Bitwise Reproducibility and the NAG Libraries

Mick Pont, NAG Oxford

mick@nag.co.uk

nag®

Experts in numerical algorithms and HPC services

Introduction to NAG

- Numerical Algorithms Group - Founded 1970
  - Co-operative software project: Birmingham, Leeds, Manchester, Nottingham, Oxford, and Atlas Laboratory
- Incorporated as NAG Ltd. in 1976
  - Not-for-profit
  - Based in Oxford, with offices in Manchester, Chicago, Tokyo, Taiwan
- Main product still the NAG Libraries
  - Also compiler, software tools, consultancy
  - CSE support

NAG Library Contents Overview

- C05 - Root Finding
- C06 - FFTs
- D01 - Quadrature
- D02 - ODEs
- D03 - PDEs
- D05 - Integral Equations
- D06 - Mesh Generation
- E01 - Interpolation
- E02 – Data Fitting
- E04 – Local Optimization
- E05 – Global Optimization
- F01-F12 - Linear Algebra
- G02 - Correlation and Regression Analysis
- G04 - Analysis of Variance
- G05 Random Number Generators
- G07 - Univariate Estimation
- G08 - Nonparametric Statistics
- G10 - Smoothing in Statistics
- G11 - Contingency Table Analysis
- G13 - Time Series Analysis
- H - Operations Research
- S - Special Functions

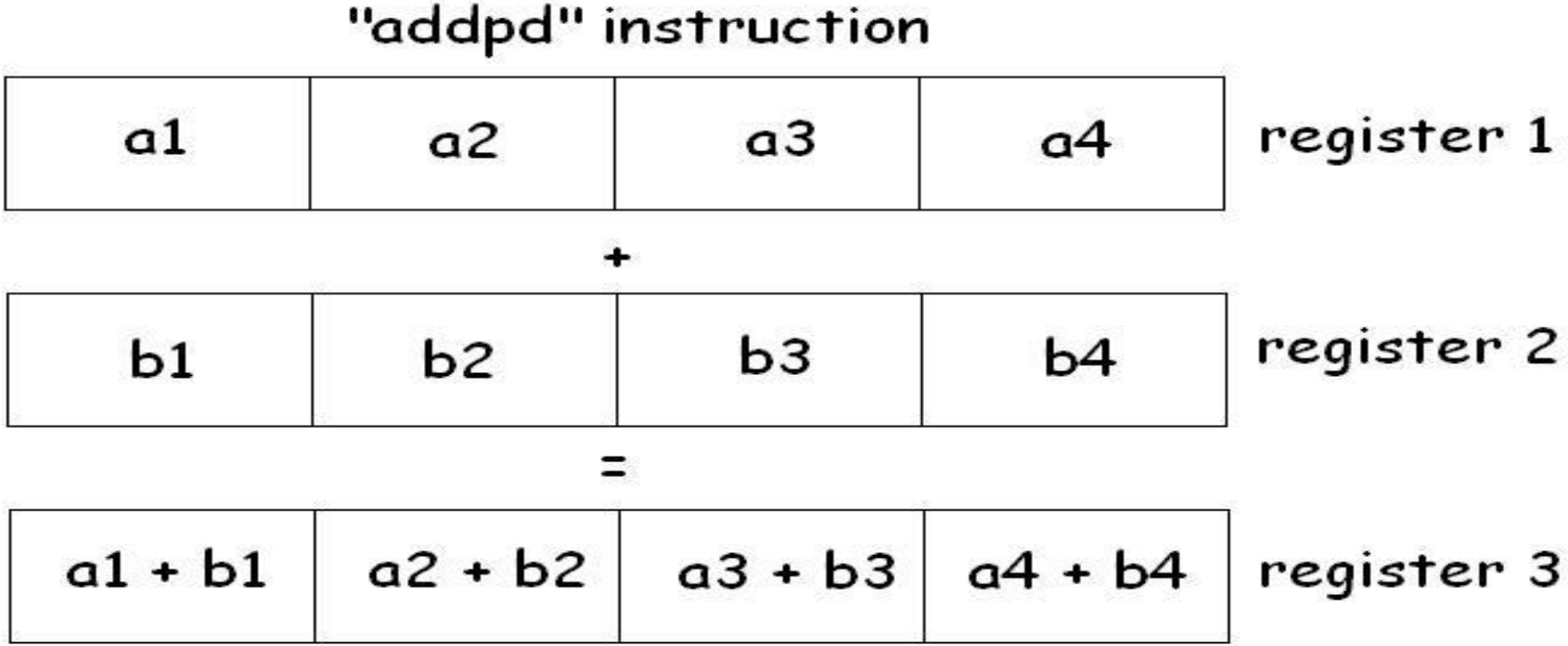
Reproducibility of results

- NAG users often want reproducible results across machines
  - But computers have finite precision
  - IEEE standard for floating-point arithmetic helps, but …
  - Vectorized register arithmetic can cause trouble
  - Compilers don’t always do the same things
- Usually differences are small
  - But not always, e.g. if a conditional statement depends on an imprecise result

SSE and AVX instructions

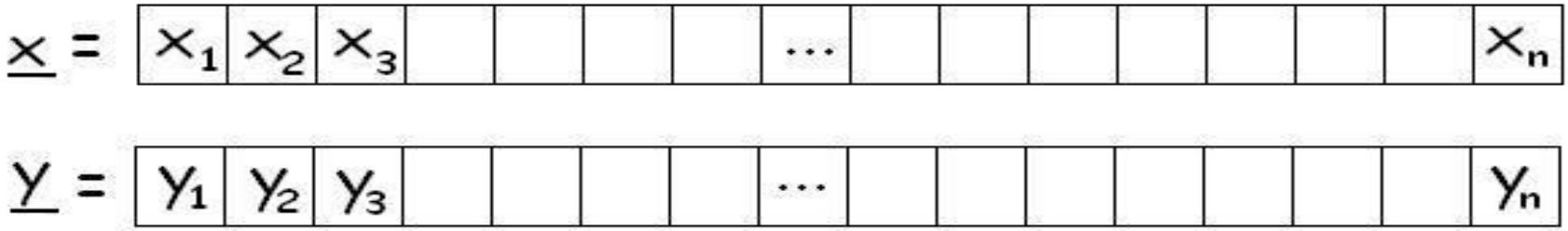
- Vectorized instructions operate on several numbers at once
- Clever compilers can take advantage of them
  - this is one of the few ways that individual processors can get faster now
- Can't or won't use them?
  - you'll not get anywhere near peak performance from your hardware

SSE / AVX



But to use these instructions memory alignment is crucial …

Example - dot product of two vectors



dot product $p = x_1 * y_1 + x_2 * y_2 + \ldots + x_n * y_n$

(if memory is not nicely aligned)

or $\tilde{p} = (x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4) + (x_5y_5 + x_6y_6 + x_7y_7 + x_8y_8) + \ldots$

(if memory is nicely aligned - this should be much faster)

Mathematically equivalent – but the two results are not necessarily identical. Does it matter? Sometimes!

ddot function in C

```
float myddot(int n, float *x, float *y) {
  int i;  float d = 0.0F;
  for (i=0; i<n; i++)
    d += x[i] * y[i];
  return d;
}
```

You can download myddot.c and a small test code from here:

- http://monet.nag.co.uk/mick/SC13_BWR/
  - Versions for Windows or Linux, each with build script
  - Assumes use of Intel C compiler

Compile ddot and the main program *(Windows)*

- First with default flags:
  - icl –O3 –c myddot.c /Fo:myddot.obj
  - icl test_align.c myddot.obj /Fe:test_align.exe
- Then with fp:precise flag:
  - icl –O3 /fp:precise –c myddot.c /Fo:myddot_precise.obj
  - icl test_align.c myddot_precise.obj /Fe:test_align_precise.exe

Compile ddot and the main program *(Linux)*

- First with default flags:
  - icc –O3 –c myddot.c –o myddot.o
  - icc test_align.c myddot.o –o test_align.exe
- Then with –fp-model precise flag:
  - icc –O3 –fp-model precise –c myddot.c -o myddot_precise.o
  - icc test_align.c myddot_precise.o -o test_align_precise.exe

Running the “non-precise” version

test_align.exe

```
Address(x) = 0x000c8090   address(y) = 0x000c8170 d =  1.665999794006e+001
Address(x) = 0x000c8094   address(y) = 0x000c8170 d =  1.665999794006e+001
Address(x) = 0x000c8098   address(y) = 0x000c8170 d =  1.665999794006e+001
Address(x) = 0x000c809c   address(y) = 0x000c8170 d =  1.665999794006e+001
Address(x) = 0x000c8090   address(y) = 0x000c8174 d =  1.665999984741e+001
Address(x) = 0x000c8094   address(y) = 0x000c8174 d =  1.665999984741e+001
Address(x) = 0x000c8098   address(y) = 0x000c8174 d =  1.665999984741e+001
Address(x) = 0x000c809c   address(y) = 0x000c8174 d =  1.665999984741e+001
Address(x) = 0x000c8090   address(y) = 0x000c8178 d =  1.665999794006e+001
Address(x) = 0x000c8094   address(y) = 0x000c8178 d =  1.665999794006e+001
Address(x) = 0x000c8098   address(y) = 0x000c8178 d =  1.665999794006e+001
Address(x) = 0x000c809c   address(y) = 0x000c8178 d =  1.665999794006e+001
Address(x) = 0x000c8090   address(y) = 0x000c817c d =  1.665999984741e+001
Address(x) = 0x000c8094   address(y) = 0x000c817c d =  1.665999984741e+001
Address(x) = 0x000c8098   address(y) = 0x000c817c d =  1.665999984741e+001
Address(x) = 0x000c809c   address(y) = 0x000c817c d =  1.665999984741e+001
Smallest value of dot product =  1.665999794006e+001 = 0x418547ad
  Largest value of dot product =  1.665999984741e+001 = 0x418547ae
Difference =  1.907348632813e-006 = 0x36000000
```

Running the “precise” version

test_align_precise.exe

```
Address(x) = 0x00398100   address(y) = 0x003981e0 d =  1.665999794006e+001
Address(x) = 0x00398104   address(y) = 0x003981e0 d =  1.665999794006e+001
Address(x) = 0x00398108   address(y) = 0x003981e0 d =  1.665999794006e+001
Address(x) = 0x0039810c   address(y) = 0x003981e0 d =  1.665999794006e+001
Address(x) = 0x00398100   address(y) = 0x003981e4 d =  1.665999794006e+001
Address(x) = 0x00398104   address(y) = 0x003981e4 d =  1.665999794006e+001
Address(x) = 0x00398108   address(y) = 0x003981e4 d =  1.665999794006e+001
Address(x) = 0x0039810c   address(y) = 0x003981e4 d =  1.665999794006e+001
Address(x) = 0x00398100   address(y) = 0x003981e8 d =  1.665999794006e+001
Address(x) = 0x00398104   address(y) = 0x003981e8 d =  1.665999794006e+001
Address(x) = 0x00398108   address(y) = 0x003981e8 d =  1.665999794006e+001
Address(x) = 0x0039810c   address(y) = 0x003981e8 d =  1.665999794006e+001
Address(x) = 0x00398100   address(y) = 0x003981ec d =  1.665999794006e+001
Address(x) = 0x00398104   address(y) = 0x003981ec d =  1.665999794006e+001
Address(x) = 0x00398108   address(y) = 0x003981ec d =  1.665999794006e+001
Address(x) = 0x0039810c   address(y) = 0x003981ec d =  1.665999794006e+001
Smallest value of dot product =  1.665999794006e+001 = 0x418547ad
  Largest value of dot product =  1.665999794006e+001 = 0x418547ad
Difference =  0.000000000000e+000 = 0x00000000
```

(If you like) examine assembly for myddot

- link /dump /disasm myddot.obj > myddot.asm
- link /dump /disasm myddot_precise.obj > myddot_precise.asm
- *Above is on Windows - on Linux use e.g.*
  - *objdump –d myddot.o*

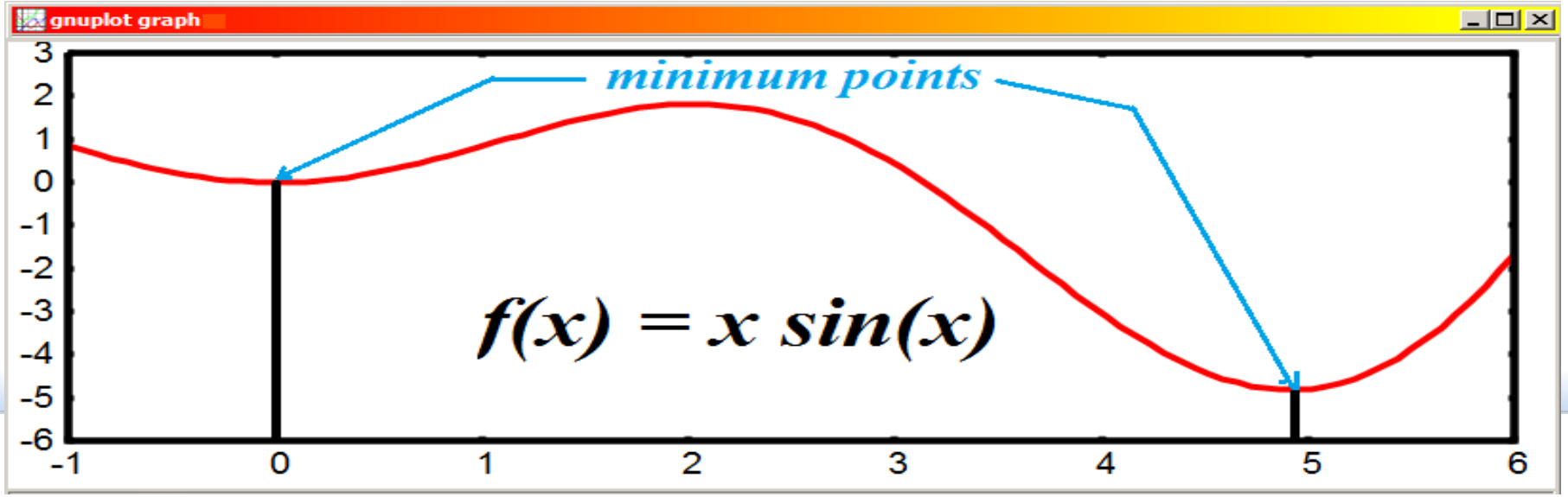
Compare and contrast the two versions – verify what is happening in the two cases. (Notice how sophisticated the non-precise one looks!)

NAG customers do notice

- We try to educate NAG users not to expect bitwise reproducibility
  - Our documentation on reproducibility tries to say why
- But some of them insist that they need BWR
  - e.g. *investment banks m*ay be constrained by regulatory procedures
- This can lead to a heavy tech support burden
  - Dot product problem in NAG routine traced by a senior quant at a major French bank (without source code!)
  - Another user insisted that a NAG sparse solver must be using a “stochastic” algorithm (it wasn’t)

What do we mean by “Optimization”?

For our purposes we mean “given a scalar real-valued mathematical function of *n* variables x_i, find values of the variables *x* that make the function as small (or as large) as possible”. To avoid reproducibility problems we now build NAG libraries with fp:precise flags.

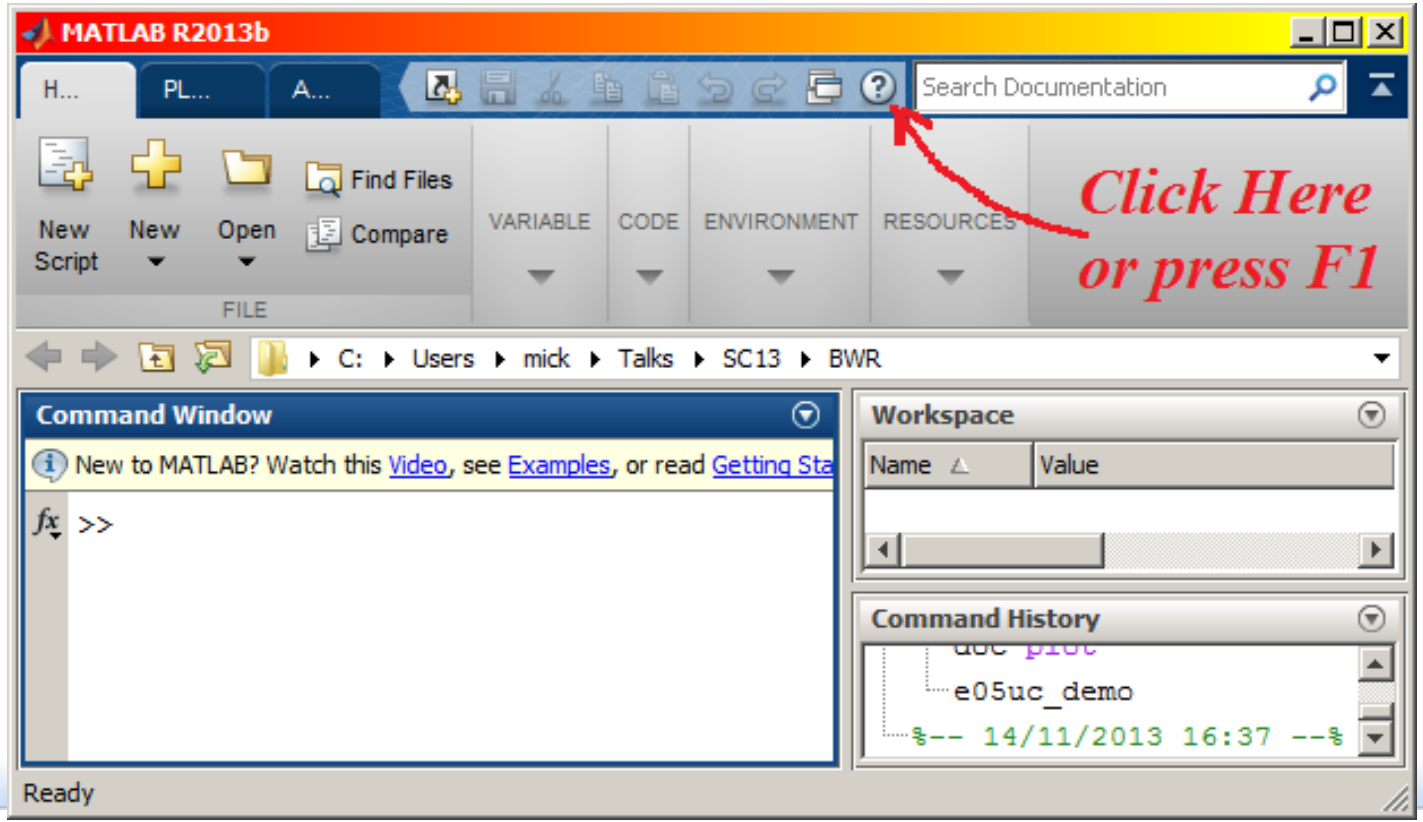


Downloading NAG software

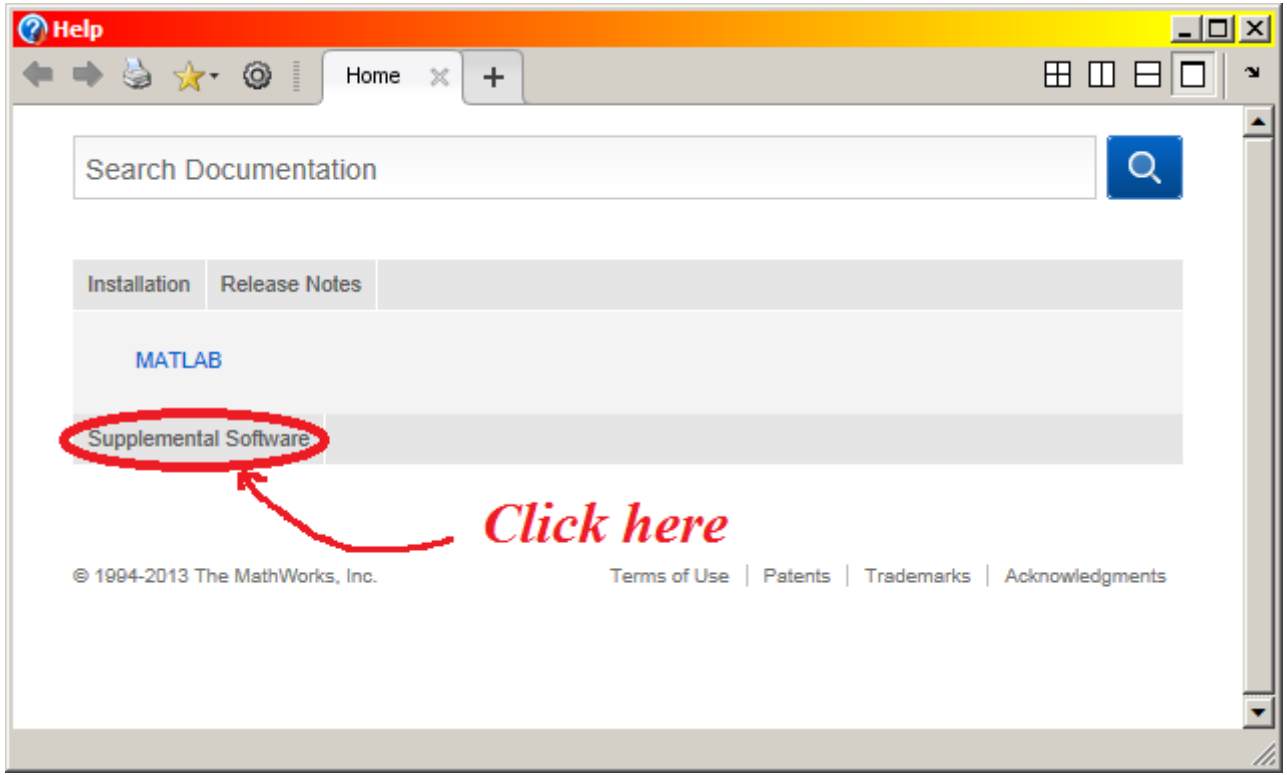
- Download NAG Toolbox for MATLAB:
  - http://www.nag.co.uk/downloads/mbdownloads.asp
- Or NAG Fortran Library:
  - http://www.nag.co.uk/downloads/fldownloads.asp
- Or NAG C Library:
  - http://www.nag.co.uk/downloads/cldownloads.asp

Write to support@nag.co.uk and ask for a *trial key* – mention Mick Pont and SC13

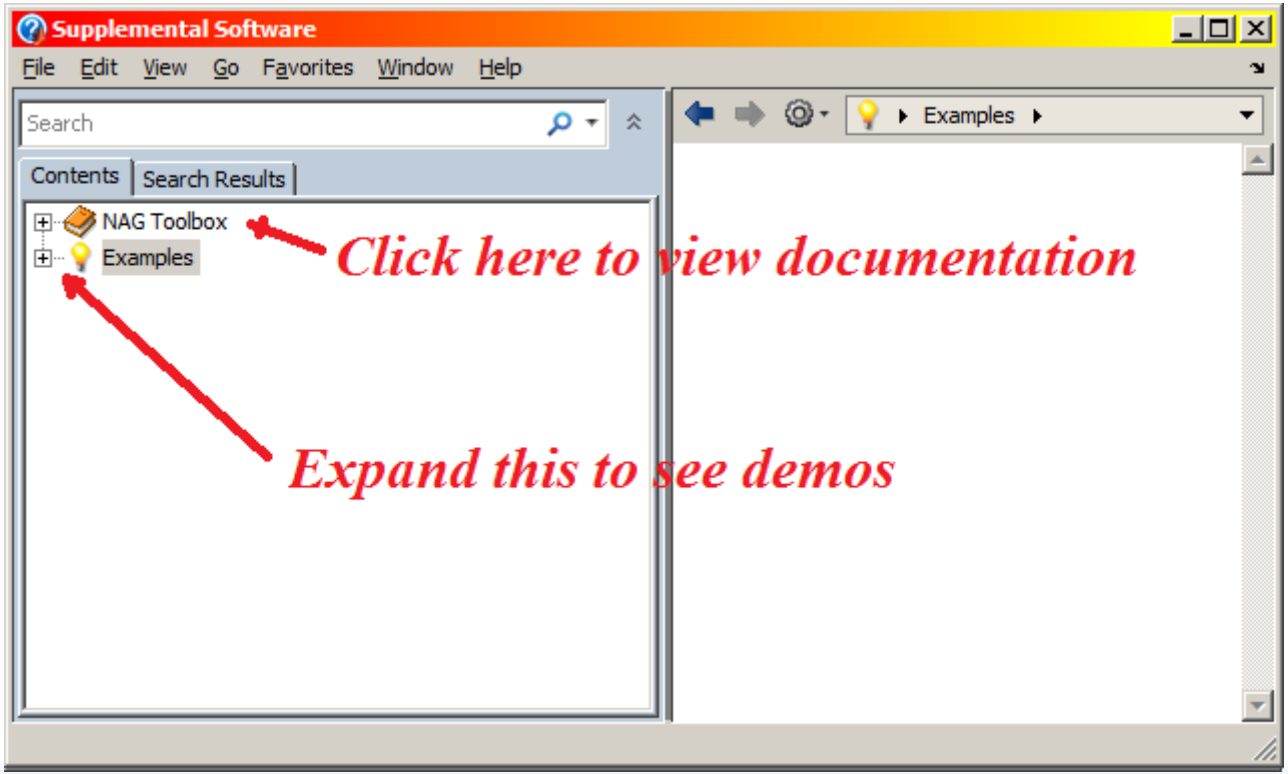
Accessing NAG Toolbox routines in MATLAB



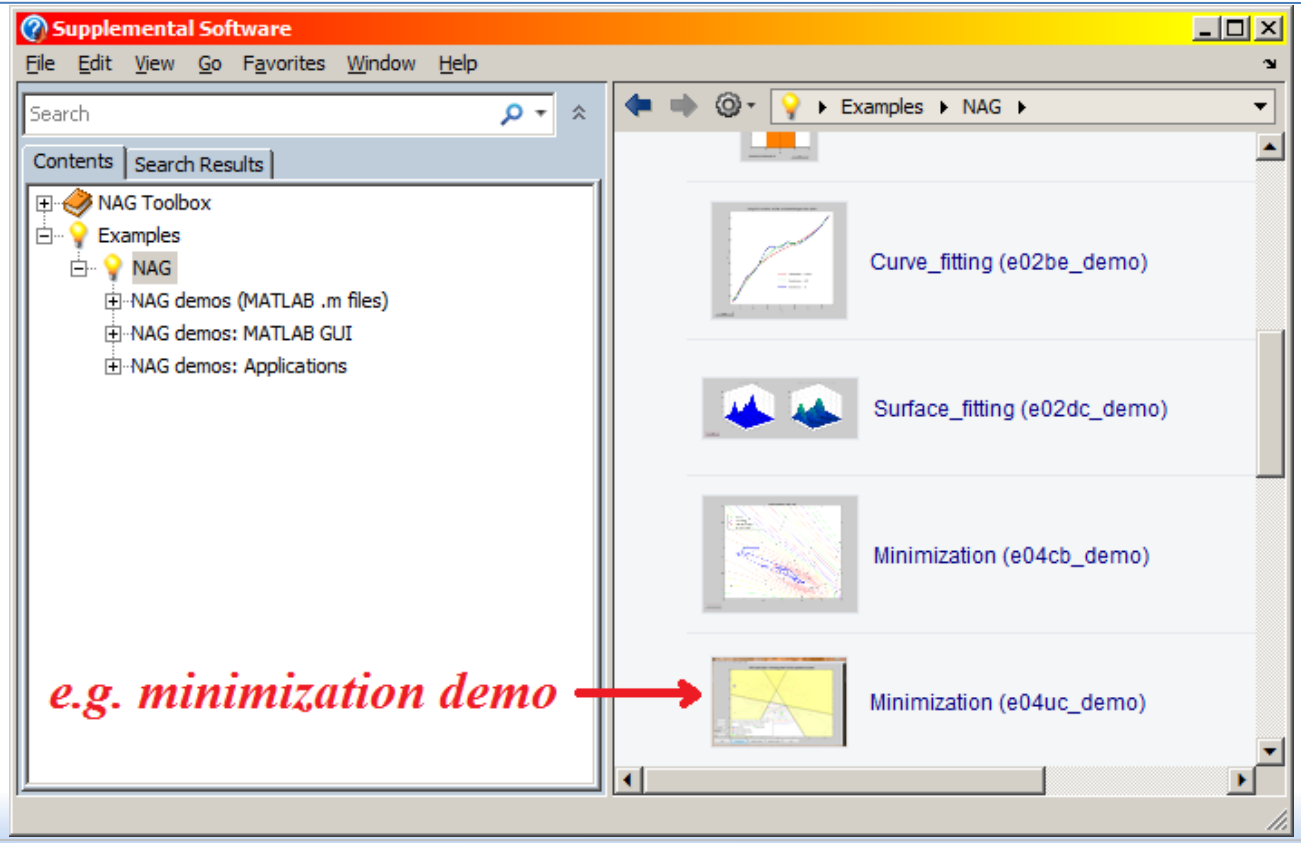
Accessing NAG Toolbox routines in MATLAB



Accessing NAG Toolbox routines in MATLAB



NAG Demos


Supplemental Software
File Edit View Go Favorites Window Help
Search
Contents
Search Results
NAG Toolbox
Examples
NAG
NAG demos (MATLAB .m files)
NAG demos: MATLAB GUI
NAG demos: Applications
Examples ▸ NAG ▸
Curve_fitting (e02be_demo)
Surface_fitting (e02dc_demo)
Minimization (e04cb_demo)
Minimization (e04uc_demo)
e.g. minimization demo