# NAG and the BLAS

Mick Pont
Development Division
NAG Ltd, Oxford
*May 19th 2016*

# NAG

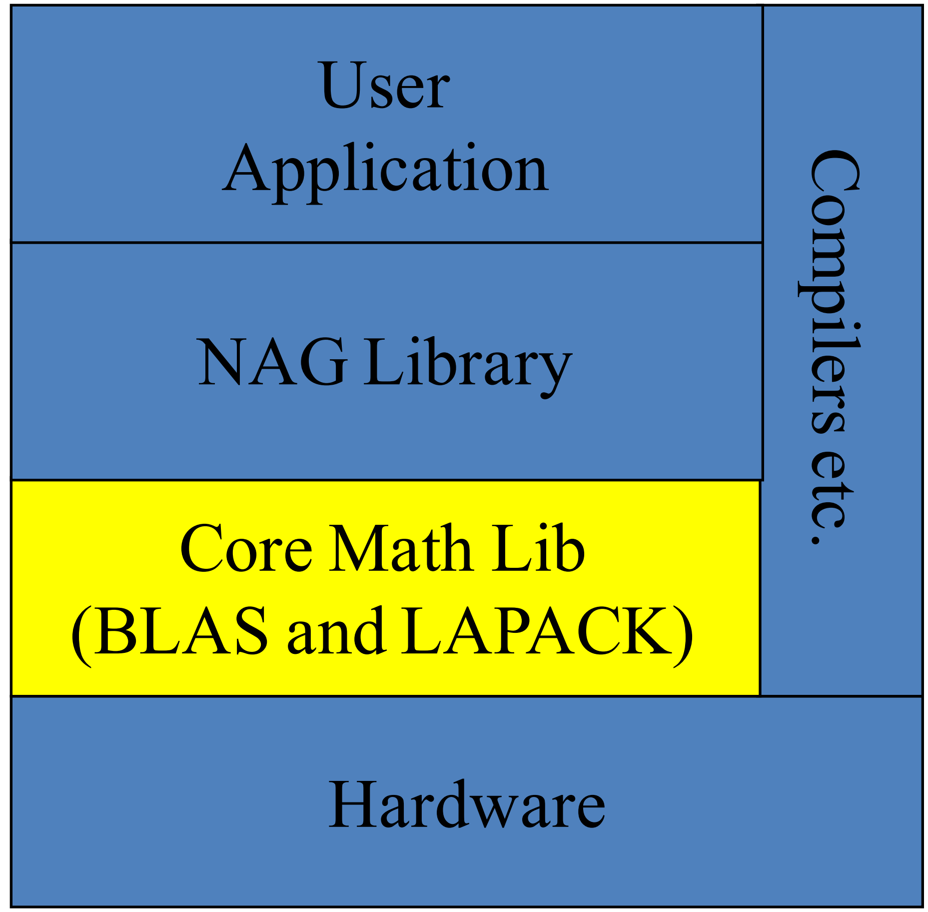
- Numerical Algorithms Group - Founded 1970
  - Co-operative software project: Birmingham, Leeds, Manchester, Nottingham, Oxford, and Atlas Laboratory
  - Inspired by Jim Wilkinson
- Incorporated as NAG Ltd. in 1976
  - Not-for-profit
  - Based in Oxford, with offices in Manchester, Chicago, Tokyo
- Mathematical algorithm development
  - Often collaborative
  - We also fund research students
- Software engineering – production and porting of software libraries
- HPC services and consultancy

# NAG Library Contents Overview

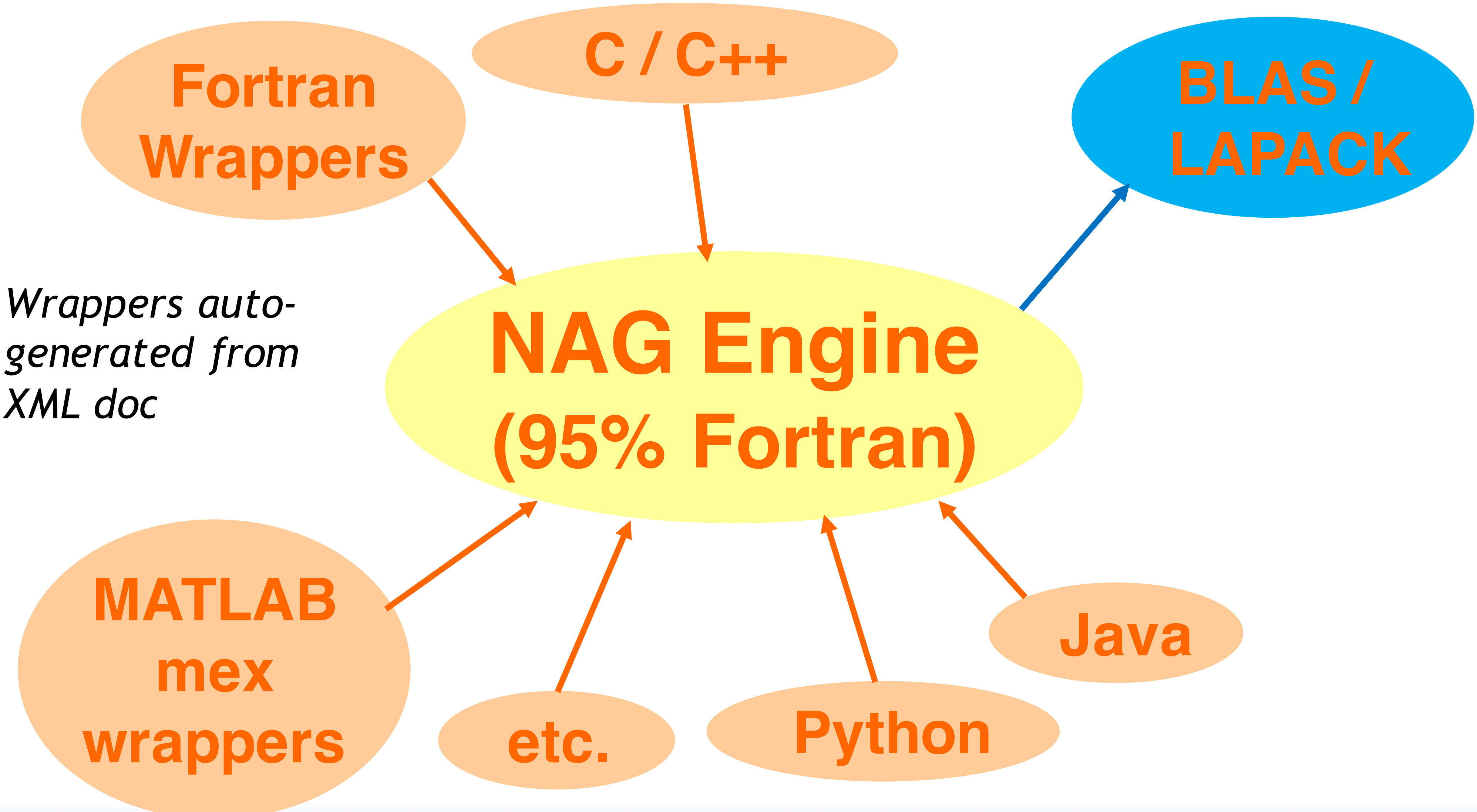
- Root Finding
- Summation of Series
- Quadrature
- Ordinary Differential Equations
- Partial Differential Equations
- Numerical Differentiation
- Integral Equations
- Mesh Generation
- Interpolation
- Curve and Surface Fitting
- Optimization
- Dense Linear Algebra
  - BLAS, LAPACK
- Sparse Linear Algebra
- Correlation and Regression Analysis
- Analysis of Variance
- Random Number Generators
- Univariate Estimation
- Nonparametric Statistics
- Smoothing in Statistics
- Contingency Table Analysis
- Survival Analysis
- Time Series Analysis
- Operations Research
- Special Functions

# NAG and the BLAS

- NAG Library is relatively high level, e.g.
  - mathematical optimization solvers
  - data fitting
- But relies heavily on lower levels like linear algebra



# NAG Library Wraps NAG Engine




*Wrappers auto-generated from XML doc*

# Test programs

- NAG library currently has about 1600 routines
- Every routine has associated detailed documentation and example program
- Also a "stringent test program"
  - For LAPACK routines these are independent of the netlib test programs
    - Ours are simpler and (hopefully) easier to extend
  - But we do also use the netlib ones

# NAG and the BLAS

## An Extended Set of FORTRAN Basic Linear Algebra Subprograms

JACK J. DONGARRA
Argonne National Laboratory
JEREMY DU CROZ and SVEN HAMMARLING
Numerical Algorithms Group, Ltd.
and
RICHARD J. HANSON
Sandia National Laboratory

---

## A Set of Level 3 Basic Linear Algebra Subprograms

JACK J. DONGARRA
University of Tennessee and Oak Ridge National Laboratory
JEREMY DU CROZ and SVEN HAMMARLING
Numerical Algorithms Group, Ltd.
and
IAIN DUFF
Harwell Laboratory

# Reproducibility

- SIMD instructions operate on several numbers at once

"vaddpd" instruction

| | | | | |
|---|---|---|---|---|
| a1 | a2 | a3 | a4 | register 1 |
| + | | | | |
| b1 | b2 | b3 | b4 | register 2 |
| = | | | | |
| a1+b1 | a2+b2 | a3+b3 | a4+b4 | register 3 |

- But to use these instructions memory alignment may be crucial …

# Example - dot product

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\underline{x}$ = | $x_1$ | $x_2$ | $x_3$ | | | | | | ··· | | | | | | | $x_n$ |
| $\underline{y}$ = | $y_1$ | $y_2$ | $y_3$ | | | | | | ··· | | | | | | | $y_n$ |

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

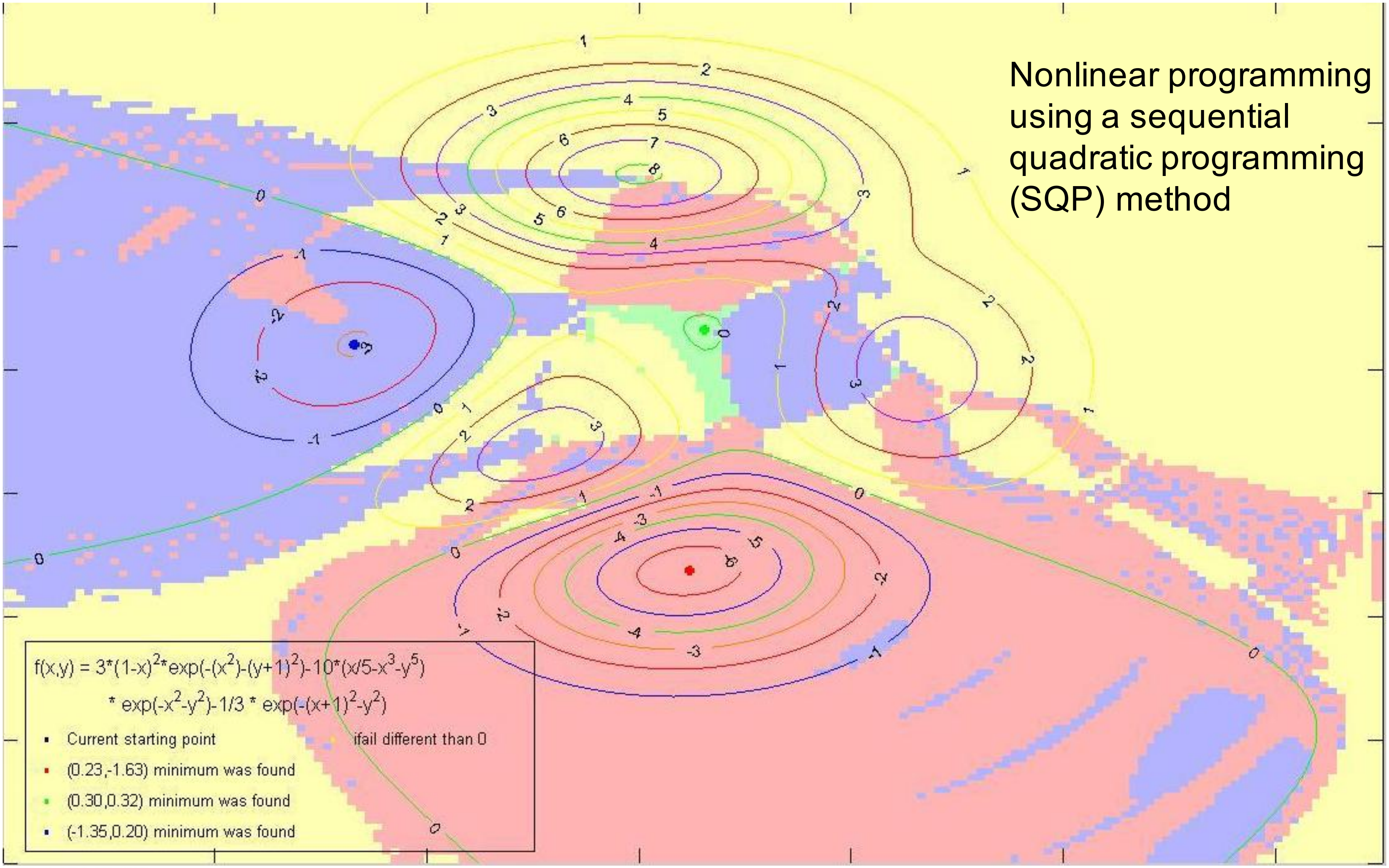
(if memory is not nicely aligned)

or $\tilde{p} = (x_1 y_1 + x_2 y_2 + x_3 y_3 + x_4 y_4) + (x_5 y_5 + x_6 y_6 + x_7 y_7 + x_8 y_8) + \cdots$

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

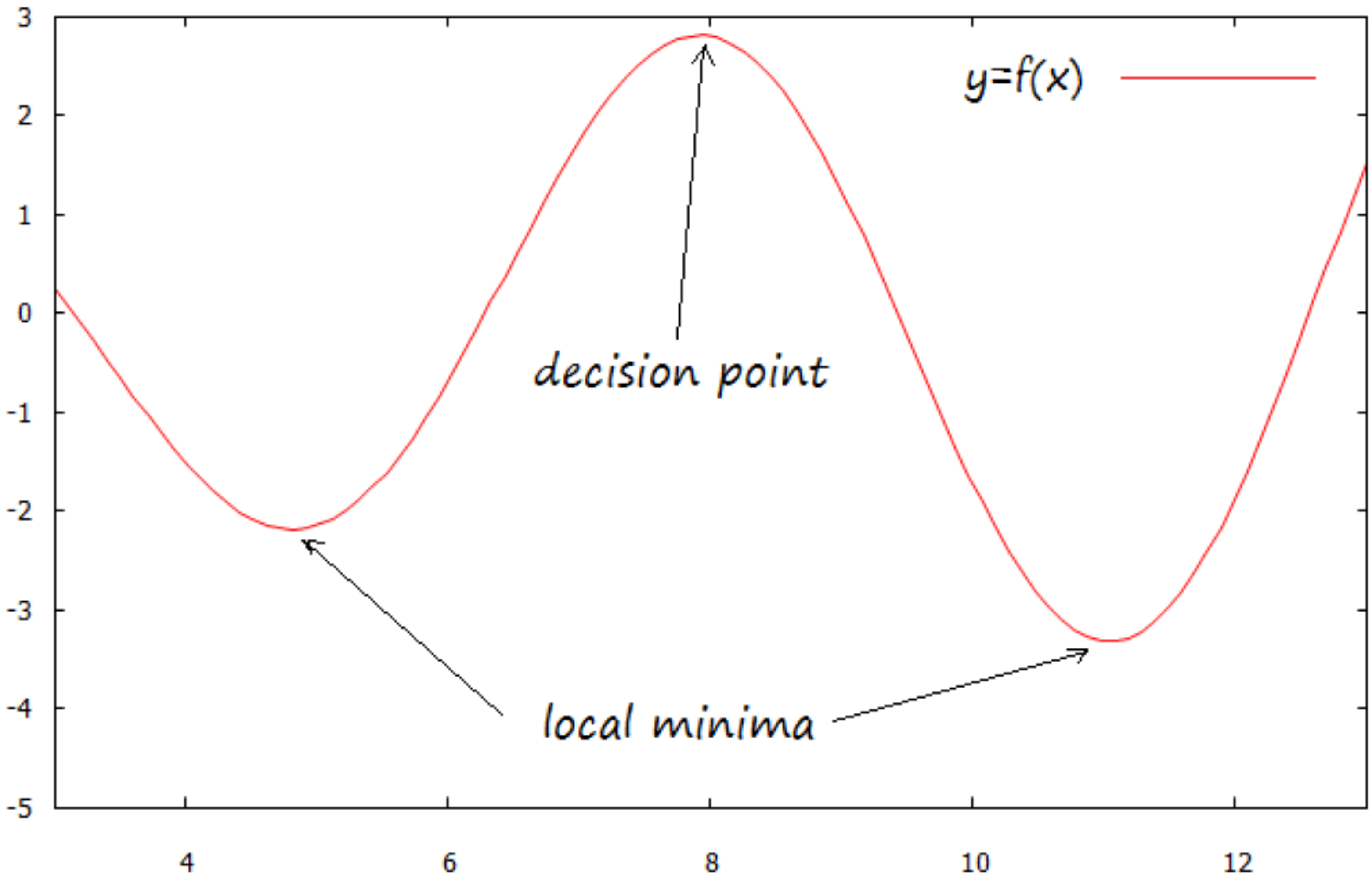
Mathematically equivalent – but the two results are not necessarily identical. Does it matter? Sometimes …

# Local (mathematical) optimization




Nonlinear programming using a sequential quadratic programming (SQP) method

# Local optimization




Without BWR – could go to one or other of the local minima on different runs of the *same program* on the *same machine*

# Importance of BWR

- Bit-Wise Reproducibility (BWR)
  - Essential for some users (e.g. in finance)
  - They may need to explain even minor differences
  - Sometimes no amount of discussion can change their minds!

- NAG users spot problems
  - e.g. with local optimization
  - e.g. with statistical analyses wrongly perceived by them as using stochastic algorithms

# Some things added in recent NAG versions

Usually added in response to customer demand

- Matrix functions
- Mixed integer nonlinear programming
- Change point analysis
- Travelling salesman problem

# Matrix Functions (work by *Higham et al*)

Given square matrix *A*, matrix function *f(A)* is a generalization of the scalar function f.

If A has a full set of eigenvectors V then it can be factorized as $A = VDV^{-1}$

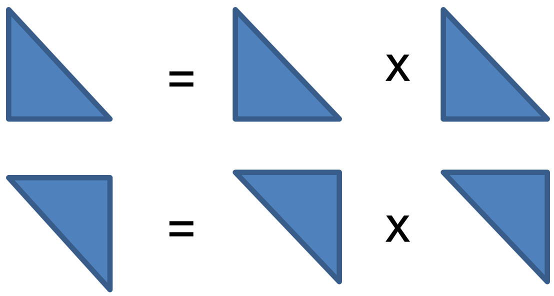
Then $f(A) = Vf(D)V^{-1}$

(It's more complicated if not a full set of eigenvectors)

Recent NAG libraries have *exp(A), log(A), sin(A), cos(A)*, general *f(A)*

# BLAS-like routines that would be useful to us

- Routines to multiply triangular matrices:




(xTRMM routines allow only one matrix to be triangular)

- Used a great deal in computing matrix functions
- Also useful in statistical applications

# BLAS-like routines that would be useful to us

- Multiplication of two symmetric matrices:

$$S_1 S_2$$

- Perhaps single-call pre- and post-multiplication:

$$X^T S X$$

where $X$ is general and $S$ is symmetric

Such operations also used a lot in statistical calculations

- e.g. time series analysis

# H02DA – Mixed Integer Nonlinear Programming

$$\mathit{minimize} \quad f(x, y)$$ (general nonlinear function)

$$\mathit{subject\ to} \quad c_j(x, y) = 0, j = 1,2, \dots m_e$$

$$c_j(x, y) \geq 0, j = m_e + 1, \dots, m$$

$x$ – continuous variables (i.e. any real number)

$y$ – binary (0/1) or integer variables (i.e. whole numbers)

# H02DA – Mixed Integer Nonlinear Programming

- H02DA is based on the algorithm MISQP
  - Exler, Lehmann, Schittkowski
- Uses a modified SQP algorithm
  - Solves a sequence of QP problems
- Some advantages:
  - Tends to use few function evaluations
    - Unlike genetic algorithms
  - Integer variables are not assumed *relaxable*
    - i.e. the objective function is only evaluated at integer points *y*

# H02DA – Mixed Integer Nonlinear Programming

Example from portfolio optimization:

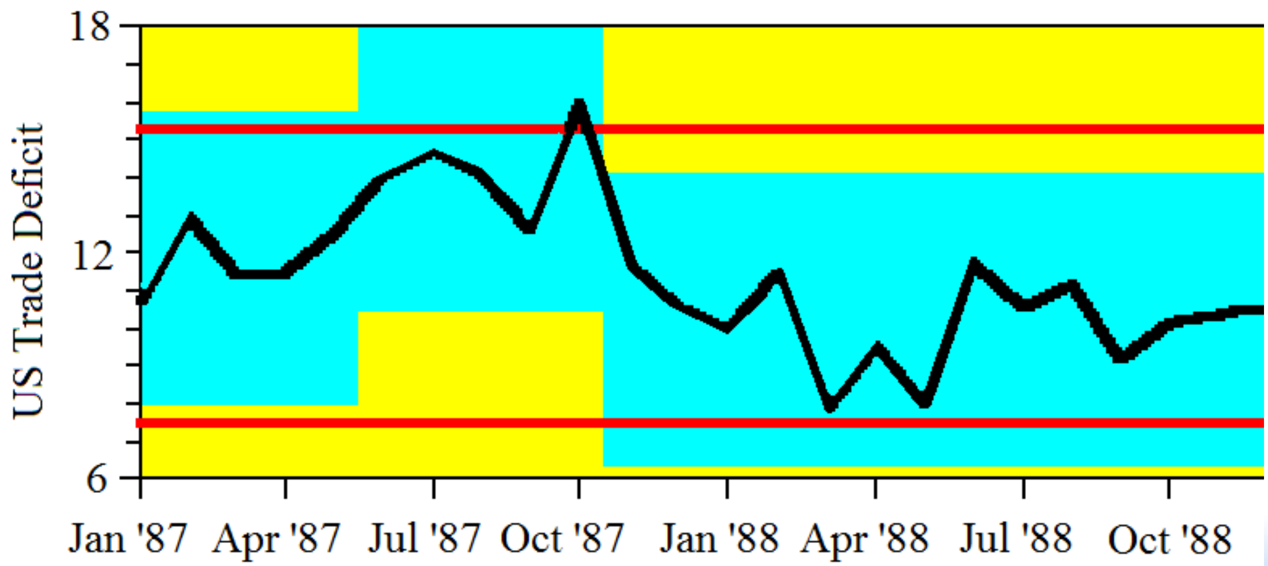
Choose $p$ assets from a set of $n$ assets such that the risk of holding the assets is minimized for a given expected return

Other examples:

- Network design for water or gas distribution
- Operational reloading of nuclear reactors

# G13N – Change Point Analysis

- G13N – Change Point Analysis
  - Detect points in a univariate time series where some feature of the data (e.g. the mean or standard deviation) changes
  - G13NA - PELT algorithm by R Killick, P Fearnhead and IA Eckely
    - PELT - Pruned Exact Linear Time
  - G13ND - Binary segmentation



# H03BB – Travelling Salesman Problem


Typical NAG salesman's path:

*Birmingham --> London --> Birmingham --> Oxford --> Birmingham --> London --> Birmingham --> New York --> Birmingham --> London --> Birmingham --> Oxford --> Birmingham --> Chicago --> Birmingham --> London --> Birmingham --> Oxford --> Birmingham --> Mumbai -->...*

Total distance travelled : endless

# H03BB – Travelling Salesman Problem

Given a set of places and the distances between them, e.g.

| Oxford | Birmingham | Warwick | Singapore | New York | Chicago | Mumbai | London | |
|---|---|---|---|---|---|---|---|---|
| | 78 | 45 | 6785 | 3397 | 3892 | 4523 | 60 | : Oxford |
| | | 26 | 6778 | 3380 | 3849 | 4545 | 101 | : Birmingham |
| | | | 6783 | 3383 | 3867 | 4531 | 82 | : Warwick |
| | | | | 9519 | 9359 | 2422 | 6733 | : Singapore |
| | | | | | 711 | 7789 | 3459 | : New York |
| | | | | | | 8047 | 3947 | : Chicago |
| | | | | | | | 4468 | : Mumbai |
| | | | | | | | | : London |

calculate something near the shortest closed path that connects each place

# H03BB – Travelling Salesman Problem

- Greedy algorithm – always go to nearest unvisited place – usually gets within 25% of best solution (but could be the worst solution)
- H02BB instead uses a *simulated annealing* approach
  - new states tested by switching pairs of places in a path
- Optimal path for previous slide computed by H03BB:

*Oxford --> London --> Mumbai --> Singapore --> Chicago --> New York --> Birmingham --> Warwick --> Oxford*

Total distance travelled : 20471 miles

Saving on typical NAG salesman expenses : large

# POP - Performance Optimisation and Productivity

- NAG is partner in the EU-funded POP Centre of Excellence
- Performance optimisation advice for parallel codes:
  - Suggestions/support on how to refactor code in the most productive way
  - Enable improved efficiency, better utilisation of HPC resources
  - Supporting wide range of languages (C, C++, Fortran, OpenMP, MPI, Cuda, …)
- Free of charge to organisations in the EU
- We hope that batched BLAS can play a part
  - Education is key
- Website: https://pop-coe.eu

Performance Optimisation and Productivity