Sparsity: the key to tackling large-scale least-squares problems

Jennifer Scott

STFC Rutherford Appleton Laboratory and
The University of Reading

jennifer.scott@reading.ac.uk

ICMS, Edinburgh, 14 January 2025

My background: PhD days

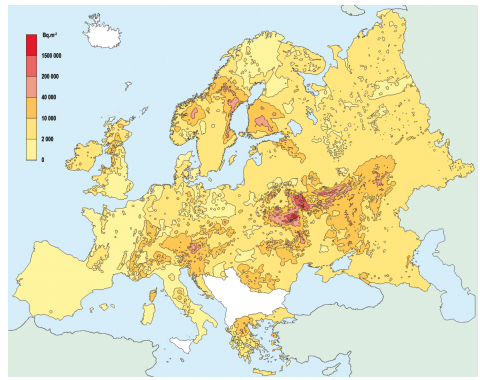
1981–1984 DPhil from Oxford University

Worked on Volterra integral equations.

- Lot of theory, limited computing but did learn some Fortran and use of mainframe computers.
  Note: first MATLAB was 1984 and Python 1991.
- Results on Gronwall integral inequalities used in proving convergence of numerical schemes still cited.
- Not all theoretical: later co-authored a paper on an application (work used by Unilever in the design of a pregnancy testing kit).

My background: first job

- 1986: After a Junior Research Fellowship (post doc) in Oxford, decided I wanted to apply my maths to real applications that involved research and computing.
- Took a position in the Theoretical Physics Department at the National Radiological Protection Board.
  This was two weeks before Chernobyl nuclear power plant disaster.



- Worked on modelling of the human eye (effects of exposure to infrared radiation).
- Learnt lots of new things; wrote two papers that continue to be cited.

My background: career progression

1986: Won a prestigious
Leslie Fox Prize in Numerical Analysis

- After 18 months at NRPB, head-hunted to join the Numerical Analysis Group at the Harwell Laboratory (next door to NRPB).
- 1990: Group moved to Rutherford Appleton Laboratory (result of Thatcher politics).

Most of my career spent at the Rutherford Appleton Lab.

Part of Science and Technology Facilities Council (STFC).
The site hosts some of the UK's major scientific facilities, including:

- ISIS Neutron and Muon Source
- Central Laser Facility
- Diamond Light Source synchrotron

Advert: SIAM UKIE Annual Meeting will be at RAL on 11 April 2025.

My background: move to university

- Part time working 1988–2007 (combining work and family).
- 2016: offered a professorship in mathematics at the University of Reading and Director at Reading of the EPSRC CDT in *Mathematics of Planet Earth* (took over from Beatrice Pelloni).
- Split my time: 2 days at RAL, 3 days at Reading.
- 2024: New EPSRC CDT in *Mathematics for our Future Climate* (with Imperial College and the University of Southampton). 90 PhD students over 5 cohorts.

The common research thread throughout my career has been

LARGE sparse matrices

In this talk, I aim to give a high level overview. I am aiming at those who know nothing in detail about this field but hope that those who know something will also see something that's new and exciting to them.

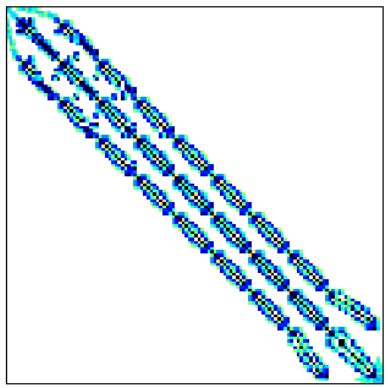
What's a sparse matrix?

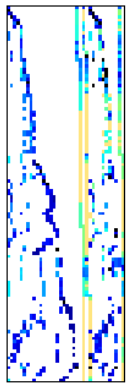
Essentially, it's a matrix with a "small" number of non zero entries. That is, a matrix that benefits from using special techniques to take advantage of the large number of zero entries.

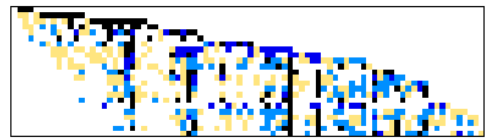
The term was introduced in 1950s by Harry Markowitz when describing his work on portfolio theory that won the 1990 Nobel Prize for Economics.

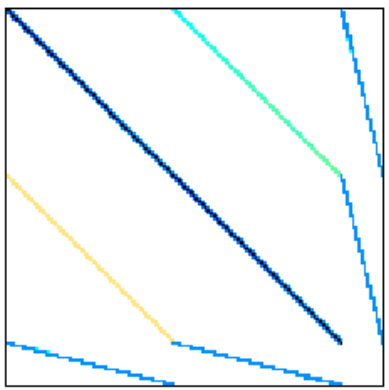
Note: any matrix not treated as sparse, is regarded as dense (even if it contains zeros).

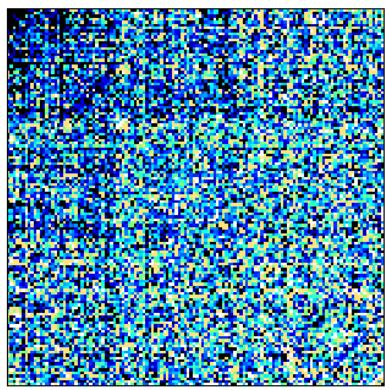
So what does a sparse matrix look like?

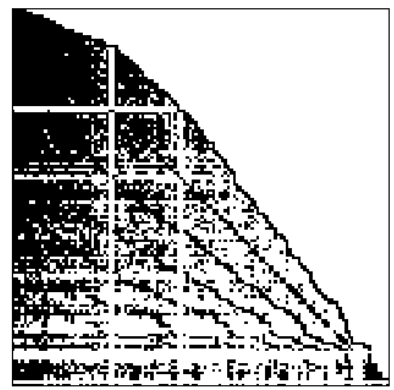












Where do sparse matrices arise?

The short answer: everywhere!

Oceanography	Oceans circulation, ...
Meteorology	Climate change, weather forecasting, ...
Structural mechanics	Elasticity, Plasticity, ...
Chemical engineering	Large molecules simulation, ...
Hydrology	Underground water remediation, pollution, ...
Fluid-dynamics	Stokes, Navier-Stokes, advection diffusion, ...
Economics	Econometric, linear programming, ...
Physics	Hamiltonian problems, thermodynamics, ...
Biosciences	Ecology, diseases control,...
Computer science	Data mining, Googling, networks, ...
...	

All give rise to matrices with different patterns and characteristics.

How much memory do we need for a dense matrix?

2 GB of RAM holds a dense matrix of size $16,000 \times 16,000$.

A dense matrix of size $10^5 \times 10^5$ needs 75 GB of RAM.

A typical modern laptop may have up to 64 GB. Huge compared to 1970s or even 2000s but not enough!

Important: The operations needed to solve a dense linear system $Ax = b$ increase with the cube of the problem size ($\mathcal{O}(n^3)$).

There is a demand to solve modest sized systems VERY FAST and to solve EVER LARGER systems.

We MUST exploit sparsity.

Sparsity is challenging

- We have to be able to store and manipulate only the non zero entries.
- We have to keep track of the non zero entries during our computation (some entries that are initially zero will become non zero so how do our data structures accommodate this?)
- Naively applying the same methods as we use for dense problems leads to a loss of sparsity and hence computation becomes impractical.

Thus exploiting sparsity involves

- Special data structures (and different structures are needed for different tasks).
- Algorithms that account for the sparsity.
- Theory that supports the algorithms (although often we have to rely on heuristics).
- Sophisticated implementations.

For the rest of this talk, we take a brief look at using sparse matrix techniques to solve large-scale least squares problems

$$\min_x \|b - Ax\|_2$$

where $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$ are given.

We seek to (numerically) compute $x \in \mathbb{R}^n$.

Our focus is on overdetermined systems ($m > n$) ("skyscraper" A).

Note: nonlinear LS problems are usually solved by an iterative procedure. At each iteration the system is approximated by a linear one, and thus the core calculation is similar in both cases.

- The least-squares method was first published by Legendre 1805.
- It is also credited to Gauss 1809, who made significant contributions to the mathematical formulation and theoretical understanding.
- Gauss successfully used LS method to approximate the orbit of the newly-discovered asteroid Ceres from the few observations that had been made of it before.

Today, LS remains a widely-used and essential tool.

It is the simplest and most commonly applied form of linear regression, which in turn is the most straightforward machine learning algorithm.

Diverse modern applications include:

- Medicine e.g., impact of environmental factors on human health.
- Finance e.g., quantify the relationship between variables such as a share price and earnings per share.
- Marketing e.g., model the relationship between advertising spending and sales.
- Image processing e.g., enhance, restore or compress images.
- Climate change e.g., study the relationship between greenhouse gas emissions and global temperature.
- Sports science e.g., predict performance of athletes by using variables such as scores, statistics, rankings and ratings etc.

The more data we have, the LARGER the problem.

A is sparse as each quantity has limited connections to other quantities.

Let's assume for simplicity that A is of full rank n.

x is the unique LS solution if and only if it satisfies the normal equations

$$Cx = A^T b, \quad C = A^T A \in \mathbb{R}^{n \times n},$$

Alternatively, if $r = b - Ax$, solve the augmented system

$$K \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \quad \text{with} \quad K = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)},$$

Which to choose? Any thoughts on pros and cons?

Normal equations:

$$Cx = A^T b, \quad C = A^T A,$$

If A has full column rank, C is symmetric positive definite ☺

Exploit existing solvers for SPD systems.

Solve using sparse Cholesky factorization and cheap triangular solves

$$C = LL^T, \quad Ly = A^T b, \quad L^T x = y.$$

Sparse case problem: the L factor fills in (adds to memory and work) ☹

Symmetrically reorder C to keep L sparse

$$PCP^T = LL^T, \quad Ly = PA^T b, \quad L^T w = y, \quad x = P^T w.$$

Computing permutation P is a separate research topic.

Algorithms employ close association between sparse matrices and graphs.

- **Local strategy**: seek at each stage to order next the row/column that leads to limit fill in the corresponding column of L (minimum degree algorithm)
- **Global strategy**: recursively subdivides C (nested dissection)

Typically, methods start with a global strategy and switch to a local strategy when subproblem sufficiently small.

Most software offers the user options

no single approach is the best for all problems

Potential issues with the normal equations: ☹

- Possible loss of information in explicitly computing C and $A^T b$.
- $\kappa(C) = \kappa(A)^2$, where $\kappa = \sigma_{max}/\sigma_{min}$ is condition number. Potential problem if A is ill conditioned (that is, $\kappa(A)$ is large, indicating solution is sensitive to small changes to input data).
- If A is (nearly) rank deficient, Cholesky factorization breaks down.
- QR factorization more stable but expensive for large n.

Another challenge: if A has a single "dense" row then C fills in.

Alternative: solve augmented system

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \quad r = b - Ax.$$

- This system is of order $n + m$ (recall $m \gg n$ is typical). ☹
- It is symmetric indefinite so Cholesky factorization not possible. ☹
- Compute $PCP^T = LDL^T$, with D block diagonal.
  P must preserve sparsity and numerical stability.

LOTS of research gone into solving such systems ☺
But remains challenging (numerical pivoting needed for stability, this complicates code, can significantly inhibit parallelism, ...)

So far, we have considered so-called sparse direct methods.

They are highly sophisticated and involve a matrix factorization.

Users like them because they can be employed largely as black box solvers (don't need to be an expert to use them).

But they are memory hungry and memory increases with the problem size.

So far, we have considered so-called sparse direct methods.

They are highly sophisticated and involve a matrix factorization.

Users like them because they can be employed largely as black box solvers (don't need to be an expert to use them).

But they are memory hungry and memory increases with the problem size.

Switch to using an iterative method

Iterative methods compute a sequence of approximate solutions $\{x^{(k)}\}$ that (hopefully) converge to the sought-after solution x in an acceptable number of iterations.

Topics of recent interest to me

- How to precondition LS problems for an iterative solver? Idea is to transform the problem to make it "nicer" to solve but easy to
- How to handle LS problems when A has a few "dense" rows?
- Can we save time/memory by using mixed precision arithmetic?
- What about huge problems where solutions are required in a short time frame?

Preconditioned iterative solvers

- For LS problems popular method is LSQR (1982) (other methods include CGLS, LSMR, ...).
- In exact arithmetic, LSQR is equivalent to applying conjugate gradient method to normal equations and is numerically more stable than CGLS.
- Rate of convergence depends on the singular values of A (poor convergence if $\kappa(A)$ is large, which it often is in real applications).

Preconditioned iterative solvers

Aim to improve convergence by preconditioning.

Solve

$$\min_{z \in \mathbb{R}^n} \|b - AM^{-1}z\|_2, \quad x = M^{-1}z.$$

Major challenge: how to find suitable M?

- Highly problem dependent
- M should be chosen so that
  - $\kappa_2(AM^{-1}) = \sigma_{max}(AM^{-1})/\sigma_{min}(AM^{-1})$ is small (clustered singular values) and less than $\kappa_2(A)$
  - products with M^{-1} and M^{-T} are cheap.

Methods I have worked on:

- Incomplete factorizations, particularly $C \approx \widetilde{L}\widetilde{L}^T$.
  Idea: "drop" entries so that $\widetilde{L}$ much sparser than Cholesky factor L. Choosing how to drop entries is the challenge. Scott and Tůma (2016).
- Algebraic domain decomposition preconditioners.
  Idea: exploit the similarity between sparsity structure of $C = A^T A$ and that of the weak formulation of PDEs. Al Daas, Jolivet, and Scott (2022).

What if A contains some dense rows?

A dense row has significantly more entries than the other rows e.g. there is a relationship between many of the variables.

$$A = \begin{bmatrix} A_s \\ A_d \end{bmatrix}, \; A_s \in \mathbb{R}^{m_s \times n}, \; A_d \in \mathbb{R}^{m_d \times n}, \; m_d \ll m_s$$

- The normal matrix $C = A_s^T A_s + A_d^T A_d$ is no longer sparse.
- A_s often rank-deficient (contains null columns).

Possible approaches:

- sparse stretching replaces dense row block by sparser but larger block;
- a block factorization method that processes the rows that are identified as dense separately within an iterative solver;
- a Schur complement approach that exploits the block structure within the augmented system formulation of the LS problem;
- updating strategies based on QR-based factorizations.

Normal equation approach

$$Cx = (C_s + A_d^T A_d)x = A_s^T c_s + A_d^T c_d.$$

Equivalent $(n + m_d) \times (n + m_d)$ blocked linear system

$$\begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix} \begin{pmatrix} z \\ A_d z \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}.$$

If $C_s = L_s L_s^T$ then signed Cholesky factorization

$$\begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix} = \begin{pmatrix} L_s & \\ B_d & L_d \end{pmatrix} \begin{pmatrix} I & \\ & -I \end{pmatrix} \begin{pmatrix} L_s^T & B_d^T \\ & L_d^T \end{pmatrix},$$

where B_d is solution of triangular system

$$L_s B_d^T = A_d^T$$

and L_d is Cholesky factor of $m_d \times m_d$ (negative) Schur complement

$$I + B_d B_d^T = L_d L_d^T.$$

If A_s is **rank deficient**, introduce shift $\alpha > 0$

Compute $C_s(\alpha) = A_s^T A_s + \alpha I = L_s L_s^T$.

Straightforward to show that

$$(C_s(\alpha) + A_d^T A_d)^{-1} = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} C_s(\alpha) & A_d^T \\ A_d & -I \end{pmatrix}^{-1} \begin{pmatrix} I \\ 0 \end{pmatrix}.$$

Use to obtain SPD preconditioner for use with e.g. LSQR.

Can generalise to incomplete factorizations of $C_s(\alpha)$.

Augmented system approach

$$\begin{pmatrix} I & 0 & A_s \\ 0 & I & A_d \\ A_s^T & A_d^T & 0 \end{pmatrix} \begin{pmatrix} r_s \\ r_d \\ x \end{pmatrix} = \begin{pmatrix} b_s \\ b_d \\ 0 \end{pmatrix},$$

where

$$r = \begin{pmatrix} r_s \\ r_d \end{pmatrix} = \begin{pmatrix} c_s \\ c_d \end{pmatrix} - \begin{pmatrix} A_s \\ A_d \end{pmatrix} x.$$

Eliminating r_s gives $(N + m_d) \times (N + m_d)$ system

$$K_r \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix}, \qquad K_r = \begin{pmatrix} -C_s & A_d^T \\ A_d & I \end{pmatrix}.$$

Sparse symmetric saddle-point systems ... challenging ...

Underlying my research: objective to translate it into software.

- Since 1990s, I have been a major contributor to the HSL mathematical software library
- HSL is the oldest mathematical software library in continual use (started 1963). Fully supported and maintained.
- It is now a specialised library, focusing on sparse matrix computations.
- It is used worldwide by academics (no charge) from a wide range of disciplines and commercial organisations.
- Commercial users in recent years have included Shell, Schlumberger, leading F1 teams, Bayer, NAG, GE Global, Exxon Mobil, Aspentech, ...

Incomplete factorization preconditioner HSL_MC35

LS sparse solver HSL_MA85

- Offers both normal system and augmented system approaches.
- Handles dense rows.
- Includes the a use of shift, combined with refinement.
- Incorporates the use of scaling.

Normal equations: uses HSL_MA87 for sparse Cholesky factorization.

Augmented system: uses HSL_MA97 for sparse LDLT factorization.

Current research interests: mixed precision

Can we reduce memory requirements and/or improve computational times and/or reduce energy consumption by using mixed precision?

Mixed precision: rather than doing everything in 64-bit (double precision) use lower precision (32-bit or even 16-bit arithmetic).

This is driven by growing availability of hardware integration of low precision special function units designed for machine learning applications.

Can numerical analysts exploit this new architecture?

Lots of interest in recent years. Reduction in memory usage, less data movement, energy saving, faster ... but what of the theory and what happens in practice?

What are some of the challenges?

	x_{min}	x_{max}
fp16	6.10×10^{-5}	6.55×10^{4}
fp32	1.18×10^{-38}	3.40×10^{38}
fp64	2.22×10^{-308}	1.80×10^{308}

- "Squeezing" a matrix in fp16 looses information.
- Computations can overflow. Code must be made "safe".
- Work to recover requested accuracy can outweigh gains of using lower precision.
- Currently limited compilers available and so prototyping only.

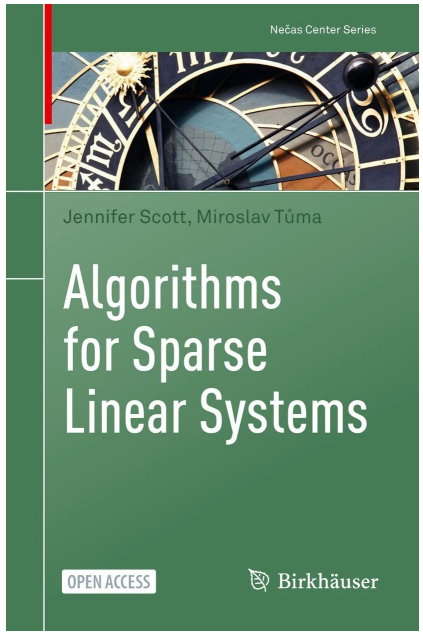
Current research interests: data assimilation in weather forecasting.

- Huge problems, time-limited
- Cannot afford to use a direct solver or to run an iterative method to convergence.
- Interest from practitioners is in what happens in a small number of iterations.
  - Can we get a useful approximate solution?
  - How can we improve the quality at minimal cost?
  - What preconditioner can we use?
  - Can randomized methods help?
  - What about using mixed precision/low precision?

Thank you for listening.

Questions or comments?

Please also talk to me this week!



Recent open access book by Scott and Tůma (2023).

Forthcoming (2025) article on solving large LS problems to be published in Acta Numerica.