NAG Toolbox *for MATLAB* and NAG Library *for Python*

Viktor Mosenkis

June 2016

nag®

Experts in numerical algorithms and HPC services

Agenda for NAG Toolbox *for MATLAB*

- The contents and how to use the NAG Toolbox *for MATLAB*
- Functionality and Performance Comparison

- Aim to show how you can benefit from the NAG Toolbox, and find appropriate help.

NAG Toolbox *for MATLAB*

- 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 and Regression Analysis
- Multivariate Analysis of Variance
- Random Number Generators
- Univariate Estimation
- Nonparametric Statistics
- Smoothing in Statistics
- Contingency Table Analysis
- Survival Analysis
- Time Series Analysis
- Operations Research

NAG Toolbox *for MATLAB*

- Mark 24 of NAG Toolbox *for MATLAB* out now.
- New content in areas of:
  - Global Optimization
  - Image processing
  - Dense Linear Algebra
  - Sparse Linear Algebra
  - Nearest Correlation Matrices
  - Random Number Generators
  - Nonparametric Statistics
  - FFTs
  - ODE
  - Integration
  - Roots of Equations
  - Option Pricing
  - Wavelets
  - Special functions

NAG Toolbox *for MATLAB*

- Algorithm are chosen for accuracy as well as performance.

- New content driven by users.
- Algorithms come from:
  - Collaboration with academic community
  - Participation in funded research projects
  - In-house development and implementation

NAG Toolbox *for MATLAB*

- Built as MATLAB mex files.
- This is actually auto-generated from XML documentation.
- Available for Windows, Linux and Mac.
- Installed under the usual MATLAB toolbox directory.
- Makes use of a DLL or shared version of the NAG Library.
- Windows and Linux versions exploits multicore.

NAG Toolbox *for MATLAB*

- The toolbox offer complementary functionality to MATLAB.
- Could be an alternative to several specialist toolboxes.
- We are not trying to compete with MATLAB!

NAG Toolbox *for MATLAB*

- The Toolbox is divided into chapters, each devoted to a branch of maths or statistics. Each has a 3-character name and a title, e.g., F03 – Determinants.
- Exceptionally, Chapters H and S have one-character names.
- All routines in the Toolbox have five-character names, beginning with the characters of the chapter name, e.g.,d01aj.
- There are also “long names” that aim to be more descriptive.

NAG Toolbox *for MATLAB*

- Documentation has an informative introduction to each chapter:
  - Technical background to the area.
  - Assistance in choosing the appropriate routine
- And a document for each routine with:
  - Description of method and references
  - Specification of arguments
  - Explanation of error exit
  - Remarks on accuracy
  - An example to illustrate use of routine, some enhanced with graphics

NAG Toolbox *for MATLAB*

- All documentation is available through the MATLAB help system.
- There are also some demos included.
- If equations are difficult to read, use the alternative PDF versions of the documents by clicking on the links at the top of each page, or accessing them via the NAG website.
- Description of examples can be find the documentation of the corresponding Fortran routine.

NAG Toolbox *for MATLAB*

- Let's take a look ...

< show help >
< lookfor >
< help contents >
< chapter intros >
< example prog here >

A Simple Example

- Here is an example of how to use the NAG Library to compute the solution of a real system of linear equations, Ax = b, where A is an n by n matrix and x and b are n vectors.

```
a = [ 1.80,  2.88,  2.05, -0.89;
      5.25, -2.95, -0.95, -3.80;
      1.58, -2.69, -2.90, -1.04;
     -1.11, -0.66, -0.59,  0.80];
b = [ 9.52;
     24.35;
      0.77;
     -6.22];
```

A Simple Example

- And we call like this:

```
[lu, ipiv, x, info] = f07aa(a, b);
x
x =

    1.0000
   -1.0000
    3.0000
   -5.0000
```

- Here the NAG routine f07aa takes two arguments, the matrix of coefficients, A, and the vector representing the right-hand side, b.

Arguments

- In our Fortran library this call would look like:

```
F07AAF(N, NRHS, A, LDA, IPIV, B, LDB, INFO)
```

- And input arguments are over-written.

- Some arguments in the toolbox have been omitted as they can always be determined at runtime:
  - Dimensions of arrays.
  - Workspace whose size depends on the problem data.
  - Arguments whose value depends entirely on that of other input data.
- Others are optional ...

Optional Arguments

- Optional arguments are provided after all compulsory arguments.
- Optional arguments appear in pairs: a string representing the name followed by the value.
- The pairs can be provided in any order.
- There are optional arguments where:
  - A sensible default value exists which applies to many problems.
  - The argument only applies to some cases.
  - The value of the argument can normally be determined from that of other arguments at runtime.

Optional arguments

- More arguments made optional in Mark 23.
- For example, in the system of equations given in the previous section, it is obvious that the size of the matrix A, n, is 4.
- However we can tell MATLAB that n is 3, in which case it will solve the system represented by the top-left 3x3 section of A, and the first three elements of b.

- And we call like this …

Optional arguments

```
[lu, ipiv, x, info] = f07aa(a, b, 'n', nag_int(3));
x

x =

    4.1631
   -2.1249
    3.9737
   -6.2200
```

- The last element of x can (should) be ignored. Since b was a 4x1 matrix on input, it will be a 4x1 matrix on output, even though the last element is not being used.

Arguments

- A similar outcome can be achieved by:

```
[lu, ipiv, x, info] = f07aa(a(1:3,1:3), b(1:3));
x

x =

    4.1631
   -2.1249
    3.9737
```

- Here x is of appropriate size.

Another Example – Overriding Defaults

- g01hb computes probabilities associated with a multivariate distribution to a relative accuracy which defaults to 0.0001:

```
g01hb(tail, a, b, xmu, sig)

ans =
    0.9142
```

- We can vary tol:

```
g01hb(tail, a, b, xmu, sig,'tol',0.1)

ans =
    0.9182
```

Errors and Warnings

- The toolbox routines can produce a number of errors. (Next slide.)
- In most cases the error message will give more precise details of how the error was triggered. For example a NAG:arrayBoundError might display the message:

```
??? The dimension of argument 2 (A)
should be at least 4
```

Errors and Warnings

- NAG:licenceError - A valid licence couldn't be found.
- NAG:arrayBoundError - Array provided is too small.
- NAG:callBackError - An error occurred when executing an M-File passed as a parameter to the routine.
- NAG:missingInputParameters
- NAG:optionalParameterError - Not in name/value pairs, or the name provided is not an optional parameter.
- NAG:tooManyOutputParameters
- NAG:typeError - A parameter is of the wrong type.
- NAG:unsetCellArrayError - A cell array has been passed without all elements being set.
- NAG:valueError - An incorrect value has been provided for a parameter.

Errors and Warnings

- The NAG routines can produce two warnings:
  - NAG:truncationWarning - A string was truncated when copying cell array of strings to a Fortran data structure.
  - NAG:warning - The NAG routine returned an error or warning.
- The latter is important, and means that on exit the value of the argument ifail (or, in chapters f07 and f08, info) was non-zero on exit.
- For details about how to interpret this value the user should consult the Error Indicators and Warnings section of the document for the particular routine.

Errors and Warnings

- However when an error is thrown none of the output parameters are available for inspection.
- By default, we only use the warning in cases where the output values may be of use
  - in determining the cause of the problem
  - as a "warm start" in subsequent calls to the routine, or
  - where the routine has found a solution but there are caveats, for example as to its accuracy.
- In all other cases we now issue the error.

Errors and Warnings

- If you do not wish to see a warning then you can disable it in the usual way, for example:

```
warning('off', 'NAG:warning')
```

- In this case it is vital that you check the value of ifail or info on exit from the routine.

< run g02aa with errors and warnings here>

Types

- The interfaces to NAG routines in the Toolbox are quite precise about the types of their arguments.
- Since MATLAB assumes by default that every number is a double users need to covert their input data to the appropriate type if it is an integer, a complex number or a logical.
- Similarly, in M-Files called by the NAG routine, the user must ensure that the results returned are of the appropriate type. This is to ensure the correct alignment between the MATLAB and Fortran types.

Types

- Specification of NAG types are described in the Documentation
  - Introduction/Calling NAG Routines From Matlab

MATLAB Data Types - Integers

- There are 8 integer data types in MATLAB:

```
int8, int16, int32, int64,
uint8, uint16, uint32, uint64
```

- The number refers to the number of bits that are used to store the value. uint* are unsigned integers.
- For example the int32 function: `myint = int32(n)`
- You can find the range of values supported by these data types with intmin and intmax:

```
>>[intmin('int8') intmax('int8')]
ans =
    -128   127
```

Integers

- For portability across versions of the toolbox that use both 32 and 64 bit integers we provide two functions:
  - nag_int(x) - converts x to the integer type compatible with the current version of the NAG toolbox
  - nag_int_name - returns the name of the integer class compatible with your version of the NAG toolbox

```
>> a = nag_int(3)
a = 3
>> nag_int_name()
ans = int64
>> b = zeros(10, nag_int_name());
>> whos
  Name          Size                    Bytes  Class     Attributes

  a             1x1                         8  int64
  ans           1x5                        10  char
  b            10x10                      800  int64
```

MATLAB Data Types - Logical

- Logical data types:

```
>>a = true; b = false; a | b
ans =
     1
```

- In many areas of MATLAB we can use integers interchangeably with logical variables. In an `if` test for example.
- However to create a logical from other datatypes, use the logical function. `mylog = logical(0)` This returns true for all but variables with value zero.

MATLAB Data Types - Complex

- The complex function constructs a complex result from real and imaginary parts.
- The statement

```
c = complex(x,y)
```

  returns the complex result x + yi, where x and y are identically sized real arrays or scalars of the same data type.
- y is optional, and without this argument a complex variable is retuned with zero imaginary part.
- You can test a variable with the isreal function, which returns false if the variable is complex.

Integers

- Integers used by the Toolbox are chosen to be compatible with those used internally by MATLAB, and will be int32 or int64.
- MATLAB does not support some operations on int64, in particular the colon operator is not defined and so they cannot be used as the start or end of a loop.
- It is often therefore necessary to use double variables and just cast to int32/64 while calling the NAG function.

Creating variables of correct type

- nag_int(1) - to create an integer of value 1
- complex(1,1) - to create 1.0000 + 1.0000i
- logical(0) - to create a logical that is .FALSE.

- If an object of the incorrect type is provided then a NAG:typeError will be thrown:

```
s01ea(0)
??? argument number 1 is not a complex
scalar of class double.
```

- MATLAB sparse arrays not supported, B = full(A)

FUNCTIONALITY AND PERFORMANCE

Comparisons

- In this section we will look at some of the functionality of the NAG Library.
- We will offer some comparisons of the Toolbox compared with (mainly) core MATLAB.

Numerical Linear Algebra

- The dense linear algebra in MATLAB built on LAPACK.
- However, you can only get part of LAPACK.
- The NAG Library implements the whole of LAPACK.
- For example, you want to compute a subset of eigenvalues?

- We expect the performance to be the same, don't we?
- Let us look at solving equations ...

Eigenvalues and Vectors – Chapter F08

- We compare:
- The MATLAB function eig

```
[v,d] = eig(a);
```

- f08fa (DSYEV in LAPACK). QR algorithm like eig

```
[a, w, info] = f08fa('V', 'U', a)
```

- f08fc (DSYEVD), divide and conquer algorithm

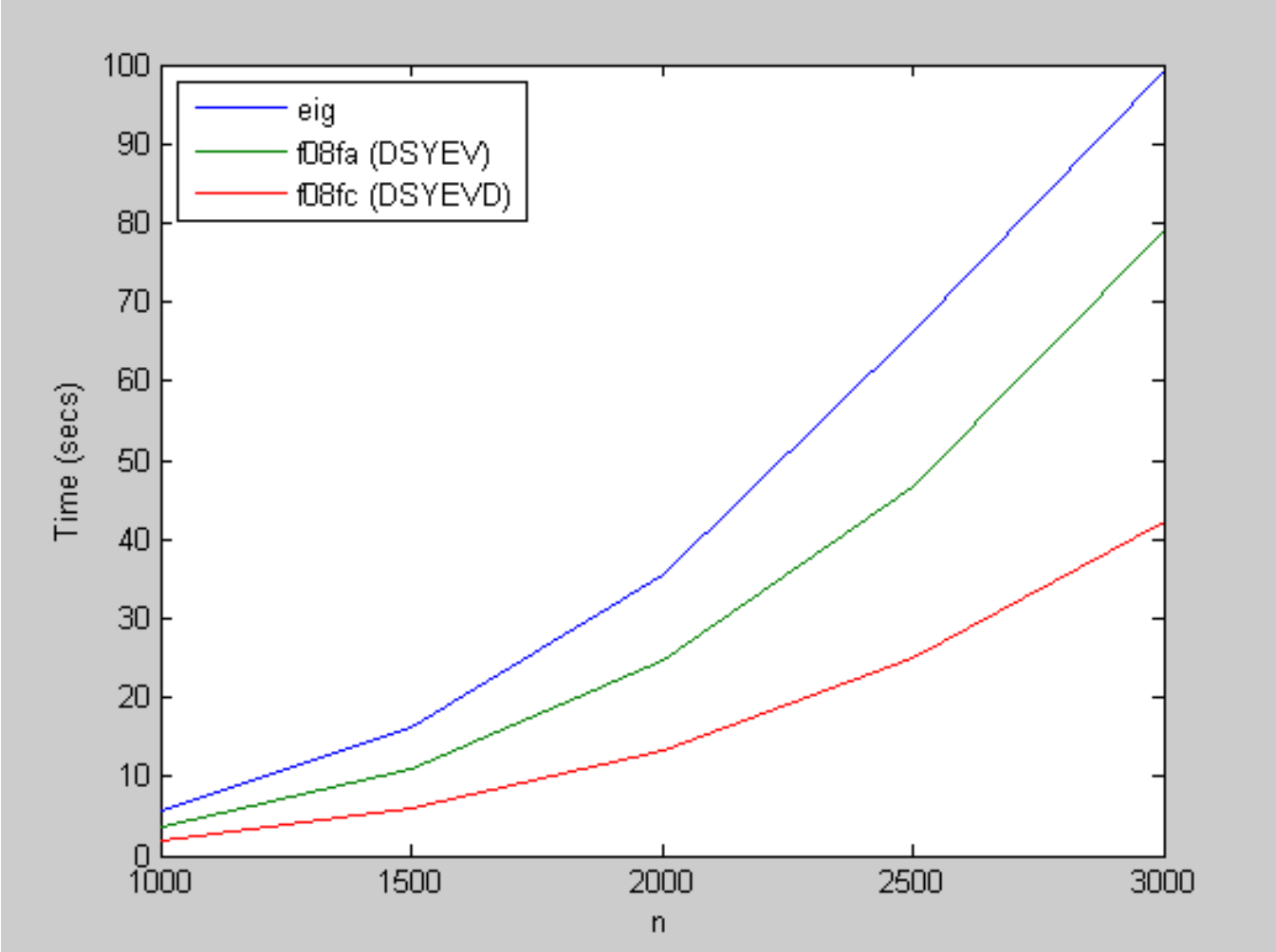
```
[a, w, info] = f08fc('V', 'U', a)
```

- Here are the results on my Intel Core Duo 2GHz laptop ...

Eigenvalues and Eigenvectors

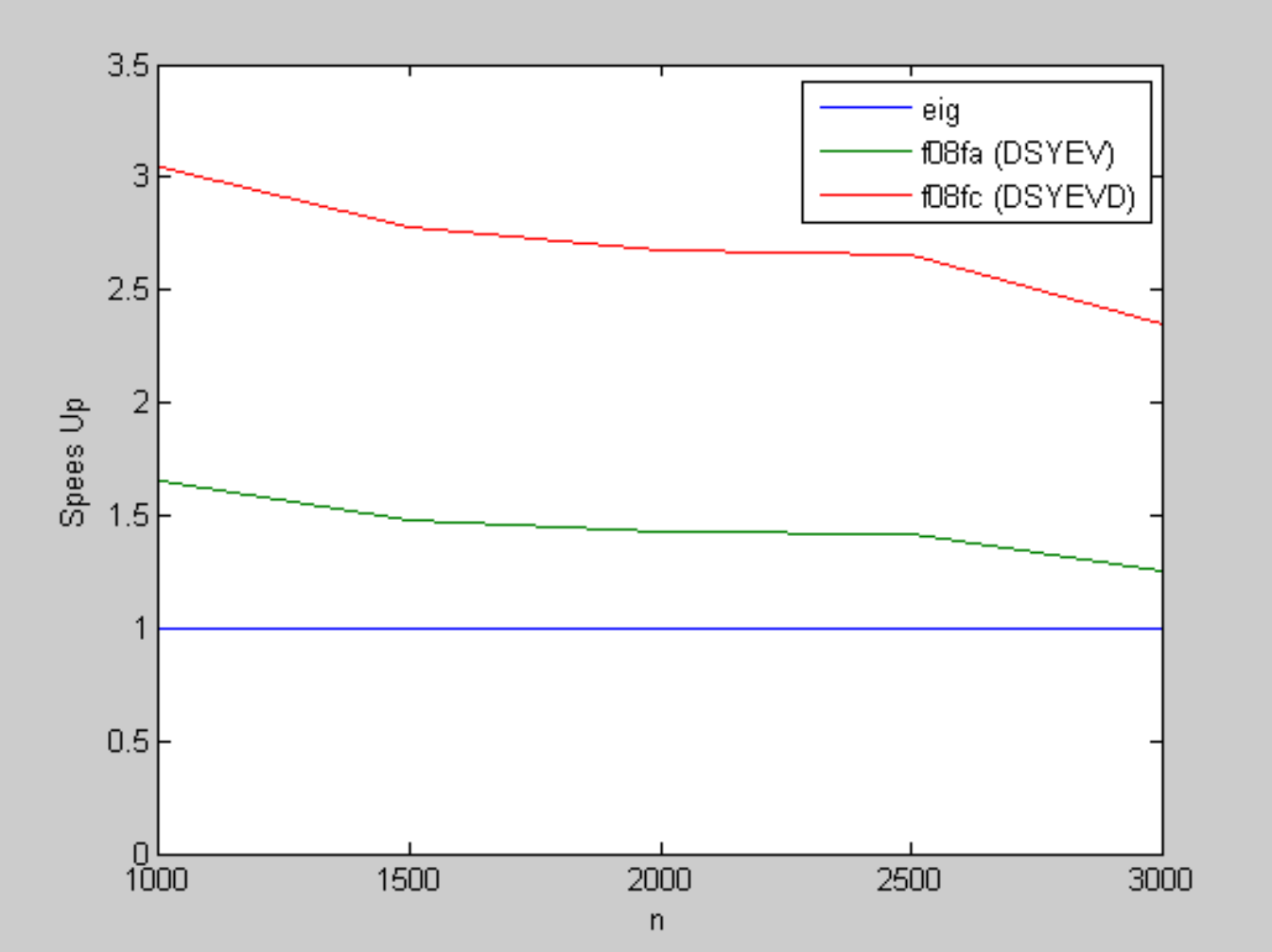
- Timings

(eig 130 secs
for n=3000
on one core)



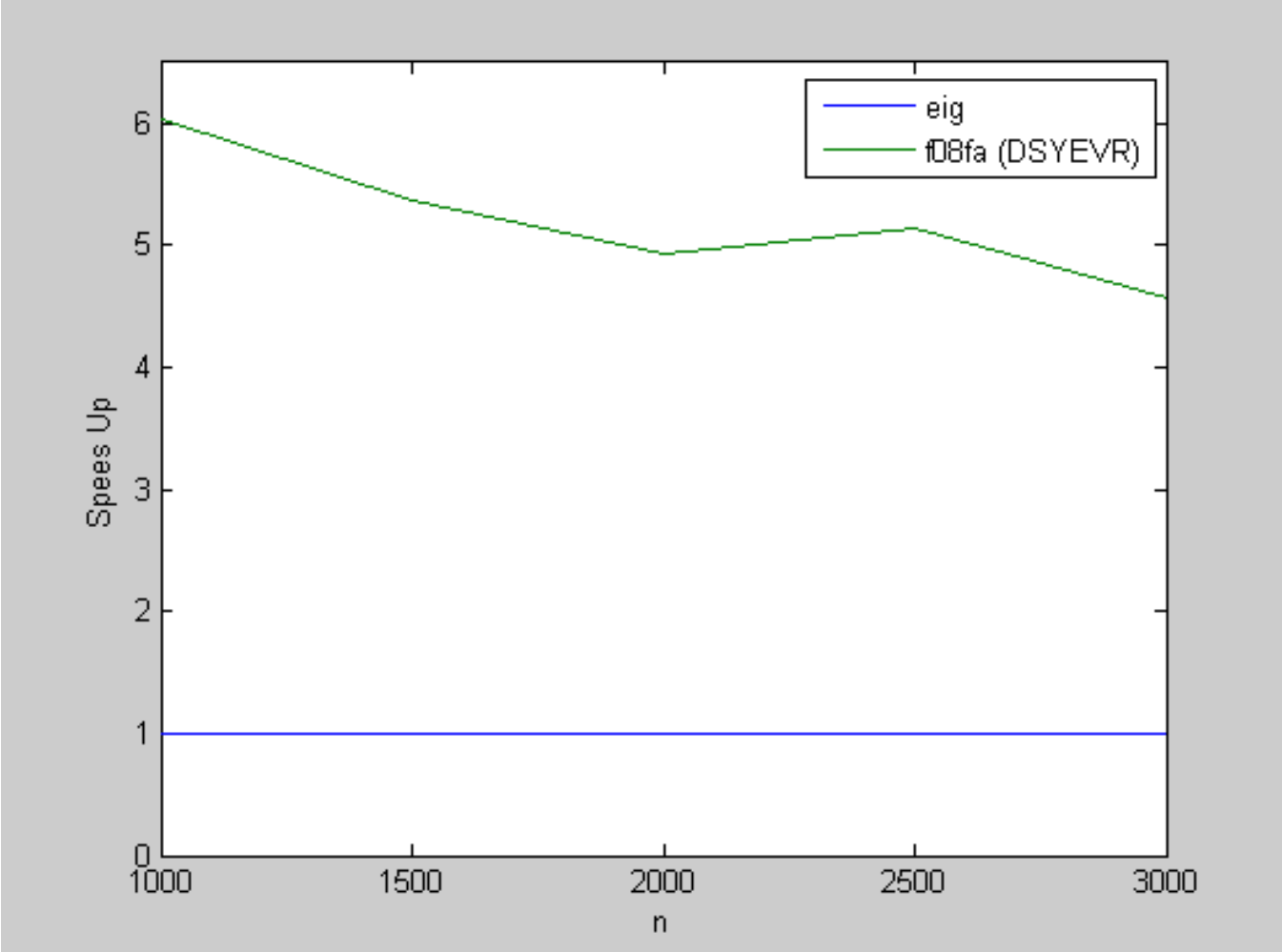
Eigenvalues and Eigenvectors

- Speedup



Subset of Eigenvalues and Eigenvectors

- Speedup



Multicore Parallelism

- Firmly in the multicore age, soon to be many core. We have to be thinking parallel.
- Algorithmic development at NAG always done with parallelism in mind.
- Internal research project looking at many core.
- Involved in OpenMP development.

- Around 1/3 of the library parallel...

Multicore Parallelism

- Root Finding
- Summation of Series (e.g. FFT)
- Quadrature
- Ordinary Differential Equations
- Partial Differential Equations
- Numerical Differentiation
- Integral Equations
- Mesh Generation
- Interpolation
- Curve and Surface Fitting
- Optimisation
- Approximations of Special Functions
- Dense Linear Algebra
- Sparse Linear Algebra
- Correlation and Regression Analysis
- Multivariate Analysis of Variance
- Random Number Generators
- Univariate Estimation
- Nonparametric Statistics
- Smoothing in Statistics
- Contingency Table Analysis
- Survival Analysis
- Time Series Analysis
- Operations Research

CONCLUSIONS

Conclusions

- MATLAB is clearly a very powerful piece of software.
- Excellent for data generation, post processing, algorithm profiling and ease of use.
- MATLAB has a huge range of mathematical functions.
- However, much of the functionality is provided by toolboxes that need to be purchased separately.
- MATLAB also doesn't provide a large choice of algorithms, or limited use of its underlying libraries, potentially affecting performance, some algorithms are simply slower.

Conclusions

- NAG extensive experience at implementing numerical code.
- We write software that is:
  - useful
  - robust
  - accurate
  - stable
  - fast
- Our software covers many area of mathematics and statistics, and all this functionality is available from within MATLAB.

Conclusions

- Performance is sometimes achieved by providing a choice of algorithms for a particular problem.
- Documentation is provided to guide you in selecting the correct algorithm.
- Example programs help you become familiar with a routine quickly.
- Information about error bounds of computed solutions are provided to help interpret your results.

NAG Library for Python

Agenda for NAG Library *for Python*

- Installation tips.
- First steps with NAG Library for Python.
  - Examples
  - Documenation

NAG Library *for Python*

- Built as interface to the NAG Library *for C*
- Available on the following platforms
  - 64-bit Windows (CLW6I25DAL)
  - 32-bit Windows (CLW3225DAL)
  - 64-bit Linux (CLL6I25DCL)
  - 64-bit Mac (CLMI623DGL)
- Dependencies
  - NAG Library *for C*
  - Numpy
  - Matplotlib (to run some graphical examples)

Installation I

- Install NAG Library *for C* for your platform
- Install python
- Install numpy (requires C compiler if not precompiled. Complicated on Windows)
- Download and install python bindings
  - Follow instruction in Readme

Installation II (easy way!!!)

- Install the NAG Library *for C* for your platform

- Install Anacoda (contains numpy and pydoc and other useful software (spyder))
  - https://www.continuum.io/downloads
- Open command line and install the python bindings by typing
  - conda install -c nag nag4py

Check the installation and licence

- Quick test of nag4py to show implementation details and check the NAG licence

```
>>> from nag4py.a00 import a00aac, a00acc
>>> a00aac()
>>> a00acc()
```

Access documentation

- Python help mechanism to information on library/chapters/functions

```
>>> import nag4py
>>> help(nag4py)

>>> import nag4py.c05
>>> help(nag4py.c05)

>>> from nag4py.c05 import c05auc
>>> help(c05auc)
```

- To get more help please consult the documentation of the corresponding chapter/function in the NAG Library *for C*

Examples

- Let’s take a look …

Types

- NAG Enums and NAG Error Structure – available in the nag4py.util module

```
from nag4py.utill import noisy_fail, NAG_TRUE
```

- NAG Structures – available in associated module for the chapter of interest

```
from nag4py.e04 import Nag_E04_Opt
myoptions = Nag_E04_Opt()
myoptions.list = Nag_False
```

- Input Integers/Doubles – input directly into Python

```
from nag4py.s import nag_cumul_normal
nag_cumul_normal(1.1)
```

Types

- Input/Output Arguments – these must first be set to numpy arrays with the correct data type. TypeError will be raised by the nag4py wrapper whenever an argument is incorrectly typed.

```
from numpy import array, empty
from nag4py.util import nag_int_type
i_arr = array([1]*10, dtype=nag_int_type)
d_arr = empty(n)
```

- Callback Functions – available in the associated chapter

```
from nag4py.e04 import NAG_E04UCC_FUN
c_callback = NAG_E04UCC_FUN(py_callback)
```