Using the NAG C Library in C/C++

Making it Easier to Write Financial Applications

Jacques du Toit, NAG
OVERVIEW OF NAG
Numerical Algorithms Group

- NAG provides mathematical and statistical algorithm libraries widely used in industry and academia
  - Established in 1970 with offices in Oxford, Manchester, Chicago, Taipei, Tokyo
  - Not-for-profit organisation committed to research & development
  - Library code written and contributed by some of the world’s most renowned mathematicians and computer scientists
  - NAG’s numerical code is embedded within many vendor libraries such as AMD and Intel
  - Many collaborative projects – e.g. CSE Support to the UK’s largest supercomputer, HECToR
Contents/Scope of the NAG Library

- Root Finding
- Summation of Series
- Quadrature
- Ordinary Differential Equations
- Partial Differential Equations
- Numerical Differentiation
- Integral Equations
- Mesh Generation
- Interpolation
- Curve and Surface Fitting
- Optimization
- Approximations of Special Functions

- Dense Linear Algebra
- Sparse 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
NAG Library and Performance

- NAG Library builds on top of “vendor libraries”
  - Low-level core maths libraries by hardware manufacturers such as AMD and Intel
  - Do very basic operations such as linear algebra, FFTs, etc
  - These are building blocks of many mathematical algorithms
  - Crucial these building blocks are as fast as possible
  - Vendors produce such libraries which exploit special features of their chips
  - NAG builds on top of these libraries to benefit from their speed
NAG Products

➤ “NAG Library” is available in many languages
  • Fortran, C/C++, Java, Matlab, Excel, .NET
  • Functionality is the same – based on same underlying code base

➤ Other products
  • NAG Fortran compiler
  • Numerical Routines for GPUs
  • ... and several others ...
NAG AT MANCHESTER
NAG at Manchester University

- Unlimited use for the Linux, Mac, Solaris and Windows
  - As long as for academic or research purposes
  - Installation may be on any university, staff or student machine

- Products that are available
  - All NAG Libraries: Fortran, C, SMP, Toolbox for MATLAB, Numerical Components for GPU & NAG Fortran Compiler

- How do you get access to the software?
  - Temporary Licence keys via support@nag.co.uk
  - Permanent Licence keys via applicationsupport-eps@manchester.ac.uk

- Anything extra / unusual
  - Michael.Croucher@manchester.ac.uk
  - Also author of [www.walkingrandomly.com](http://www.walkingrandomly.com) where NAG features – good and bad!
Support

▶ Full access to NAG Support support@nag.co.uk
  • Request support or temporary licence keys using your university e-mail address please xxxx@xxxx.ac.uk

▶ Our software:
  • Includes online documentation - also www.nag.co.uk
  • Supplied with extensive example programs

▶ UK Academic Account Manager:
  • louise.mitchell@nag.co.uk
NAG C LIBRARY
Read the User’s Note

- Says how to compile from command line
- Says how to compile from Visual Studio
- Says which additional vendor libraries you need to link to
- Explains some compiler flags
- Says where to find example programs and how to run them

Read the Essential Introduction

- Explains a lot of what we’ll talk about now, in more detail
- Error handling, row/column major order, data types, calling conventions ...
NAG C Library – Structure

Structure

- Library is broken into chapters, each with a chapter introduction (doc on web e.g. g05)
- Each chapter deals with particular class of mathematical problems

Each chapter contains various routines (functions)

- Functions have a short name (e.g. g05kfc) and a long name (nag_rand_init_repeatable)
- First 3 characters in short name identify Chapter, next 2 identify function, last character is always “c”
- Tutorial solutions (code) will use long names, tutorial question sheet will use short name (to save space)
NAG C Library – Error Handling

Error handling

- All functions have a *NagError* parameter as last argument

```c
NagError fail;
// SET_FAIL(fail);
INIT_FAIL(fail);
Integer seed=23, lseed=1, state[12], lstate=12;
g05kfc(Nag_BaseRNG,0,seed,lseed,state,&lstate,&fail);
```

- When error occurs, a NAG routine has four options:
  1. Set *fail.code* and return to user (don’t print to console)
  2. Set *fail.code*, print error message and return to user
  3. Set *fail.code*, print error message and stop program
  4. Set *fail.code*, maybe print message and call user-supplied error handling function
NAG C Library – Error Handling

- If use `INIT_FAIL(fail)`, routine only sets `fail.code` and returns

```c
NagError fail;
INIT_FAIL(fail);
Integer seed=23, lseed=1, state[12], lstate=12;
g05kfc(Nag_BaseRNG,0,seed,lseed,state,&lstate,&fail);
if(fail.code != NE_NOERROR) {
    ... // You MUST do your own error checking
}
```

- If use `SET_FAIL(fail)`, routine sets `fail.code`, prints an error message to the console and returns to the user

```c
NagError fail;
SET_FAIL(fail);
Integer seed=23, lseed=1, state[12], lstate=12;
g05kfc(Nag_BaseRNG,0,seed,lseed,state,&lstate,&fail);
if(fail.code != NE_NOERROR) {
    ... // You should probably still check if an error occurred ...
}
```
NAG C Library – Error Handling

- If use *NULL*, routine prints an error message to console and stops your program

```c
Integer seed=23, lseed=1, state[12], lstate=12;
g05kfc(Nag_BaseRNG,0,seed,lseed,state,&lstate,NULL);
// No need for NagError structure or to do any error checking!
```

- If want to use your own error handler (a function which NAG Library will call when an error occurs), then read Section 3.6.3 in the Essential Introduction

- For the Tutorials, we will simply use *NULL*

  - Keeps everything nice and simple
NAG C Library – Integers

- Integer data types
  - The NAG C Library uses a custom data type `Integer` for ints
  - This will be a signed 32bit or 64bit int, depending on your implementation
    - E.g. when processing huge arrays (>4GB) you need 64bit indexes
  - When a routine asks for an `Integer` (in the routine doc), please give it an `Integer` and not an `int`. I.e. declare your integer variables as type `Integer`.
  - Note: `Integer` is always signed! So you should not do this:

```c
Integer n;
unsigned int u;
nag_some_routine_to_compute_n(&n,NULL);
u = n; // Your compiler should complain at you here!
```
NAG C Library – Row and Column Major

- Row major and Column major storage
  - Some routines (e.g. f16pac) have a \texttt{Nag\_OrderType} argument
  - C doesn’t have concept of 2D or 3D arrays when you allocate memory on heap (using \texttt{new} or \texttt{malloc})
  - So if have matrix, you must tell NAG Library how that matrix is stored
Example: suppose have the following matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}$$

We could code this up as

```c
double A[3*4] = {a11,a12,a13,a14,  a21,a22,a23,a24,   a31,a32,a33,a34};
```

- Store a *row* at a time: corresponds to *Nag_RowMajor*

Or we could code this up as

```c
double A[4*3] = {a11,a21,a31,  a12,a22,a32,  a13,a23,a33,  a14,a24,a34};
```

- Store a *column* at a time: corresponds to *Nag_ColMajor*
The one is simply the transpose of the other

So which is better?

- In general, both are equally good
- Most LAPACK and BLAS routines expect `Nag_ColMajor`
- If call these with `Nag_RowMajor`, routine may need to transpose your matrix before passing it to underlying algorithm
- This could impact performance, if matrices are large

If calling LAPACK or BLAS, try to store matrices as `Nag_ColMajor`
NAG C Library – Callbacks

C/C++ has at least 3 different calling conventions

<table>
<thead>
<tr>
<th>Function</th>
<th>Calling Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>func1</td>
<td>__cdecl</td>
</tr>
<tr>
<td>func2</td>
<td>__stdcall</td>
</tr>
<tr>
<td>func3</td>
<td>__fastcall</td>
</tr>
</tbody>
</table>

- Pretty low level – has to do with who cleans up the stack
- Only an issue when you pass a function pointer (callback) to a NAG routine (e.g. c05awc)

Your callback has to have same calling style that NAG library expects

- If not, should get a compiler error. Otherwise will see very weird behaviour: stack corruption, segfaults, or (worst of all!) no errors but wrong answers!
NAG C Library – Callbacks

There’s a very easy way to get it right!

- NAG Library headers define a macro `NAG_CALL` which has the correct calling style you should use
- Use `NAG_CALL` whenever you write a function you’ll pass to the NAG Library (a callback)

```c
double NAG_CALL myFunc(double x, NagComm * comm)
{
    ... // Function body
}

int main()
{
    ...
    c05awc(&x,eps,eta,myFunc,nfmax,&comm,NULL);
    ...
}
```
Can I pass a class member function to a NAG Library routine?

class MyClass {
    double * myData;
    public:
    // Constructor
    MyClass();
    // Public member function
    double NAG_CALL myFunc(double x, NagComm * comm);
};

int main()
{
    ...
    MyClass Z;
    c05awc(&x,eps,eta,Z.myFunc,nfmax,&comm,NULL);  // Won't compile!
    ...
}
Class member functions are very different from ordinary functions!

• You cannot pass them to NAG Library routines
• There is a simple workaround:
  ▪ Write a normal function (say, foo) with the correct prototype
  ▪ Put a pointer to your class in a NagComm structure (say, comm)
  ▪ Call the NAG routine, passing in foo and comm
  ▪ The NAG routine will call foo and give it comm
  ▪ Inside foo, take the pointer to your class from comm and call your member function
SOME COMMON FINANCIAL PROBLEMS
Implied Volatility

- What is implied volatility?
- Where is it used?
- How is it computed?
Implied Volatility

➤ What is implied volatility?

• Volatility I must put into Black-Scholes equation to reproduce a given price, i.e.

\[ \text{Find } \theta_\ast \text{ such that } BS_{\text{Call}} (S, K, T, r, q, \theta_\ast) = C_\ast \]

➤ Where is it used?

• Everywhere!! Pricing (quotes), calibration, modelling, ...

➤ How is it computed?

• Have to invert Black-Scholes formula \( BS_{\text{Call}} \)
• Impossible to do analytically, so have to do it numerically
• Use \textit{root finders} (chapter C05) together with the Black-Scholes formula (chapter S30)
Simulation

- What is simulation?
- Where is it used?
- How is it done?
Simulation

What is simulation?
• Method of generating possible future scenarios in order to study what might happen

Where is it used?
• Pricing, risk modelling, hedging, immunising, ...

How is it done?
• You need a source of randomness (what is randomness?)
  ▪ RNGs (Chapter G05) with different distributions
• You then need a way of turning that randomness into a future scenario
  ▪ Solving SDEs, time series forecasting, econometric models, ...
Calibration

- What is calibration?
- Why do we calibrate?
- Where is it used?
- How is it done?
Calibration

What is calibration?

- Process of choosing model parameters to match prices I observe in the market today

Why do we calibrate?

- Price of a vanilla (call/put/barrier) is the price given on your screen! Don’t calibrate to price vanillas
- Calibrate to price exotics and to hedge

Where is it used?

- Any model with free parameters (all models) will need to be calibrated
- Calibration is the process of choosing values for the free parameters
Calibration

Let’s formalise this a bit

• Suppose we have a model with parameters $\alpha_1, \ldots, \alpha_p$ (e.g. Black-Scholes, Heston, Hull-White, SABR ...)

• Model gives formula $F_{Call}(S, T, K, r, \alpha_1, \ldots, \alpha_p)$ to price call option with maturity $T$, strike $K$, current stock price $S$ and risk free interest rate $r$

• In the market, we observe call prices $(C_1, K_1, T_1), (C_2, K_2, T_2), \ldots, (C_n, K_n, T_n)$

• According to our model (if it’s correct), we should have $C_i = F_{Call}(S, T_i, K_i, r, \alpha_1, \ldots, \alpha_p)$ for all $i = 1, \ldots, n$

• So choose $\alpha_1, \ldots, \alpha_p$ so that $C_i = F_{Call}(S, T_i, K_i, r, \alpha_1, \ldots, \alpha_p)$ for all $i = 1, \ldots, n$
Calibration

Simples!!

- Unfortunately not ...
- Typically \( p \approx 6 \) while \( n \geq 20 \). Impossible to get equality
  - Some models have \( p \geq n \) so that equality is theoretically possible
- So we want a **best fit**. Common approach

\[
\min_{\alpha_1,\ldots,\alpha_p} \sum_{i=1}^{n} w_i \left( C_i - F_{Call}(S, K_i, T_i, r, \alpha_1, \ldots, \alpha_p) \right)^2
\]

- Called **non-linear least squares optimisation**
- Difficult problem to solve in general (constraints!)
- Routines in Chapter E04 to do this (see Decision Trees in the E04 Chapter Introduction)
Nearest Correlation Matrix

- Models of more than one asset all have *correlation*
  - What is correlation?
  - Essential for basket options, interest rate models, multi-factor models (e.g. SLV), ...

- Mathematically, a correlation matrix $C \in \mathbb{R}^{n \times n}$ is
  - Square
  - Symmetric with ones on diagonal
  - Positive semi-definite: $x^T C x \geq 0$ for all $x \in \mathbb{R}^n$

- Estimating correlations is difficult!
  - Historical data is typically dirty, has missing values, contains arbitrages, ...
Nearest Correlation Matrix

- Most estimation techniques will give a symmetric, square matrix with ones on the diagonal
  - They WON’T give a positive semi-definite matrix!
  - If you use these estimates, in certain market conditions you will get negative variances

- NAG Library can find the “nearest” correlation matrix to a given square matrix $A$
  - g02aac solves problem $\min_C \|A - C\|_F^2$ in Frobenius norm
  - g02abc incorporates weights $\min_C \|W^{1/2}(A - C)W^{1/2}\|_F^2$
  - Weights useful when have more confidence in accuracy of observations for certain variables than for others
And Others ...

- Interpolation/surface fitting (e01/e02)
  - Used in local volatility modelling, rate modelling, ..

- Partial Differential Equation solvers (d03)
  - Used in pricing and calibrating models

- Quadrature (numerical integration) (d02)
  - Get option prices from some semi-analytic formulae

- FFT routines (c06)
  - Used in some pricing formulae

- Time series analysis and estimation (g13/g07)
  - Building forecasting models, e.g. in econometrics

- Etc, etc ....
Tutorial

► Time to get your hands dirty!