The NAG Library for Python — Technical Document

The NAG Python bindings are available from [http://www.nag.co.uk/python.asp](http://www.nag.co.uk/python.asp).

Bindings are available for Windows (64 bit), Linux (64 bit), and Mac (64 bit).

The package will pick up the NAG C Library on your LD_LIBRARY_PATH for Linux, DYLD_LIBRARY_PATH for Mac, and PATH for Windows.

Below is a chart that should help converting types between Python and C. On the back of this page is an example calling a NAG optimization routine utilizing a callback function.

<table>
<thead>
<tr>
<th>NAG / C Types</th>
<th>Python Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAG Enums and NagError Structure</td>
<td>from nag4py.util import NagError, Nag_TRUE, Nag_FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>NAG Structures</td>
<td>from nag4py.e04 import Nag_E04_Opt</td>
</tr>
<tr>
<td></td>
<td>myoptions = Nag_E04_Opt()</td>
</tr>
<tr>
<td></td>
<td>myoptions.list = Nag_FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Integers/Doubles</td>
<td>from nag4py.util import NagError</td>
</tr>
<tr>
<td></td>
<td>from nag4py.s import nag_shifted_log</td>
</tr>
<tr>
<td></td>
<td>nag_shifted_log(1.1, NagError())</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Integers/Doubles</td>
<td>from ctypes import c_double</td>
</tr>
<tr>
<td></td>
<td>x_out = c_double(0.0)</td>
</tr>
<tr>
<td></td>
<td>NAG_CALL(x_out, NagError())</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer/Double Arrays</td>
<td>from ctypes import c_double, POINTER</td>
</tr>
<tr>
<td></td>
<td>x = numpy.array([1.1, 2.2, 3.3])</td>
</tr>
<tr>
<td></td>
<td>pointer_x = x.ctypes.data_as(POINTER(c_double))</td>
</tr>
<tr>
<td></td>
<td>NAG_CALL(pointer_x, NagError())</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Callback Functions</td>
<td>from nag4py.e04 import NAG_E04UCC_FUN</td>
</tr>
<tr>
<td></td>
<td>c_callback = NAG_E04UCC_FUN(py_callback)</td>
</tr>
</tbody>
</table>

Other Notes

- NagError will have a silent exit by default. To change this set `fail.eprint = Nag_TRUE`.
- Always check the return code (`fail.code`) for any error message (`fail.message`) on exit from a NAG C Library function.
- When passing data through ctypes, be cautious when using arrays. Make sure the data type of the numpy array matches the associated NAG C Library type.
Python example calling a NAG optimization routine (nag_opt_lsq_no_deriv/e04fcc).

```python
from ctypes import c_double, POINTER, cast, py_object
from numpy import array, zeros, double
from nag4py.e04 import nag_opt_lsq_no_deriv, e04xxc, NAG_E04FCC_FUN, Nag_E04_Opt
from nag4py.util import Pointer, NagError, Nag_Comm, Nag_FALSE, Nag_NoPrint, Nag_TRUE

def py_lsqfun1(m, n, x, fvec, comm):
    """
    Python least squares function
    """
    userdata_ptr = cast(comm[0].p, py_object)
    userdata = userdata_ptr.value
    t1 = userdata[0]
    t2 = userdata[1]
    t3 = userdata[2]
    y = userdata[3]
    for i in xrange(m):
        fvec[i] = (x[0] + t1[i] / (x[1] * t2[i] + x[2] * t3[i]) - y[i])

def main():
    # set up data as numpy arrays
    y = array([0.14, 0.18, 0.22, 0.25, 0.29, 0.32, 0.35, 0.39, 0.37, 0.58, 0.73, 1.34, 2.10, 4.39])
    t1 = array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0])
    t2 = array([15.0, 14.0, 13.0, 12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0])
    t3 = array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0])
    # number of variables in problem
    n = 3
    # number of observations in problem
    m = 15
    # Set up the starting point
    x = array([0.5, 1.0, 1.5])
    p_x = x.ctypes.data_as(POINTER(c_double))
    # set up working memory
    fvec = zeros(m, double)
    p_fvec = fvec.ctypes.data_as(POINTER(c_double))
    fjac = zeros(m * n, double)
    p_fjac = fjac.ctypes.data_as(POINTER(c_double))
    # fsumsq: After call to e04fcc this will contain
    # sum of squares of the residuals
    fsumsq = c_double(0.0)
    # tdfjac: The last dimension of the array fjac
    tdfjac = n
    
    # Create the NagError structure
    fail = NagError()
    fail.eprint = Nag_TRUE
    
    c_lsqfun1 = NAG_E04FCC_FUN(py_lsqfun1)
    
    comm = Nag_Comm()
    userdata = t1, t2, t3, y
    comm.p = cast(id(userdata), Pointer)
    
    # Set up options
    options = Nag_E04_Opt()
    e04xxc(options)
    options.list = Nag_FALSE
    options.print_level = Nag_NoPrint
    
    nag_opt_lsq_no_deriv(m, n, c_lsqfun1, p_x, fsumsq, p_fvec, p_fjac, tdfjac, options, comm, fail)
    
    print "Final Solution:"
    for i in range(0,n):
        print "  %.3E" % x[i]
    print "The sum of squares is %.3E" % fsumsq.value
```

Note the Python callback function has the same signature as the C callback.

C Function Signature

```c
void nag_opt_lsq_no_deriv(Integer m, Integer n, 
void(*lsqfun)(Integer m, Integer n, 
const double x[], double fvec[], 
Nag_Comm *comm),

double x[], double *fsumsq, double 
*fvec[], double fjac[], Integer 
tdfjac, Nag_E04_Opt *options,

Nag_Comm *comm, NagError *fail)
```

- Set pointers to numpy arrays
- Print any error messages
- Set the callback to the Python function
- Pass user data to the objective function via comm.p
- Call the NAG routine