET (1)

- Library of classes and methods
- Bring up *Functions Palette* from block diagram
- Open *Connectivity* collection, then *.NET* item
  - functions for creating .NET objects, setting properties etc
- Select *Invoke Node* item and drag onto diagram
- Right-click on node: *Select Class >> .NET >> Browse...*
  - brings up *Select Object From Assembly* dialog
- Select *NagLibrary*
  - find the DLL using *Browse...* if it's not already listed

Using NAG Library for .NET (2)

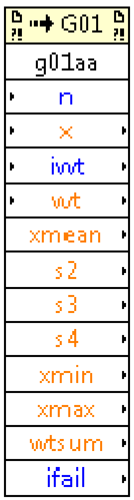
- Double-click on *NagLibrary* item
  - shows all objects in this assembly
  - select the desired class
    - click OK to load it into the Property node
- Click on *Method*
  - shows all methods in this class
    - function arguments are explicitly shown
  - select the desired method
    - function arguments are shown in block diagram
- Add LabVIEW controls (input) and indicators (output)
  - to front panel, then wire them up

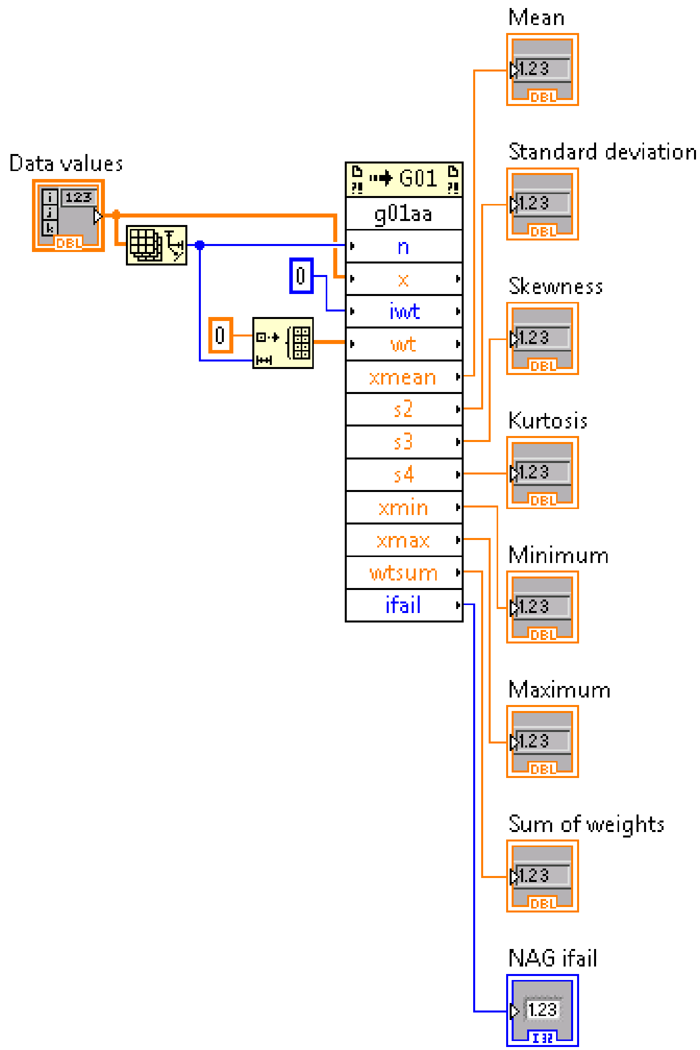
Example: g01aa method

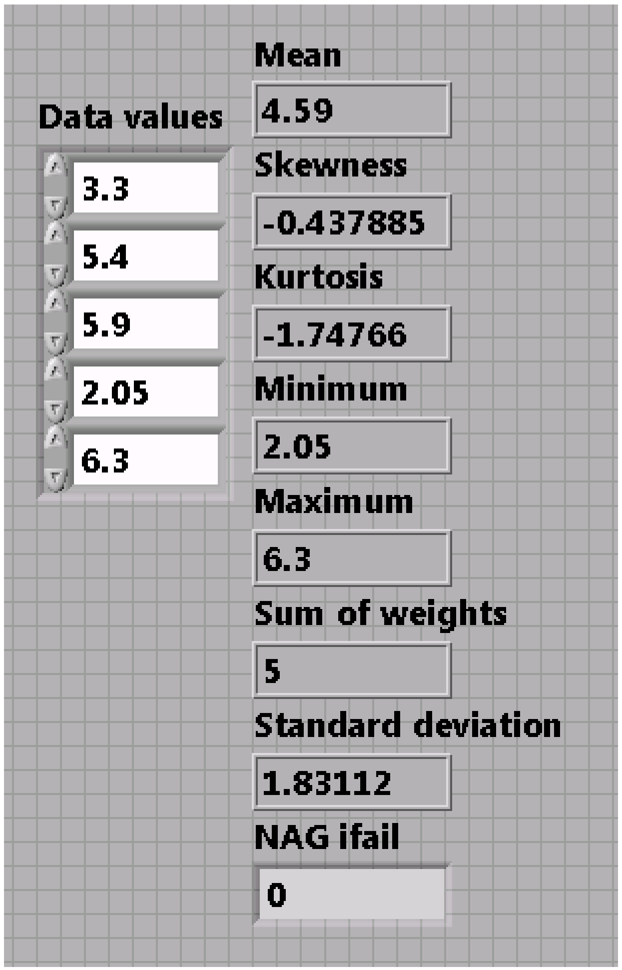
- Calculates simple statistics for a set of ungrouped data

```
//  g01aa Example Program Text
using System;
using NagLibrary;
using System.IO;
public class G01AAE
  {
   double s2, s3, s4, wtsum, xmax, xmean, xmin;
   int i, iwt, j, n, ifail;
...
   G01.g01aa(n, x, ref iwt, wt, out xmean, out s2, out s3, out s4,
                out xmin, out xmax, out wtsum, out ifail);
   if (ifail == 0) {
     Console.WriteLine(" {0}", "Data as input -");
     for (i = 1; i <= n; i++) {
       Console.Write(" {0, 12:f1}{1}", x[i - 1], i%5==0?"\n":"");
     }
     Console.WriteLine("  {0}{1,13:f1}", "Mean          ", xmean);
     Console.WriteLine("  {0}{1,13:f1}", "Std devn      ", s2);
...
```

Using g01aa in LabVIEW







Overview

- An introduction to NAG
  - NAG numerical libraries
- NAG and LabVIEW
  - how to call NAG routines from LabVIEW
    - NAG Library for .NET
    - NAG Fortran Library, NAG C Library
    - using a wrapper library
- Conclusions
  - finding out more

Using NAG Library for C or Fortran (1)

- Library of functions and subroutines
- Bring up *Functions Palette* from block diagram
- Open *Connectivity*, then *Libraries & Executables* item
  - functions for calling code from libraries
- Select *Call Library Function* item, drag onto diagram
- Right-click on node: *Configure…*
  - brings up *Call Library Function* dialog
- Enter the DLL name in *Library name or path*
  - find it using the *open* icon. Need to include the path.

Using NAG Library for C or Fortran (2)

- Look in the *Function name* menu
  - shows all functions in this library
  - select the desired function
    - and stdcall (WINAPI) as the calling convention
- Click on *Parameters*
  - enter function parameters and types
  - translate each into a LabVIEW type, then click *OK*
- Add LabVIEW controls (input) and indicators (output)
  - to front panel, then wire them up
  - turn on display of parameter names for readabiity
    - right-click on node, set *Name Format >> Names*

Example: g01aac routine

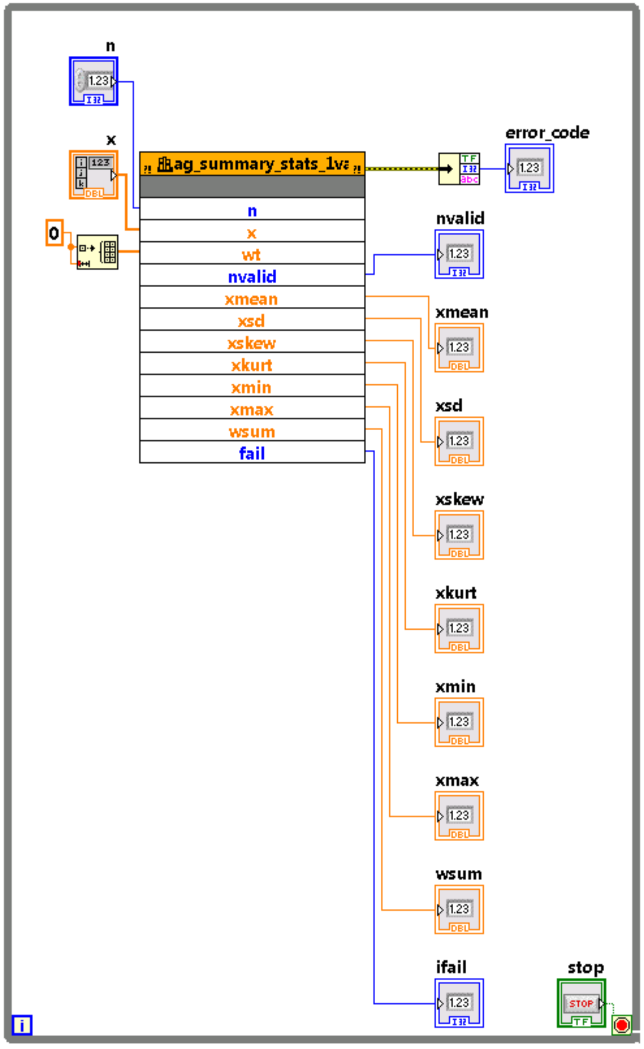
- Calculates simple statistics for a set of ungrouped data

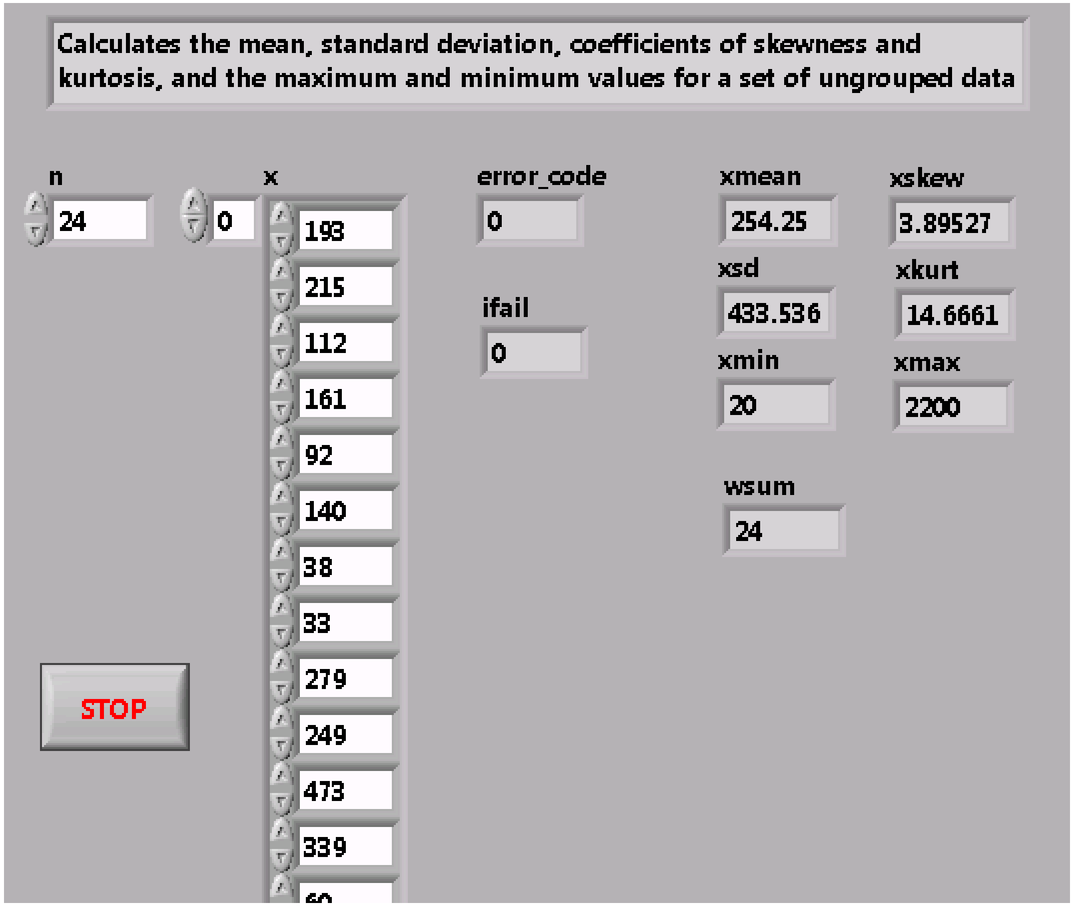
```
/* nag_summary_stats_1var (g01aac) Example Program */
#include <nag.h>
#include <nagx04.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagg01.h>
int main(int argc, char *argv[])
{
  Integer  n, nvalid;
  NagError fail;
  double wsum, *wt = 0, *x = 0, xkurt, xmax, xmean, xmin, xsd, xskew;
...
  nag_summary_stats_1var(n, x, (double *) 0, &nvalid, &xmean, &xsd,
                               &xskew, &xkurt, &xmin, &xmax, &wsum, &fail);
  if (fail.code == NE_NOERROR) {
     fprintf(fpout, "Successful call of nag_summary_stats_1var (g01aac)\n");
     fprintf(fpout, "Mean           %13.1f\n", xmean);
     fprintf(fpout, "Std devn       %13.1f\n", xsd);
...
```

Translating the data

C name	C type	LabVIEW type	LabVIEW const?	LabVIEW type	LabVIEW pass	LabVIEW array format	LabVIEW C type
`n`	`Integer`	Numeric	Y	Signed 32-bit Integer	Value	—	`const int32_t`
`x`	`const double array`	Array	Y	8-byte Double	—	Array Data Pointer	`const double*`
...	...	...	...	...	...	...	...
`xmean`	`double*`	Numeric	N	8-byte Double	Pointer to Value	—	`double*`
...	...	...	...	...	...	...	...
`fail`	`NagError*`	Numeric	N	Signed 32-bit Integer	Value	—	`int32_t`

Using g01aac in LabVIEW

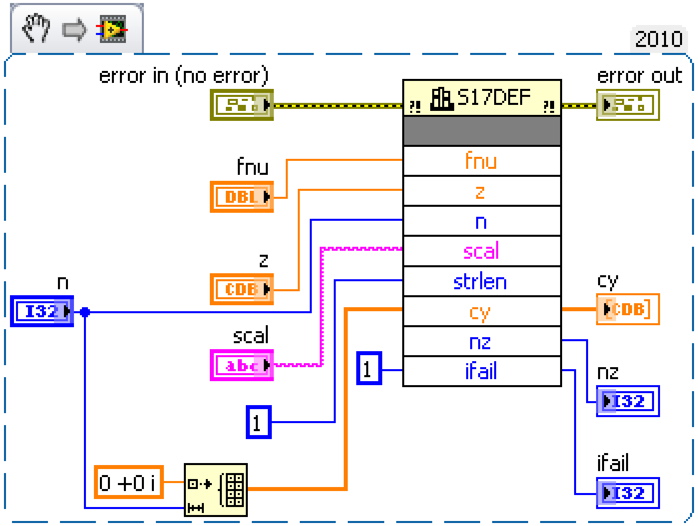




S17DEF example

- Complex Bessel function from NAG Fortran Library
  - supports complex arguments
    - cf built-in LabVIEW Bessel function
- Use *Call Library Function* to specify it as

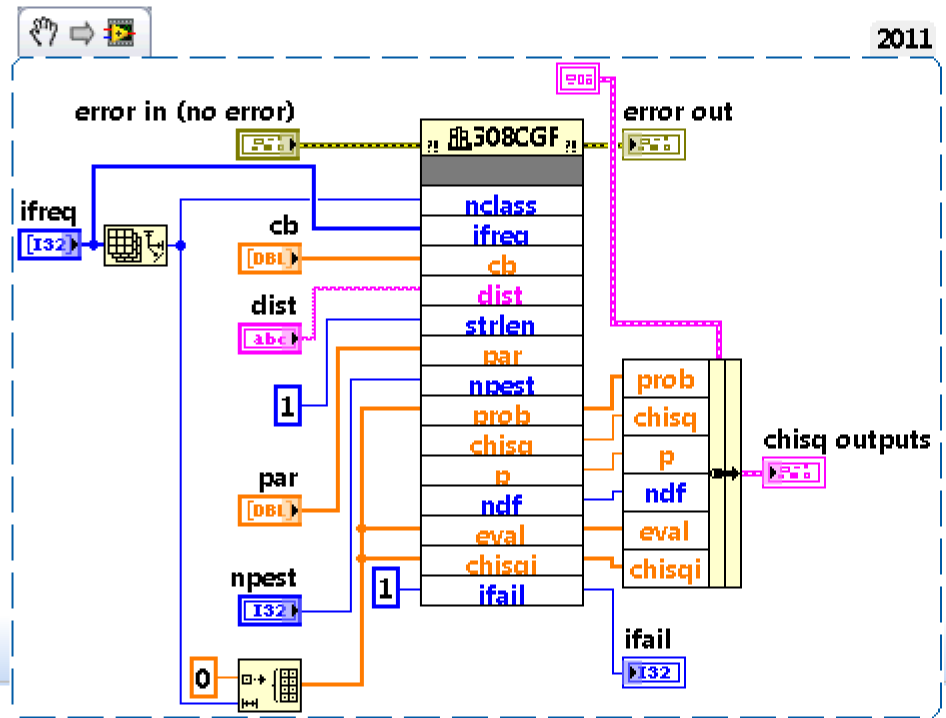
```
void S17DEF( double *fnu, void *z,
        int32_t *n, CStr scal,
        int32_t *strlen, void *cy,
        int32_t *nz, int32_t *ifail);
```



G08CGF example

- Chi-squared test from NAG Fortran Library
- Computes test statistic for χ^2 goodness-of-fit test
  - with chosen number of class intervals
  - does a random sample arise from a specified distribution?
- Use Call Library Function to specify it as

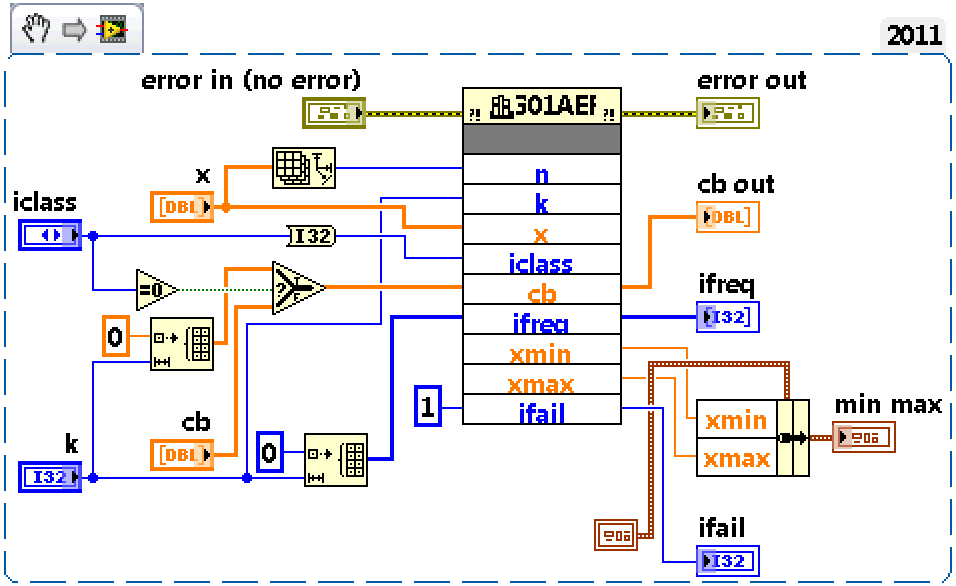
```
void G08CGF( int32_t *nclass,
int32_t *ifreq, double *cb,
CStr dist, int32_t strlen,
double *par, int32_t *npest,
double *prob, double *chisq,
double *p, int32_t *ndf,
double *eval, double *chisqi,
int32_t *ifail );
```



G01AEF example

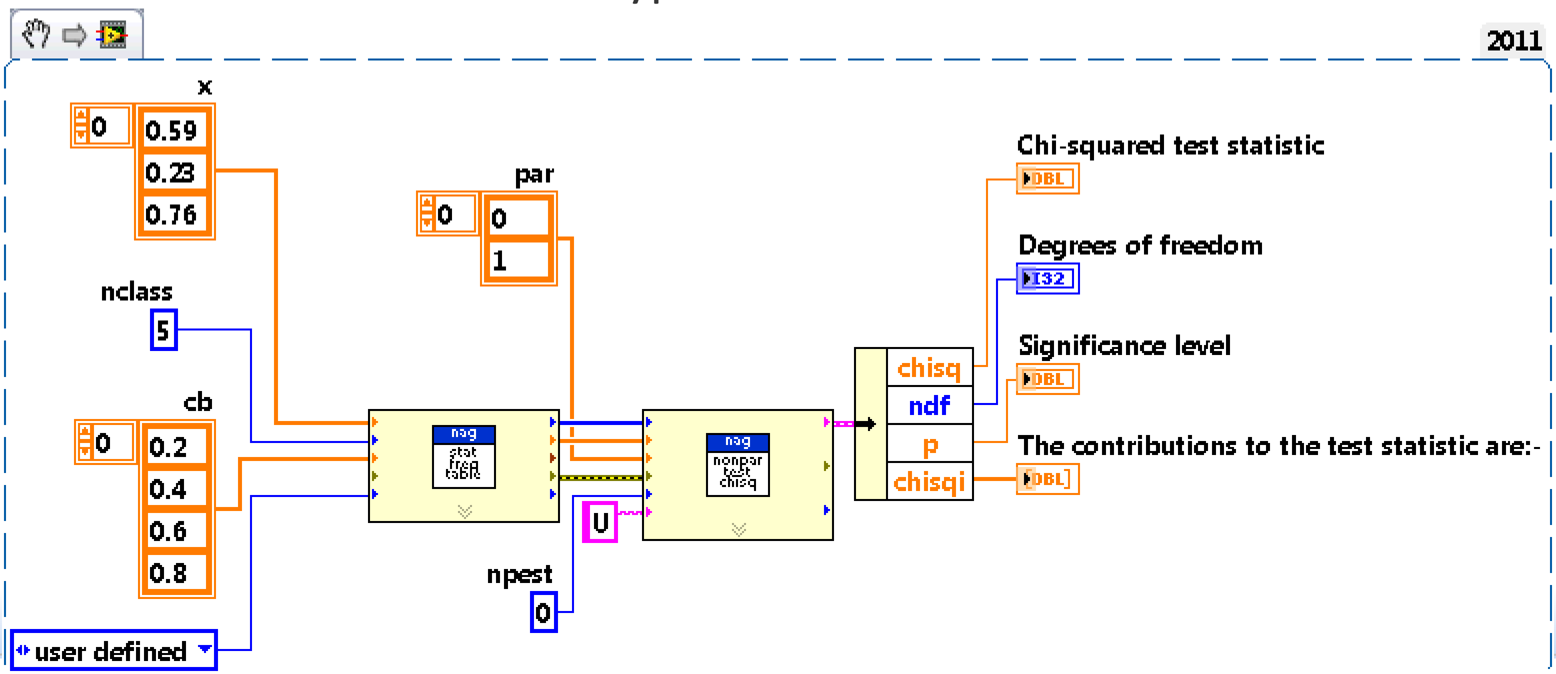
- Constructs a frequency distribution of a variable
  - according to supplied class-boundary values
    - can be user-supplied or internally-calculated
- Use Call Library Function to specify it as

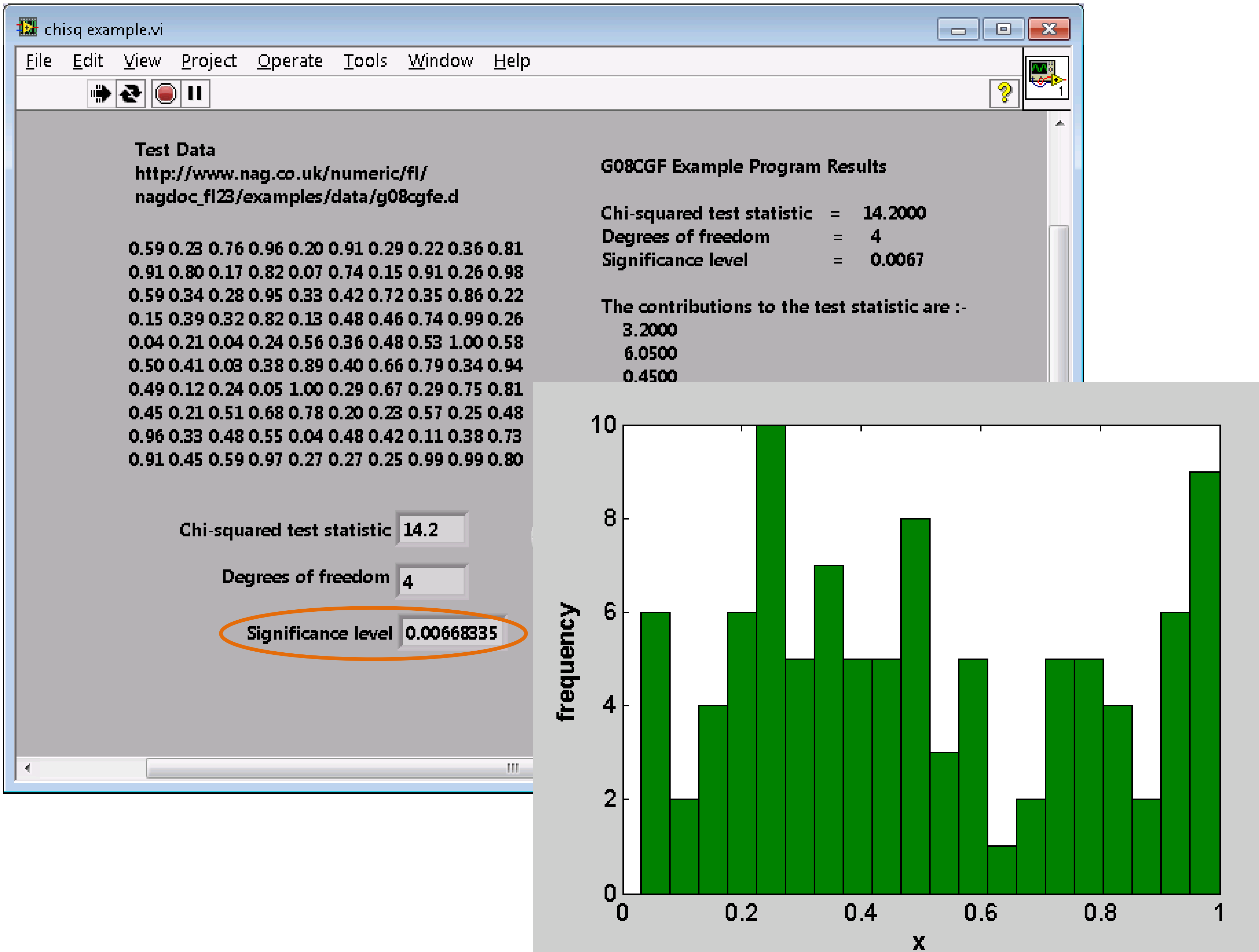
```
void  G01AEF( int32_t *n,
int32_t *k, double *x,
int32_t *iclass, double *cb,
int32_t *ifreq, double *xmin,
double *xmax,
int32_t *ifail );
```



Putting it together

- Are a sample of 100 numbers uniform?
  - is there evidence to suggest that a sample of 100 randomly generated values do not arise from a uniform distribution?
    - divide (0,1) into 5 equal classes, get frequencies from G01AEF
    - use G08CGF to test hypothesis




chisq example.vi
File Edit View Project Operate Tools Window Help
Test Data
http://www.nag.co.uk/numeric/fl/
nagdoc_fl23/examples/data/g08cgfe.d
0.59 0.23 0.76 0.96 0.20 0.91 0.29 0.22 0.36 0.81
0.91 0.80 0.17 0.82 0.07 0.74 0.15 0.91 0.26 0.98
0.59 0.34 0.28 0.95 0.33 0.42 0.72 0.35 0.86 0.22
0.15 0.39 0.32 0.82 0.13 0.48 0.46 0.74 0.99 0.26
0.04 0.21 0.04 0.24 0.56 0.36 0.48 0.53 1.00 0.58
0.50 0.41 0.03 0.38 0.89 0.40 0.66 0.79 0.34 0.94
0.49 0.12 0.24 0.05 1.00 0.29 0.67 0.29 0.75 0.81
0.45 0.21 0.51 0.68 0.78 0.20 0.23 0.57 0.25 0.48
0.96 0.33 0.48 0.55 0.04 0.48 0.42 0.11 0.38 0.73
0.91 0.45 0.59 0.97 0.27 0.27 0.25 0.99 0.99 0.80
Chi-squared test statistic 14.2
Degrees of freedom 4
Significance level 0.00668335
G08CGF Example Program Results
Chi-squared test statistic = 14.2000
Degrees of freedom = 4
Significance level = 0.0067
The contributions to the test statistic are :-
3.2000
6.0500
0.4500
10
8
6
4
2
0
frequency
0
0.2
0.4
0.6
0.8
1
x

Overview

- An introduction to NAG
  - NAG numerical libraries
- NAG and LabVIEW
  - how to call NAG routines from LabVIEW
    - NAG Library for .NET
    - NAG Fortran Library, NAG C Library
    - using a wrapper library
- Conclusions
  - finding out more

Building a wrapper library (1)

- Create a DLL which wraps selected NAG functions
  - possibly using a simplified interface
  - compiled from source using e.g. Visual Studio
- Import the DLL into LabVIEW
  - via *Tools >> Import >> Shared Library (.dll)...*
- LabVIEW will create a project library of wrapper VIs
  - one for each selected function in the DLL

Building a wrapper library (2)

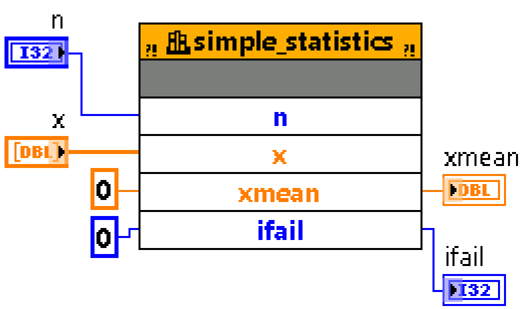
- Wizard allows creation or updating of VIs
- Specify name and location of DLL and header files
- Specify any include paths or preprocessor definitions
  - to be used when parsing the header file
- Select the functions from the DLL to be wrapped
- Specify the name and location of the project library
- Specify the type of error handling
- Configure the VIs and controls
  - specify parameter types, input/output etc

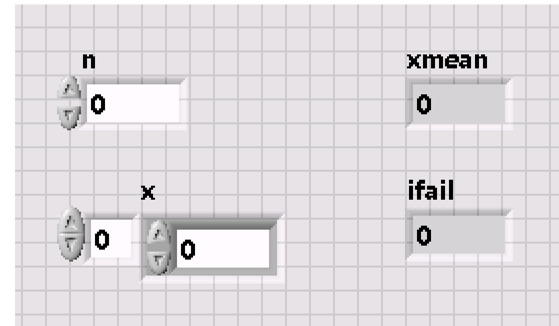
Example: wrapping g01aac

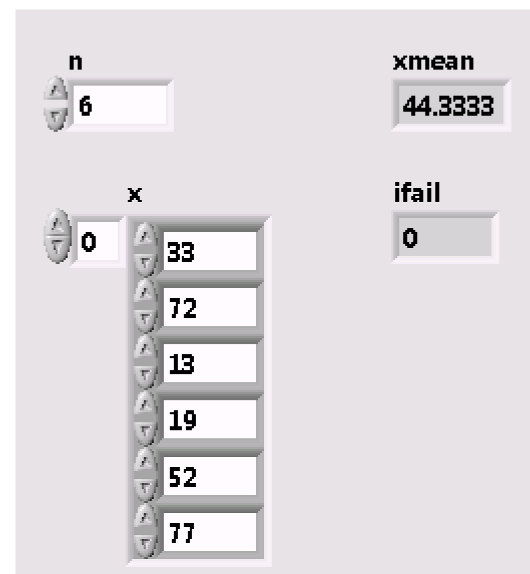
```
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg01.h>
#include "NAGWRAPPED.h"
...
_declspec (dllexport) void simple_statistics(int n, double *x, double *xmean, int *ifail)
{
    static NagError fail;
    int             nvalid;
    double          wsum, *wt = 0, xsd, xskew, xkurt, xmin, xmax;

    nag_summary_stats_1var(n, x, (double *)0, &nvalid, xmean, &xsd, &xskew,
                                 &xkurt, &xmin, &xmax, &wsum, &fail);
    if (fail.code == NE_NOERROR) {
        *ifail = fail.code;
    }
    else {
        *ifail = fail.code;
    }
}
```

simple_statistics.vi

```
_declspec (dllexport) void simple_statistics(int n, double *x, double *xmean, int *ifail);
```







Example routines (so far)

- Fortran
  - e04uff – minimization using SQP
  - f01blf – rank & pseudo inverse
  - f02bjf – evals for generalized eproblem
  - f03aaf – determinant for real matrix
  - f06eaf – scalar product of 2 real vectors
  - f06ejf – Euclidean norm
  - f06raf – 1-norm, Frobenius norm
  - f06yaf – matrix-matrix operations
  - f07fdf – Cholesky factorization of matrix
  - f08kbf – min norm soln to least-squares
  - f08naf – evals for real nonsym matrix
  - f08waf –evals for generalized eproblem
  - g01aaf – simple statistical calculations
- C
  - c06ekc – circular convolution of 2 vectors
  - c09cac – computes 1D wavelet transform
  - **e04nfc** – general QP problems
  - f01ecc – matrix exponential
  - f07aec – real system w/ many rhs
  - f07adc – LU factorization of real matrix
  - f07agc – condition number of matrix
  - f07ajc – inverse of matrix
  - f07fdc – Cholesky factorization of matrix
  - f07tec – real triangular system w/ many rhs
  - f08aec – QR factorization of real matrix
  - f08nec – reduces real matrix to Hessenberg
  - f08qhc – solves Sylvester matrix eqn

Example routines (so far)

- C (contd)
  - f08vac – computes GSVD of A and B
  - f16yac – multiplies real general matrices
  - g01aac – simple statistical calculations
  - s01bac – shifted-log function
- .NET
  - c09cc – 1D multi-level wavelet
  - **e01ba** – cubic spline interpolant
  - e04uf – minimisation using SQP

Wrapped routines (so far)

- f01ecc – matrix exponential
- f02bjc – evals and evectors of generalized eigenproblem
- f07adc – LU factorization of real matrix
- f07aec – real system w/ many rhs
- f07agc – condition number of matrix
- f07ajc – inverse of matrix
- f07fdc – Cholesky factorization of matrix
- f07tec – real triangular system w/ many rhs
- f08aec – QR factorization of real matrix
- f08nec – reduces real matrix to Hessenberg
- f08qhc – solves Sylvester matrix eqn
- f08vac – computes GSVD of A and B
- f16yac – multiplies real general matrices

Overview

- An introduction to NAG
  - NAG numerical libraries
- NAG and LabVIEW
  - how to call NAG routines from LabVIEW
    - NAG Library for .NET
    - NAG Fortran Library, NAG C Library
    - using a wrapper library
- Conclusions
  - finding out more

Conclusions

- NAG offers software components for developers
  - no wheel-reinvention, stone canoes, chocolate teapots
- Portable
  - constantly being implemented on new architectures
  - made accessible from different software environments
    - LabVIEW, Matlab, Excel, R,…
- Reliable
  - extensive experience at implementing numerical code

Finding out more

- More on NAG and LabVIEW:
  http://www.nag.co.uk/numeric/LabView.asp
- Technical support and help with NAG products:
  support@nag.co.uk
- Today's speaker:
  jeremy.walton@nag.co.uk