Using the Time Series Database kdb+ with q and the NAG Library

Kx Summer Kick Off - #analyticsatspeed

30th May 2018

John Holden
Vice President Global Markets
john.holden@nag.co.uk
Introduction
- I promise not to be long and arduous
- ... hold on tight for a whirl wind intro

How to Use the NAG Library with kdb+ and q
- A long and arduous process?

New Foreign Function Interface (FFI) Extension to q
- To the rescue....

How to Use the NAG Library with the FFI Extension to q
- A most pleasurable experience

Future Work
The Numerical Algorithms Group

- Experts in Numerical Computation and High Performance Computing
- Founded in 1970 as a co-operative project out of academia in UK
- Operates as a commercial, not-for-profit organization
  - Funded entirely by customer income
- Worldwide operations
  - Oxford & Manchester, UK
  - Chicago, US
  - Tokyo, Japan
- Over 3,000 customer sites worldwide
- NAG’s code is embedded in many vendor libraries (AMD, ARM, Intel)
NAG Products & Services

AD and adjoints

Numerical Library

Numerical Services

HPC Services

Hundreds of user-callable components
Lots of collaborations worldwide
Many-core R&D (e.g. Xeon Phi, GPU, ...)

Code and algorithm development, tuning, tailoring

Software/application innovation
Technology evaluation & migration
Procurement & commercial advice
Skills & training

Algorithmic Differentiation solutions, including dco/c++, dco/fortran and dco/map
The NAG Library

- Hundreds of routines devoted to numerical analysis and statistics, the NAG Library helps users build applications for many different industries and fields.

- For your current and future programming environments
  - NAG Library routines are available for C, C++, .NET, Python, MATLAB and Java others
  - NAG Library routines can be called many computer languages/environments such as Q, Hadoop and Apache SPARK, Visual Basic, Octave, Scilab, R, Haskell, Julia etc.
  - NAG is Cloudera certified
  - Assists migration of applications to different environments
Not just for Engineers, Quants - Data Scientists too....

- Machine Learning including
  - Linear regression (with constraints)
  - Logistic regression (with constraints)
  - Principal Component Analysis
  - Hierarchical cluster analysis
  - K-means

- Statistics including
  - Summary information (mean, variance, etc)
  - Correlation
  - Random number generation
  - Time Series

- Optimization including
  - Linear, nonlinear, quadratic, and sum of squares for the objective function
  - Constraints can be simple bounds, linear, or even nonlinear

- Matrix functions
  - Inversion
  - Nearest correlation
  - Eigenvalues + eigenvectors
Many clients - most Tier 1 Banks have licences

> 70% have global licences

Typically the NAG Library and/or our AD tools are embedded in the banks own “quant” libraries (C++, .NET, Java, Python,...)

- Also used by insurance companies, hedge funds, investment managers, traders, FSI ISVs, ...
Section One

How to Use the NAG Library with kdb+ and q
How to Use the NAG Library with kdb+ and q

Lengthy process for users
- Write C/C++ source code to wrap NAG C Library routine
- Include k.h header file from Kx Systems with C/C++ source code
- Must translate K types to C types, invoke NAG C Library routine, then translate C types back to K types
- Compile C/C++ source code into shared object
- Include path to shared object
- Include path to NAG C Library
- Use q to call wrapped NAG C Library routine from shared object

Explained in prior work by Brian Spector
- “Calling the NAG C Library from kdb+” (15 May 2013)
- http://blog.nag.com/2013/05/calling-nag-c-library-from-kdb.html
// Nearest Correlation Matrix (g02aaq.c)

#include <stdlib.h>
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>

#define KxVER 3  // kdb+ version 3
#include "k.h"  // Must follow kdb+ version

K g02aaq(K kg)
{
    // Verify that the function input is a K mixed list
    // NOTE: the complete type is a K list of K lists of doubles
    if (kg->t != 0)
        return krr("The input must be a list");

    // Determine the dimension of the matrix
    // NOTE: we assume that the matrix is square
    Integer n = kg->n;
    Integer pdg = n;

    // Declare a C array of size n*pdg to store values from the
    // mixed list kg, which we will then pass to the NAG routine
    double *g = 0;
    g = NAG_ALLOC(n*pdg, double);

    // Define two pointers to access the elements of the
    // mixed list kg
    K* kg_out; // outer list
    F* kg_inn; // inner list (64 bit double)

    // Assign the K pointer to the input matrix kg
    // NOTE: since kg is a mixed list of K types, we use kX()
    // as the accessor function
    kg_out = kK(kg);

    // Loop over the input mixed list kg
    for (int i = 0; i < n; i++) {
        // NOTE: we assume the mixed list kg contains only
        // doubles, and thus use kF() as the accessor function
        kg_inn = kF(kg_out[i]);

        // Confirm that the inner list is the same length as
        // the outer list
        if (kg_out[i]->n != pdg)
            return krr("The matrix should be square");

        // Define the C array g by looping over the elements in
        // the inner list
        for (int j = 0; j < pdg; j++) {
            g[i*n + j] = kg_inn[j];
        }
    }

    // continued on next slide
// continued from previous slide

// Define additional input parameters for NAG NCM routine
Nag_OrderType order = Nag_RowMajor;
double errtol = 1.00e-7;
Integer maxits = 200;
Integer maxit = 10;
Integer pdx = n;

// Declare return values for NAG NCM routine
Integer iter;
Integer feval;
double nrmgrd;

// Define NAG error argument
NagError fail;
INIT_FAIL(fail);

// Declare a K simple list to store the solution to
// the NAG routine
K kx;
kx = ktn(KF, n*pdx);

// Define a pointer to the solution matrix, which will
// be passed to the NAG routine
F* x = kF(kx); // 64 bit double

// Invoke NAG NCM routine
g02aac(Nag_RowMajor, g, pdg, n, errtol, maxits, maxit, x, pdx,
   &iter, &feval, &nrmgrd, &fail);

// Define K scalar values using the return values from the
// NAG routine. These values will be stored in a K mixed
// list and returned by this function
K kiter = ki(iter); // Number of Newton steps
K kfeval = ki(feval); // Number of function evaluations
K knrmgrd = kf(nrmgrd); // Norm of the gradient at the
// final Newton step

// Define a K mixed list to return from this function
K kout = knk(4, kx, kiter, kfeval, knrmgrd);

// Free the array g
NAG_FREE(g);

if (fail.code != NE_NOERROR)
   return r0(kout), krr(ss(fail.message));

return kout;

} // end of g02aaq()
/ main.q

/ Define input matrix (e.g. mixed list)
g:( 2.0 -1.0 0.0 0.0f;
   -1.0 2.0 -1.0 0.0f;
   0.0 -1.0 2.0 -1.0f;
   0.0 0.0 -1.0 2.0f)

/ Make the function g02aaq, defined in the shared object libg02aaq.so,
/ available in a q session, where:
/ (a) 2 loads the function dynamically
/ (b) 1 is the number of arguments the g02aaq accepts

```
g02aaq:(`libg02aaq 2:`g02aaq;1))
```

/ Invoke the function and store the results in the mixed list s

```
x:g02aaq[g]
```

/ Display solution (NCM is first element of the mixed list s)

```
show x(0)
```
Section Two

New Foreign Function Interface (FFI) Extension to q
Overview of Foreign Function Interface (FFI)

- Extension to kdb+ for loading and calling dynamic libraries using pure q
- Main purpose is to provide a stable interfaces for external libraries
- Officially released on 22 November 2017
- “No support is offered for crashes caused by use of this library”
- Currently in discussions with FFI developers to extend functionality (much more on this later)
How FFI Works

- Exposes new functions within FFI namespace
  - `cf` (call function)
    - Syntax: `cf[ret_type;fn]`
      - `ret_type` is the return type (char atom)
      - `fn` is the symbol name of a function
    - The types of arguments passed to the function are inferred from the `q` types and should match the width of the arguments expected by the C function
  - `bind` (create projection)
    - Syntax: `bind[fn;arg_types; rtn_type]`
      - `fn` is the symbol name of a function
      - `arg_types` are the argument types (char vector)
      - `ret_type` is the return type (char atom)
    - Returns a `q` function, bound to the specified C function for future calls

- Additional routines currently being explored
FFI Example Using Pseudocode

\l ffi.q

/ Define parameters to pass to function within shared object
param_1:value_1
param_2:value_2
param_N:value_N

/ Invoke function within shared object
.ffi.cf[("func_name_return_type";`shared_object.so`func_name)]
  (param_1;param_2;param_N)
Drastically Simplifies Development Process

Old method (e.g. before FFI)
- Write C/C++ source code to wrap NAG C Library routine
- Include k.h header file from Kx Systems with C/C++ source code
- Must translate K types to C types, invoke NAG C Library routine, then translate C types back to K types
- Compile C/C++ source code into shared object
- Include path to shared object
- Include path to NAG C Library
- Use q to call wrapped NAG C Library routine from shared object

Using FFI
- Include path the NAG C Library
- Use q to call alternative C interface for NAG C Library routine
Drastically Simplifies Development Process

- Eliminate majority of required steps from the development process
- Reduce number of development tools required
- Shorten overall development time
- Easier to debug
- Easier to maintain...
Section Three

How to Use the NAG Library with the FFI Extension to q
Compatibility Between NAG Library and q

- NAG C Library employs numerous Structs and typedefs (e.g. non-intrinsic C types) as arguments for routines
- q lacks the framework to create user-defined types
- q intrinsic types **will not** satisfy type requirements for NAG C Library
Solution: Use NAG Alternative C Interfaces with q

- Alternative C interfaces use only C intrinsic types (e.g. int, double, char, arrays, etc.)
- These C intrinsic types map directly to Fortran intrinsic types
- q intrinsic types will satisfy type requirements for NAG Alternative C Interface
Example: Nearest Correlation Matrix solver (g02aac)

/ g02aace.q
\l ffi.q

/ Define input parameters to NAG routine
g:(2.0 -1.0 0.0 0.0f; -1.0 2.0 -1.0 0.0f; 0.0 -1.0 2.0 -1.0f; 0.0 0.0 -1.0 2.0f)
ldg:1#4j
n:1#4j
errtol:1#0.0f
maxits:1#0j
maxit:1#0j
x:(0.0 0.0 0.0 0.0f; 0.0 0.0 0.0 0.0f; 0.0 0.0 0.0 0.0f; 0.0 0.0 0.0 0.0f)
ldx:1#4j
iter:1#0j
feval:1#0j
nrmgrd:1#0f
ifail:1#0j

/ Invoke NAG routine
.ffi.cf[" ";`libnagc_nag.so`g02aaf_]
   (g;ldg;n;errtol;maxits;maxit;x;ldx;iter;feval;nrmgrd;ifail)

/ Display solution
show x
Example: NAG Quadrature routine (d01bd)

/ This quadrature example integrates x^2 between 0 and 2
/ d01bdce.q

\1 ffi.q

/ Definition of q callback routine
/  f(x) = x^2
func:{[x] x*x}

/ Define input parameters
a:1#0f
b:1#2f
epsabs:1#0f
epsrel:1#0.0001f
result:1#0f
abserr:1#0f

/ Invoke NAG routine
.ffi.cf[("","libnagc_nag.so\`d01bdf\_")]
  ((func;1#"f";"f");a;b;epsabs;epsrel;result;abserr)

/ Display solution
show result
Call-back routines currently have a limit of 8 arguments

Unable to use routines that require a Fortran Subroutine callback (e.g. void C function)
- Able to update the value of a parameter passed to the callback routine used within a C function
- The C function is unable to recognize this updated value

Thanks to Fx engineers we now have fixes for both these issues, which need to be checked and tested

Even without the fixes we can address ~ 85% of the algos in the NAG Library
Section Four

Future Work
Future Work

- Continue to collaborate with Kx systems engineers
- Complete testing
  - of latest fixes / solutions supplied by Fx kdb+ team
- Produce examples for additional routines
  - suggestions and feedback from clients and other end users most welcome
- Publish blog / technical paper
Acknowledgements

Several large NAG and Kx clients for motivating this work

@NAG
- Chris Brandt
- Mick Pont
- Andrew Lawson

@Kx
- Fintan Quill
- Glenn Wright
- Mark Sykes
- James Corcoran
- Simon Garland
- Oleg Finkelshtein
- Sergey Vidyuk**

+ Brian Spector* for his 2013 paper
Questions?
Experts in High Performance Computing, Algorithms and Numerical Software Engineering

www.nag.com | blog.nag.com | @NAGtalk