The NAG Optimization Roadmap – Problems We'd Like to Solve

Mick Pont, NAG Oxford

mick@nag.co.uk

nag®

Experts in numerical algorithms and HPC services

Agenda

- Intro to NAG
- Current optimization in the NAG libraries
- Things we'd like to do
- Challenges and opportunities of modern hardware

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-F08 - Dense Linear Algebra
- F11-F12 – Sparse 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
- G12 - Survival Analysis
- G13 - Time Series Analysis
- H - Operations Research
- S - Special Functions

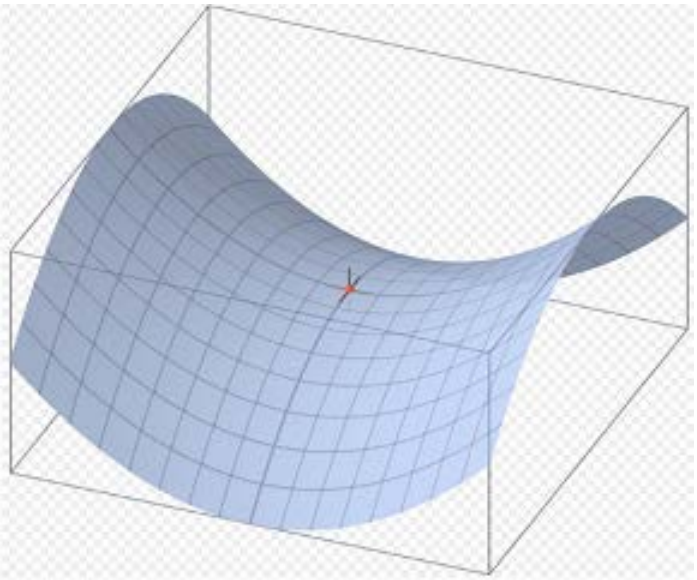
Current NAG Optimization includes

- Modified Newton methods
  - Use the Hessian, or a finite difference approximation, to define search direction
- Quasi Newton methods
  - approximate the Hessian by a matrix $B^{(k)}$ which is modified at each iteration to include information obtained about the curvature of F along the current search direction $p^{(k)}$.
- Conjugate gradient methods
  - No need to store order n matrix
  - Good for large problems
- SQP (sequential quadratic programming) methods for NLP

Recognising a solution (unconstrained problems)

- At a solution point:
  - the gradient of the objective function is zero
  - the Hessian matrix $G(\underline{x})$ of second partial derivatives of the objective function classifies the point:
    - positive definite: local minimum
    - negative definite: local maximum
    - indefinite: a saddle point

$$G(\underline{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$



Saddle point

Problem Types

- Linear programming: objective and constraints are all linear functions of x
  - E04MF uses a two-phase method:
    - Minimize sum of infeasibilities
    - Then minimize the objective while remaining feasible

- Quadratic programming:

  Minimize $f(x) = c^T x + \frac{1}{2} x^T H x$

  subject to $l \leq x \leq u, \quad l \leq Ax \leq u$

  - Hessian matrix H must be symmetric positive semi-definite – not stored explicitly
  - NAG routines can handle dense, sparse or non-convex problems

Least squares problems

Linear:

- e.g. Minimize $f(x) = \frac{1}{2}\|b - Ax\|^2$

  See e.g. E04NC

Nonlinear:

- Minimize $f^T f$ where f is a vector of m functions
  - Subject to nonlinear constraints
  - See e.g. E04US

Workhorse routines for general NLP

- E04UC
  - Derived from SNOPT (Gill, Murray and Saunders)
  - Sequential Quadratic Programming (SQP) method
  - NAG codes currently based on SNOPT Version 7.x
- E04UF
  - Reverse communication version of above
    - (i.e. no user-supplied function)

Current NAG optimization classification

- Number of variables:
  - Single variable $f(x)$
  - Multiple variable $f(x_1, x_2, \ldots, x_n)$
- Type of objective:
  - Linear
  - Quadratic
  - sum of squares
  - Nonlinear
- Constraints:
  - None
  - Simple bounds on variables $lb \leq x \leq ub$
  - Linear
  - Nonlinear

New optimization at next NAG release

- PENNON – penalty method for nonlinear and semidefinite programming (NLP and SDP)

Minimize	$f(x)$
Subject to	$g_i(x) \geq 0$
	$\mathcal{A}(x) \succcurlyeq 0$

where $\boldsymbol{\mathcal{A}}(x)$ is a matrix operator $\mathbb{R}^n \longrightarrow \mathbb{S}^m$
e.g. $\boldsymbol{\mathcal{A}}(x) = A_0 + \Sigma x_i A_i$
$\mathbb{S}^m$ is a symmetric matrix of order m
$A \succcurlyeq 0$ means A positive semidefinite

PENNON internally uses HSL routine MA97 for sparse factorization

PENNON in next release of NAG Library (CL24)

Semidefinite programming useful for:

- Image recognition:
  - Weinberger and Saul (2004) - Unsupervised Learning of Image Manifolds by Semidefinite Programming
- Approximation, Graph Theory, Quantum Computing:
  - Gartner and Matousek (2006) - Approximation Algorithms and Semidefinite Programming
- Nearest correlation matrix (alternative to current methods - see next slide)

- Current work also involves extending PENNON for efficient handling of Second Order Cone Programming problems
  - (Alain Zemkoho with M. Kocvara / U. Birmingham)

$$\begin{aligned} \text{Minimize} \quad & f^T x \\ \text{Subject to} \quad & \|A_i x + b_i\|_2 \leq c_i^T x + d_i, \quad i = 1,2,\dots,m \\ & Fx = g \end{aligned}$$

Nearest correlation matrix

Input matrix: $A = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix}$

NCM (NAG routine G02AA - *Qi, Sun, Borsdorf, Higham*):

$$\begin{bmatrix} 1.0000 & -0.8084 & 0.1916 & 0.1068 \\ -0.8084 & 1.0000 & -0.6562 & 0.1916 \\ 0.1916 & -0.6562 & 1.0000 & -0.8084 \\ 0.1068 & 0.1916 & -0.8084 & 1.0000 \end{bmatrix}$$

NCM via PENNON with preserved structure:

$$\begin{bmatrix} 1.0000 & -0.6823 & & \\ -0.6823 & 1.0000 & -0.5344 & \\ & -0.5344 & 1.0000 & -0.6823 \\ & & -0.6823 & 1.0000 \end{bmatrix}$$

Is preserved structure useful?

N.B. probably need to ensure that the NCM is strictly positive semidefinite (no negative eigenvalues no matter how tiny!). Both methods let you do this.

Other things new in CL24 release

- Global optimization using multi-start variants of local optimizers:
  - General constrained NLP
  - Nonlinear least squares
- Travelling salesman problem
  - Implemented via simulated annealing!

And things in our roadmap:

- Interior point method for linear programming (LP)
  - Using primal and dual Simplex formulations
- Interior point for nonlinear programming (NLP) - based on IPOPT
- Integer LP – based on Simplex LP
- Mixed Integer NLP – based on MISQP, K. Schittkowski

Optimization for Finance

Things that we want to be able to handle better

- Derivative pricing
- Portfolio optimization

Derivative Pricing

- *Derivative*
  - A contract between two parties that specifies conditions (e.g. dates, values) under which payments are to be made between the parties
  - Usually based on an underlying asset such as a stock
- Various formulae can be used to price a derivative
  - e.g. Black-Scholes-Merton, Heston
- Attempt to put a value on an option - the right to buy or sell an underlying stock at a pre-determined price on or before a certain date
  - NAG library currently has routines to evaluate closed form solutions
  - Can also evaluate "the Greeks" – sensitivities of prices to changes in relevant factors
    - e.g. underlying stock price, risk-free rate of interest
    - Prompted by concern over exposure to changes in these factors

Calibrating the model

- Quantitative analysts aim to determine parameters (such as volatility) of a model which make the pricing formula produce the same result as the current market price of a derivative
  - "Calibrating the model"
  - Plug parameters back into the formula to evaluate the Greeks
  - You might think this is a dubious thing to do!
- The process must be performed very quickly for lots of data
  - It is a constrained, weighted non-linear least-squares problem
  - Often a solution close to the previous solution is required but this is not embodied in the mathematical model – tricky for NAG solvers!
- N.B. reproducibility of results is important in finance
  - Transactions checked by separate computations – risk reduction
  - Different results can incur massive overheads in trading costs
  - Reproducibility partly depends on compilers

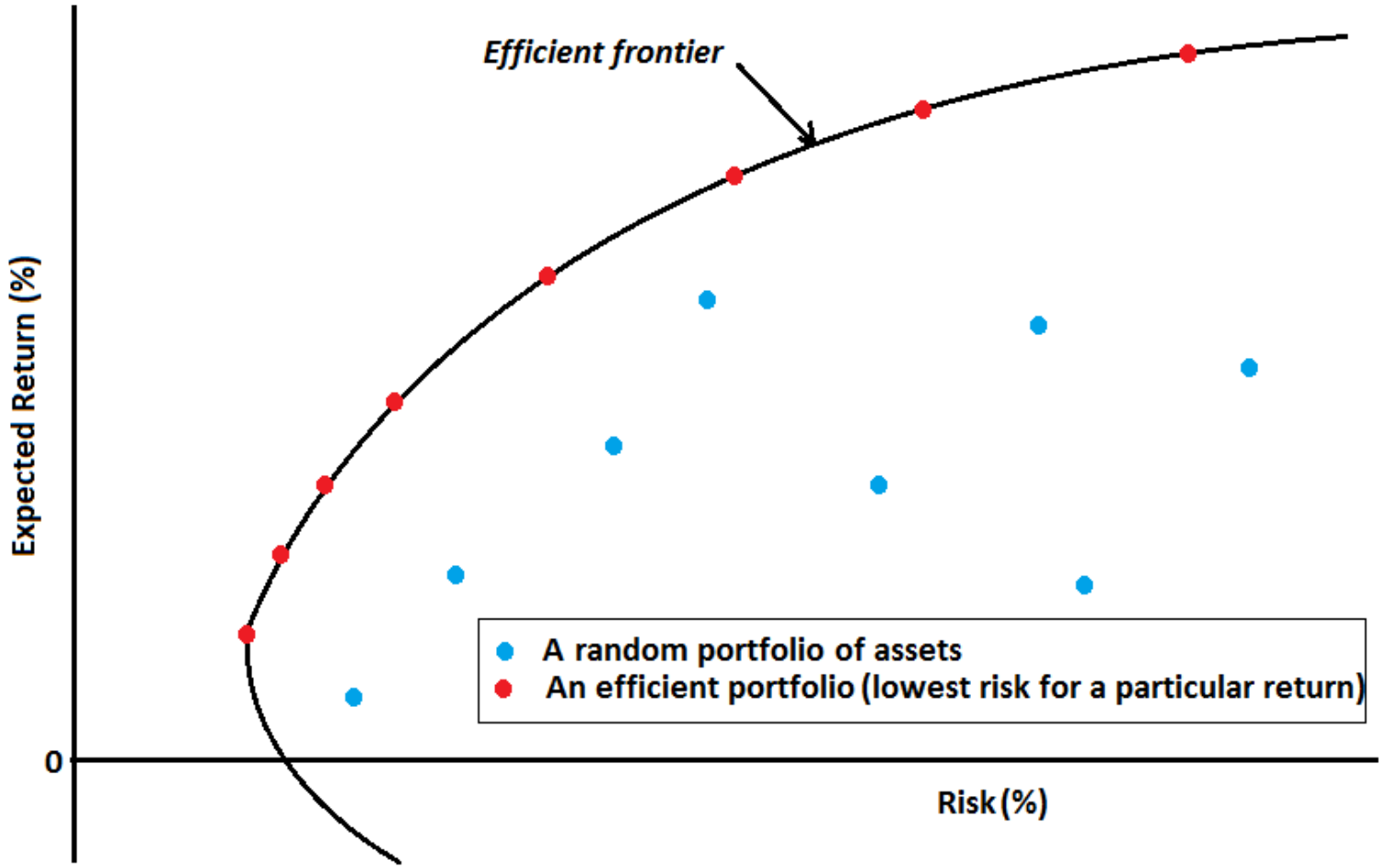
Portfolio Optimization

- *Portfolio* : a set of assets (e.g. stocks).
- Problem: given a choice of stocks available from a market, how might one choose which ones to own? May want to:
  - Minimize the amount of risk given a required investment return *or*
  - Maximize potential investment returns given a certain acceptable amount of risk
- H. Markowitz (1950s) first showed that overall risk can be reduced by constructing an appropriately weighted portfolio of stocks

Portfolio Optimization

- Simple Markowitz model requires:
  - μ_i : the expected returns associated with each of n assets
  - V : the covariance matrix associated with the assets (shows how much a change in value of one asset is mirrored by a change in each of the others)
  - l_i and u_i : lower and upper bounds on the amount of each asset we are willing (or able) to hold
- We wish to find:
  - x_i : the weights given to each asset (i.e. how much of each asset should be held). Likely constraint: $\Sigma x_i = 1$
- Classical Markowitz optimization problem (QP):
  - Minimize $x^T V x$ subject to $\mu^T x = r$ (i.e. minimize risk associated with desired return r)

The Famous Efficient Frontier



Markowitz Model

- Takes no account of *transaction costs*
  - Charges associated with buying or selling an asset
- A portfolio will need to be *rebalanced* from time to time
  - As more information becomes available over time, the optimal holding will change
  - The more often, the more the transaction costs build up
- More complicated formulae are used nowadays
  - They are nonlinear, and not always quadratic
  - They take into account transaction costs

Problems with modern models

- Models do not generally involve *smooth* functions
  - Transaction costs may vary in discrete jumps
  - Need something robust in face of piecewise discontinuities
- Assets usually bought in discrete parcels
  - Not really continuous variables
  - Likely to be *integer* variables with *large* values
- *Small* amounts of an asset may not be worth trading
  - Often have a minimum asset constraint $x_i \geq a$ where any amount less than a is not worth holding

Short and *Long* Positions

- Long position : assets bought and held normally
- Short position : assets sold before being owned
  - Seller *borrows* them, or hopes to buy them at a lower price before having to sell them
  - Not all asset holders are allowed to do this
- For portfolio optimization, allowing short as well as long holdings can be a big problem
  - Twice as many variables – some for short and some for long – call them sh and lh
  - Obviously not good to hold long and short in same asset so add a *complementarity constraint*: $sh.lh = 0$

Other Constraints

- May have a limit on the total number of assets held
  - e.g. an index tracking fund might choose from only 30 stocks
- May be simple bounds on holding of assets
  - $lb_i \leq x_i \leq ub_i$
- May be linear constraints
  - On asset sector, or geographical location of assets, etc.
- Problem sizes – number of stocks to choose from
  - Relatively small in UK for example
  - Much larger in USA – could be considering 5000 stocks
- Many constraints could be modelled using integer variables – but problem is then very difficult to solve
  - In practice, short cuts are used

All these features make these difficult problems - our efforts to solve with existing software have not been fruitful!

Other things we are asked for

- Out-of-core optimizer for QP
  - Where the problem to be solved is very large
  - But e.g. the Hessian and constraint matrices are not sparse
  - So will not fit in available memory

Challenges/Opportunities of modern hardware

- In the past – only had single core processors to worry about
  - They got faster every year – and so did our code!
- Now we have to consider
  - Single core performance is getting slower
  - Multiple cores and many cores are the norm
  - Floating-point parallelism in the instruction set is getting wider
    - SSE -> AVX -> …?
- GPUs, Intel Phi and “fusion”-type chips
  - Programming is certainly not getting easier
  - Chance of getting non-reproducible results increases
- If we do nothing our programs run slower
- But massive parallelism is available - how can we take advantage of it?

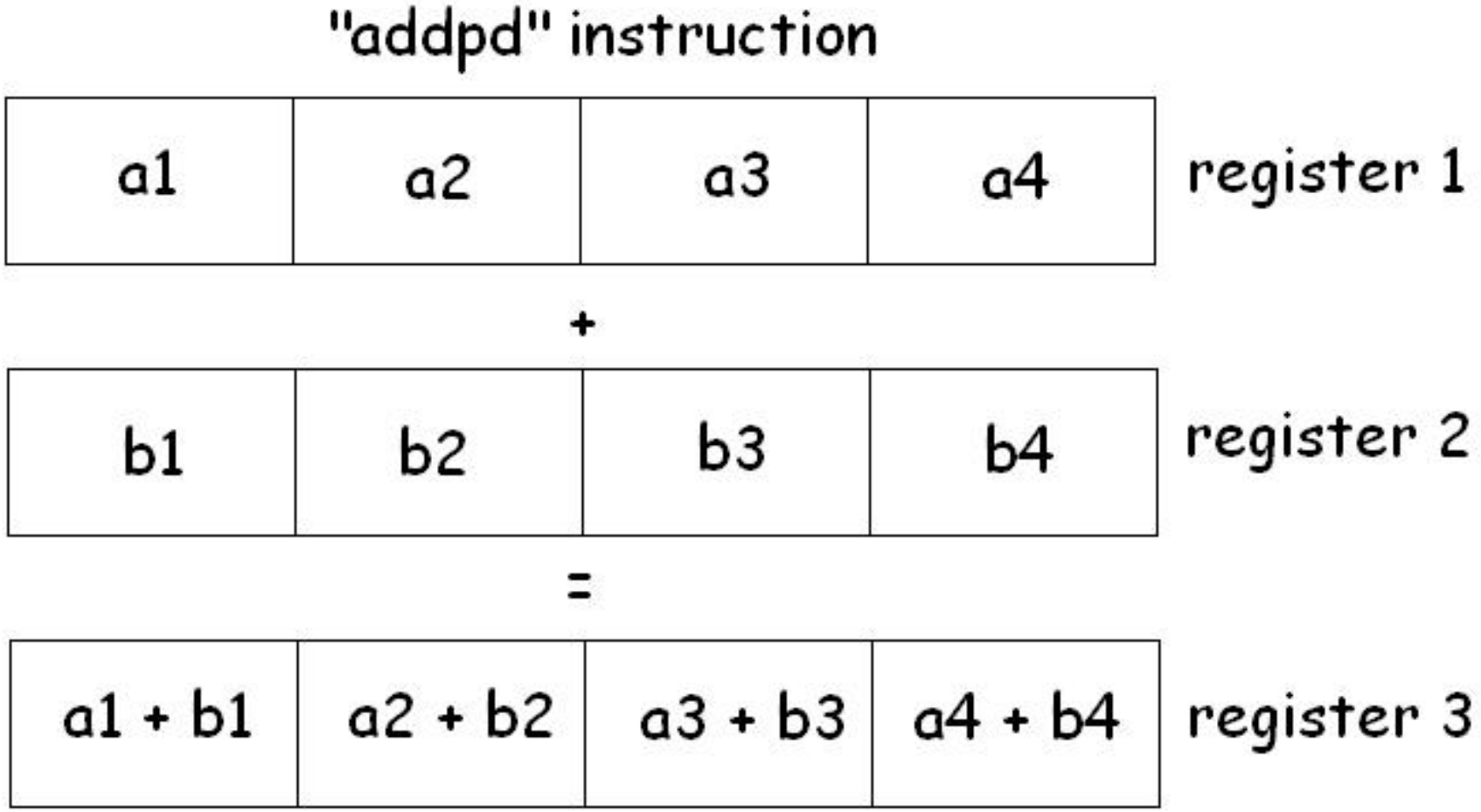
Reproducibility of results

- NAG users often ask for reproducible results across machines
  - But computers have finite precision
  - IEEE standard for floating-point arithmetic helps, but …
  - Chip design is making things more complicated
    - AMD vs Intel
  - 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

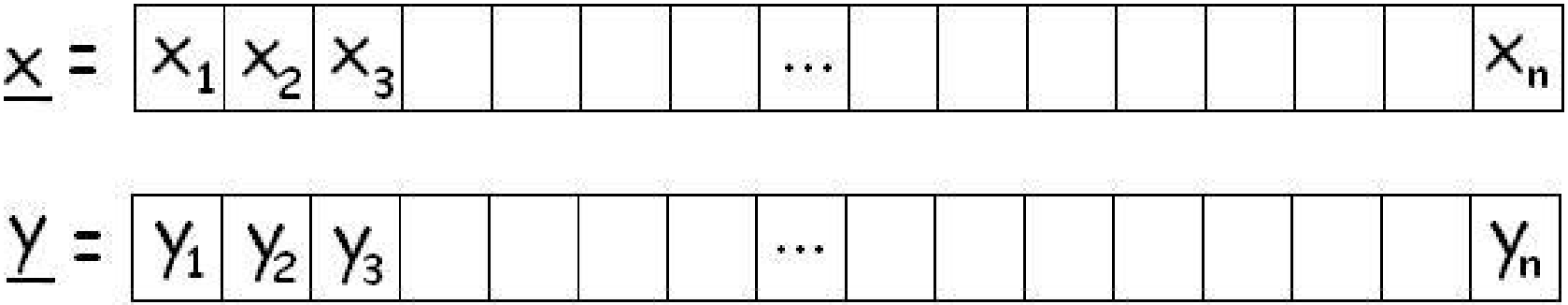
- Vectorized instructions operate on several numbers at once
- Clever compilers can take advantage of them
  - this is one of the few ways that individual processors can get faster now
- Can't or won't use them?
  - you'll not get anywhere near peak performance from your hardware

SSE / AVX



But to use these instructions memory alignment is crucial …

Bitwise reproducibility example - dot product of two vectors



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

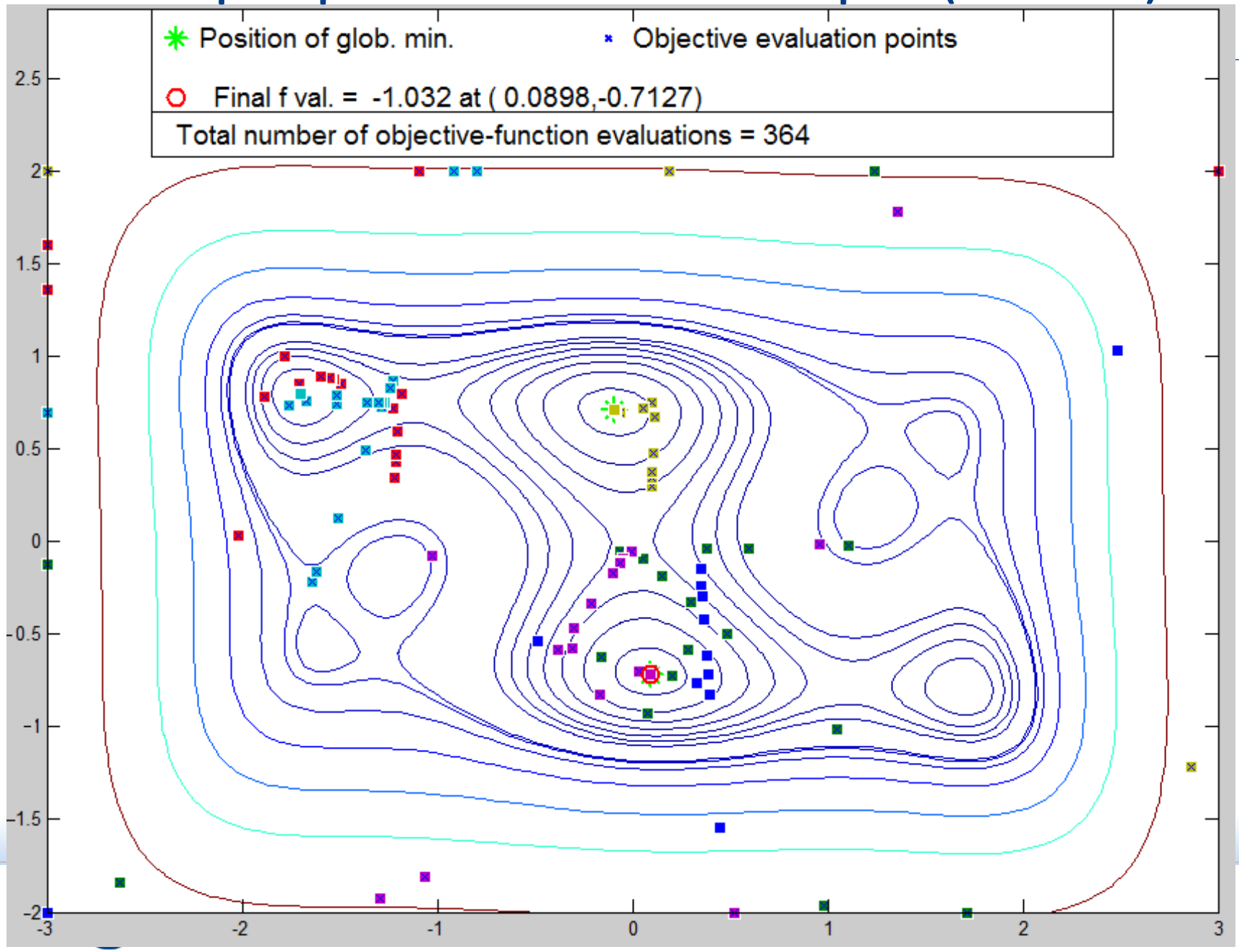
(if memory is not nicely aligned)

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!

Simple parallelism – example (E05UC)



Can we take advantage of massive parallelism?

- What if we could evaluate the objective function a million times in parallel?
  - Would completely new algorithms become available?
  - Would exact derivative information be less important?
    - Sometimes the function to be minimized is not differentiable anyway
  - Would reproducibility become even harder?

Summary

- Various things on the NAG roadmap
  - These we mostly know how to do – just need time and effort
- Some new things we don't know how to do well
  - So not on roadmap yet
  - But there is high demand for them – e.g. portfolio opt.
- Modern hardware presents challenges for all software, including for optimization
  - Can take advantage of parallel linear algebra
  - But we also need to find ways to take advantage of new hardware …
  - … while trying to avoid portability and maintainability headaches