Using the NAG Library for Python with Kdb+ and PyQ

Christopher Brandt, Numerical Algorithms Group Inc.

1 Background

This paper provides detailed instructions on how to use the NAG Library for *Python* with kdb+ and PyQ. The NAG Library contains more than 1,800 mathematical and statistical routines, and is accessible by numerous programming languages (including Python, C++, Java, Fortran, etc.). PyQ is an extension to kdb+ featuring zero-copy sharing of data between Python and the q programming language. The enclosed examples will illustrate how to access routines within the NAG Library for *Python* using data stored in kdb+.

2 Setting Up the Workspace

Installation for both the NAG Library for *Python* and the PyQ extension to kdb+ may be performed using pip.

To install the NAG Library for *Python*:

```
$ python -m pip install --extra-index-url
    https:nag/com/downloads/py/naginterfaces_nag naginterfaces
```

The PyQ package is developed by Kx Systems, the developers of kdb+. To install it:

```
$ python -m pip install pyq
```

Both the NAG Library for *Python* and kdb+ are commercial software packages that require active licenses for their respective usage. To obtain a temporary license for the NAG Library for *Python*, please contact NAG support at support@nag.com.

3 Examples

The following three examples demonstrate how to call the NAG Library for *Python* routines using kdb+ and PyQ. These examples were carefully selected, as they cover techniques found in the majority of usage cases a customer will encounter across all 1,800+ routines within the Library. If your usage case falls outside of these three examples, please contact NAG support for assistance.

3.1 Example One: BLAS Routine DAXPY

Our first example demonstrates how to perform the linear algebra operation

$$y \coloneqq \alpha x + b.$$

Below is the NAG Library for *Python* signature for this routine.

```
naginterfaces.library.blas.daxpy(alpha,x,y)

Parameters: alpha: float
            x: float, array-like, shape(n)
            y: float, array-like, shape(n)

Returns:    y: float, ndarray, shape(n)
```

Within our terminal, we begin by initiating a PyQ interactive session.

```
$ pyq
```

Next, we import PyQ and the BLAS module of the NAG Library for *Python*.

```
>>> from pyq import q
>>> from naginterfaces.library import blas
```

We then enter a q environment and define our parameters as q objects.

```
>>> q()
q) alpha:0.5f
q) x:4#2 2 2 2f
q) y:4#4 4 4 4f
```

Finally, we exit the q environment and invoke the NAG routine.

```
q) \
>>> z = blas.daxpy(float(q.alpha), q.x, q.y)
>>> z  # display solution: array([4., 4., 4., 4.])
```

3.2 Example Two: Nearest Correlation Matrix

Our second example employs a nearest correlation matrix routine which, for a given approximate correlation matrix G, computes the nearest correlation matrix X by minimizing the weighted Frobenius norm

$$\left\| W^{1/2}(G - X)W^{1/2} \right\|_F^2$$

where W is a diagonal matrix of weights.

The NAG Library for *Python* signature for this routine is below.

```
naginterfaces.library.correg.corrmat_nearest_bounded(
  g,opt,alpha=None,w=None,errtol=0.0,maxits=0,maxit=200)

Parameters: g: float, array-like, shape(n,n)
            opt: str, length 1
            alpha: None or float, optional
            w: None or float, array-like, shape(n), optional
            errtol: float, optional
            maxits: int, optional
            maxit: int, optional

Returns:    x: float, ndarray, shape(n,n)
            itera: int
            feval: int
            nrmgrd: float
```

Within our interactive PyQ session, we begin by importing the Correlation and Regression Analysis module of the NAG Library for *Python*.

```
>>> from naginterfaces.library import correg
```

Next, we enter a q environment and define our parameters as q objects.

```
>>> q()
q) alpha:0.5f
q) x:4#2 2 2 2f
q) g:4 4#2 -1 0 0 -1 2 -1 0 0 -1 2 -1 0 0 -1 2f
q) opt:"B"
q) alpha:0.02f
q) w:4#100 20 20 20f
```

We then exit the q environment and invoke the NAG routine.

```
q) \
>>> x, feval, itera, nrmgrd = correg.corrmat_nearest_bounded(
          q.g, str(q.opt), float(q.alpha), q.w)
>>> x  # display solution
```

3.3 Example Three: Numerical Integration

With our final example, we demonstrate how to incorporate a user-defined callback function with a NAG Library for *Python* routine. This example approximates the definite integral

$$\int_a^b f(x)dx.$$

The NAG Library for *Python* signature for this routine is below.

```
naginterfaces.library.quad.dim1_fin_smooth(f,a,b,epsabs,epsrel,data=None)

Parameters: f: callable, result = f(x,data=None)
               Parameters:
                   x: float
                   data: arbitrary, optional, modifiable in place
            a: float
            b: float
            epsabs: float
            epsrel: float
            data: arbitrary, optional

Returns:    result: float
            abserr: float
```

We start by importing the Quadrature module of the NAG Library for *Python*.

```
>>> from naginterfaces.library import quad
```

Next, we enter a q environment and define our parameters as q objects.

```
>>> q()
q) a:0f
q) b:2f
q) epsabs:0f
q) epsrel:0.0001f
```

We then exit the q environment and define an integrable Python function. To satisfy this parameter we may use either a Python function or a lambda expression.

```
q) \
>>> def f(x):
      return x*x
...
```

With our problem now fully defined, we invoke the NAG routine to compute our solution.

```
>>> result, error = quad.dim1_fin_smooth(
      f, float(q.a), float(q.b), float(q.epsabs), float(q.epsrel))
>>> result  # 2.6666666666666667
>>> error   # 1.4802973661668755e-14
```

4 Additional Usage Cases

NAG recently published the technical report Using the NAG Library with Kdb+ in a Pure Q Environment discussing how to call the NAG Library using the new Foreign Function Interface (FFI) from Kx. Additionally, the NAG Blog

titled Calling the NAG C Library from Kdb+ details how to incorporate the NAG Library with kdb+ within a C++ program. We speculate that among our shared clients, a mixture of these methods will be employed.

If your desired usage case happens to fall outside of those described within our current publications, please contact NAG support at support@nag.com for assistance with your application.

5 References / Bibliography

(1) Calling the NAG C Library from Kdb+ http://blog.nag.com/2013/05/calling-nag-c-library-from-kdb.html
(2) Get Going with Kdb+ https://code.kx.com/v2/
(3) Kdb+ and Python: embedPy and PyQ https://kx.com/blog/kdb-python-embedpy-pyq/
(4) NAG GitHub Organisation https://github.com/numericalalgorithmsgroup/
(5) NAG Library for *Python* Manual https://www.nag.com/numeric/py/nagdoc_latest/index.html
(6) Using Foreign Functions with Kdb+ (FFI) https://code.kx.com/q/interfaces/ffi/
(7) Using Python with kdb+ (PyQ) https://code.kx.com/q/interfaces/pyq/
(8) Using the NAG Library with Kdb+ in a Pure Q Environment https://www.nag.com/doc/techrep/pdf/tr1_18.pdf