C++ONLINE

ANDREW DRAKEFORD

TALK:

DATA ORIENTED DESIGN

THE MISSING STEP

2025

Leave One Out Regression- Performance Run

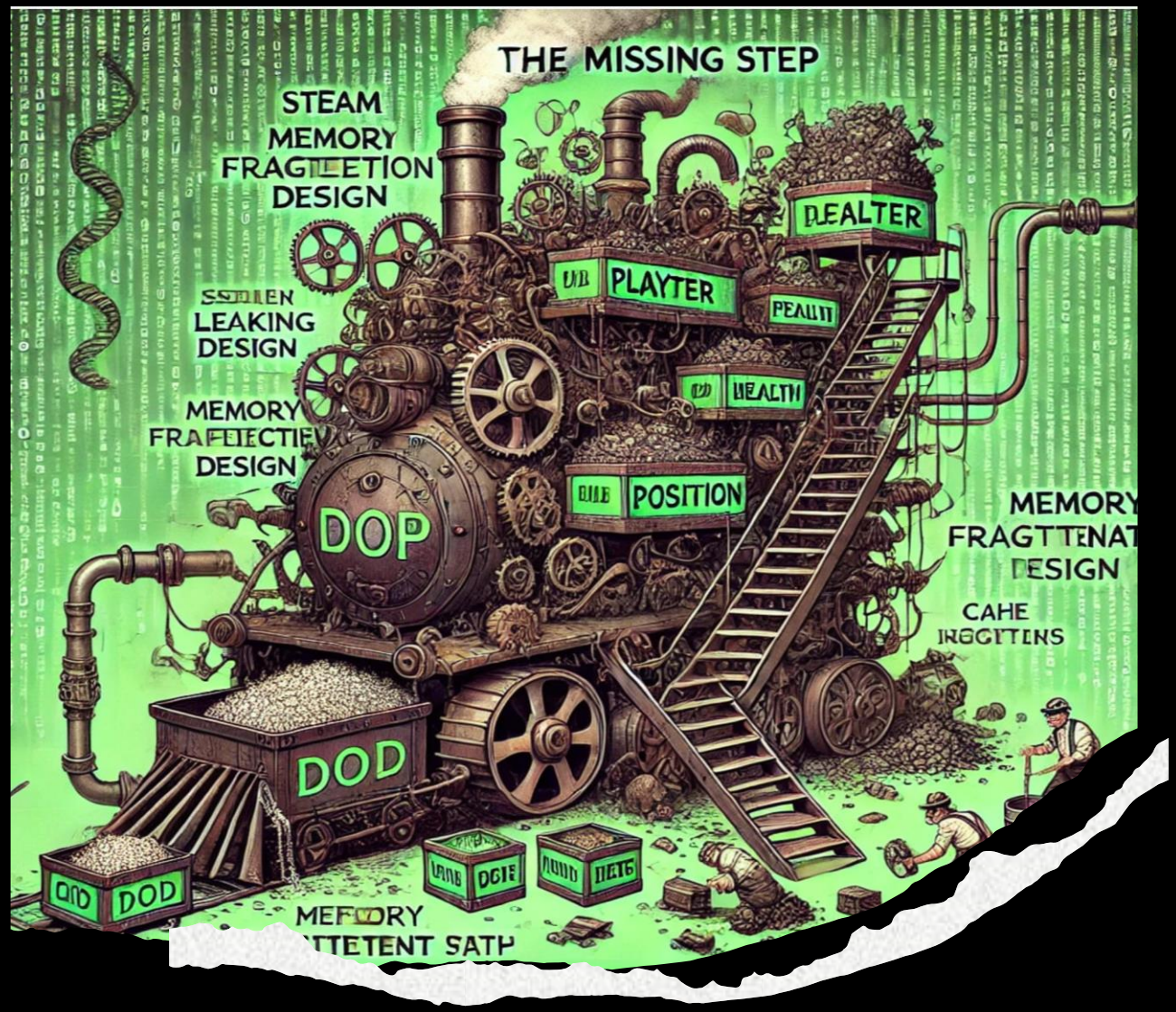
- One million elements, One million leave one out regressions:
- Reference implementation

```
generating data set size 1000000
setting data
fitting data
1.46681e+07 milli seconds per fit
D:\online25\DR3\x64\ICC2023\LeaveOneOutRegression.exe (process 79504) exited with code 0.
Press any key to close this window . . .
```

- New implementation

```
18.0961 milli seconds per fit
D:\online25\DR3\x64\ICC2023\LeaveOneOutRegression.exe (process 58192) exited with code 0.
Press any key to close this window . . .
```

Data Oriented Design – The missing Step



DOD (Mike Acton style)

- Programs just transform data from one form to another
- Understand the problem better by understanding the data better
- Different problems require different solutions
- Different data implies different solutions?
- Understand the cost to understand the problem
- Understand the hardware => understand cost => understand problem
- Code is data
- Everything is a data problem

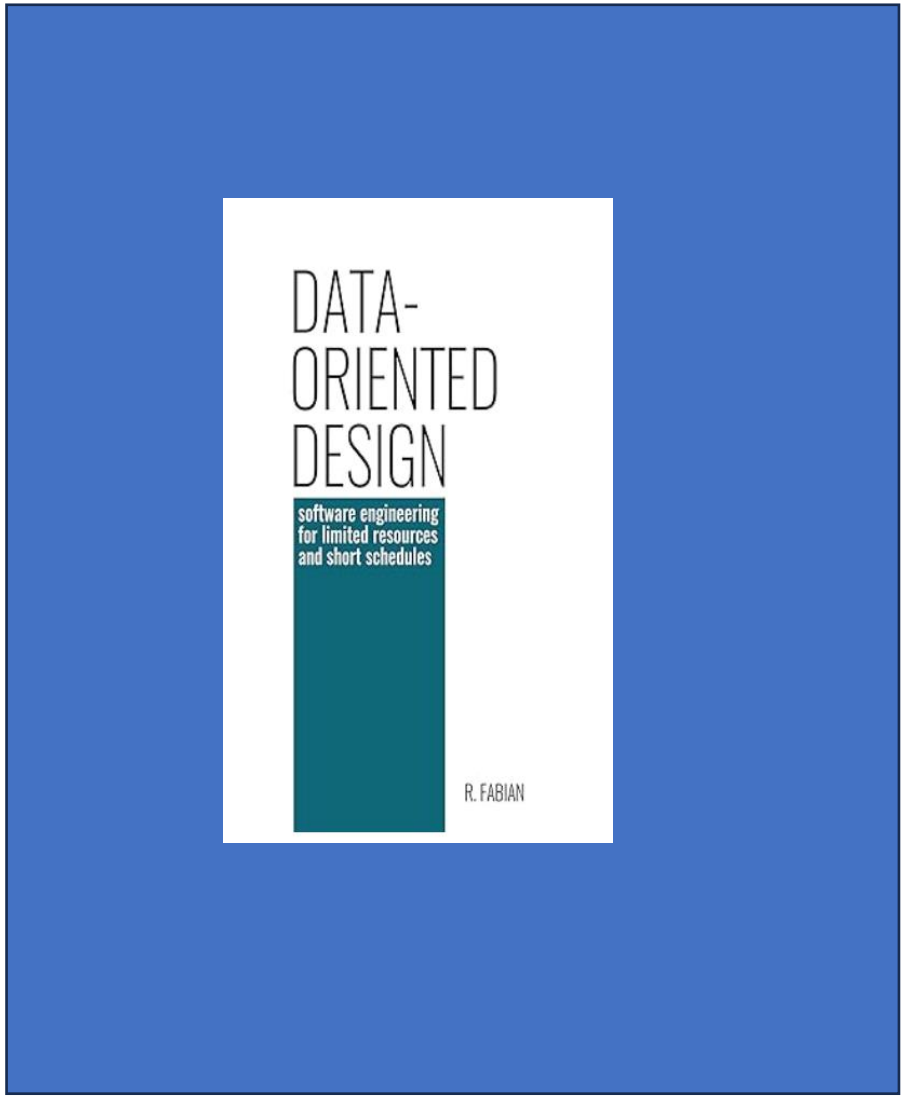
Data-Oriented Design Principles

- Don’t solve problems you don’t have ????? Lean
- Latency and throughput not the same
- Where one there are many=> solve most common issue first
- **CONTEXT** the more you have the better you can make your solution
- **NUMA** prebuilt data access extends back right to creation process in gaming
- **Software** is a real thing running on real hardware
- **Reason** must prevail (evidence)

Data-Oriented Design Book Richard Fabian

- Also mentions power of ordering
- This will be a common underlying thread

- https://www.dataorienteddesign.com/dodbook/



Comparison of DOD and Object-Oriented Design (OOD)

- **Data-Oriented Design (DOD)**:
  - Focuses on transforming data from one form to another.
  - Emphasizes understanding the data to understand the problem better.
  - Different problems require different solutions, and different data implies different solutions.
  - Understanding the cost of operations is crucial to understanding the problem.
  - Understanding the hardware helps in understanding the cost and the problem.
  - Code is data, and everything is a data problem.
- **Object-Oriented Design (OOD)**:
  - A core concept in OOD is information hiding, which simplifies interactions between objects.
  - Encourages developing a simpler model of the world and building higher-level structures without needing to know the details.
  - Unfortunately, this hides what actually happens in terms of data transformations.
  - The system structure often resembles the structure of the teams that built it (Conway's Law).
  - Teams form around components, leading to hidden data transformations and lack of access to information, which hinders reasoning about non-functional behaviour (performance).

Micro Architecture Optimisations C++ Alliterations

- Trash the templates (code bloat)
- Vilify the virtual
- Imbibe the inlines
- Ban the bool
- Can caching - recalculate
- Love the lambdas
- Eviscerate Exceptions

Micro Architecture Optimisations
C++ Alliterations

- Trash the templates (code bloat)
- Vilify the virtual
- Imbibe the inlines
- Ban the bool
- Can caching - recalculate
- Love the lambdas
- Eviscerate Exceptions

- WE ARE NOT FISHING FOR MINNOWS

THE REAL PROBLEM

THE MISSING STEP

Real Life Solution Is Highly Dimensional

- Decomposition and algorithm choices
- Compile time off-loading with constexpr
- Spatial layout
- Execution ordering
- Vectorization
- Caching and execution ordering
- Handling Randomness and look-ups

- WE MUST CONSIDER THESE FACTORS IN CONTEXT OF PROBLEM DOMAIN

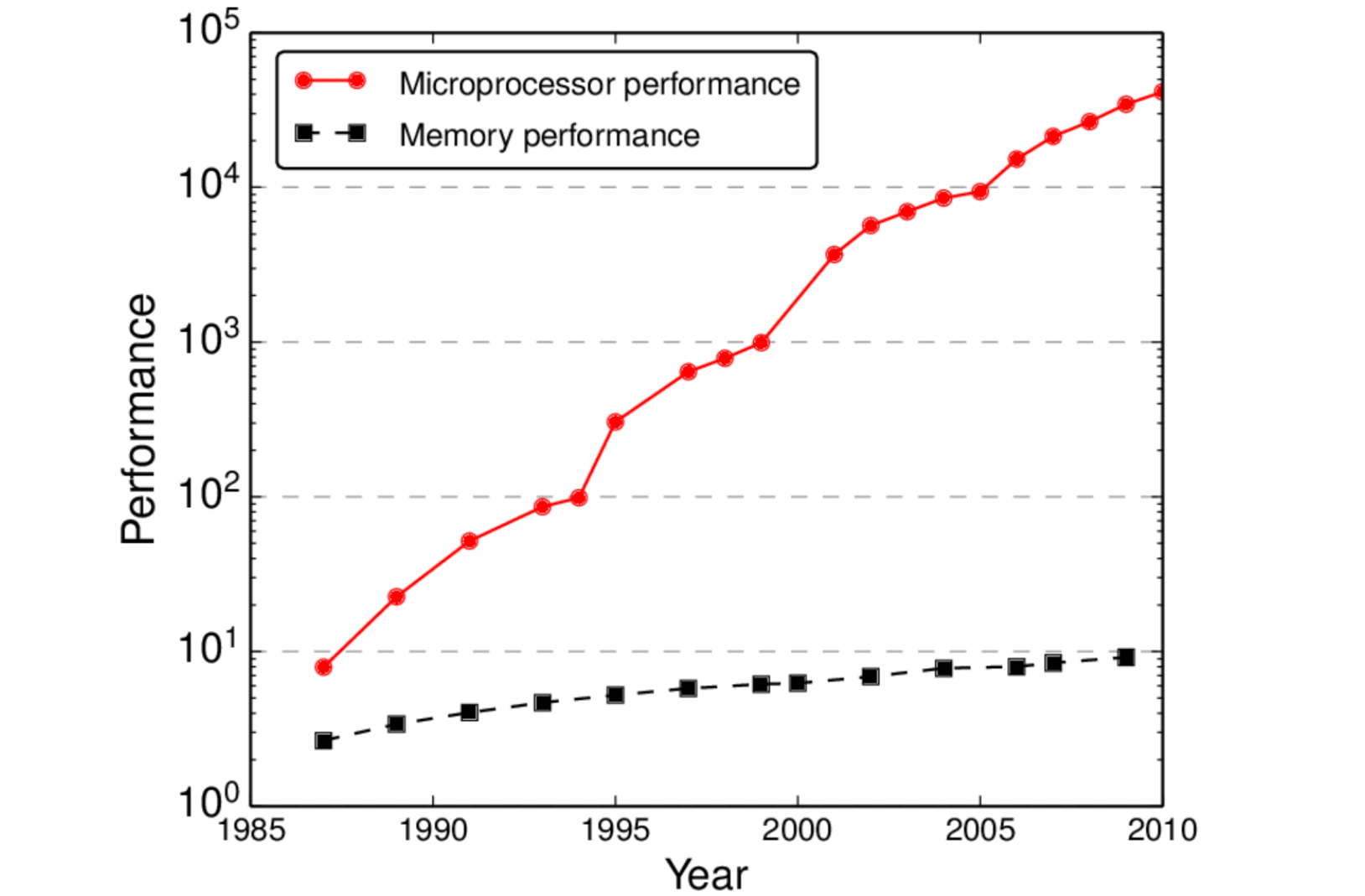
Sub Problem Grouping

- Logical .. Algorithmic
- Physical spatiotemporal ordering, layout and sequencing
- Problem Domain
  - Idiosyncratic utility function
  - Idiosyncratic side information/ constraints
  - Idiosyncratic invariants or data patterns to exploit

Additional Tools

- Polya problem-solving
- Skiena algorithmic design
- Socratic questioning
- Data orientation
- Vectorization

Hardware
The Challenge


Microprocessor performance
Memory performance
Performance
Year
10^5
10^4
10^3
10^2
10^1
10^0
1985
1990
1995
2000
2005
2010

Memory access costs

- Register
- L1 4 cycles 32K -48 k data
- L2 10 to 25 cycles 256k to 1Mb
- L3 40 cycles shared to 100Mb
- RAM 200 cycles

- See Performance Analysis and Tuning on Modern CPUs

Modern processors

- Out-of-order superscalar processing – Instruction-level parallelism
- Single Instruction Multiple Data
- Modern processors pay a heavy price when not able to receive a continuous stream of data
- Engineered caching solutions to mitigate slower memory
- Don’t like branching, speculative execution – branch prediction

Hardware properties

- Important processor state:
- The pipeline
- What we hold in registers
- What we hold in cache memory L1 L2 L3
- What we hold in the instruction cache
- What-ever state is held by predictive hardware
- Outcomes of branching decisions
- What we recently processed (functions/ instructions)
- What regions of memory we recently accessed

Important Idioms

- Memory Access, Ordering and Layouts
- Vectorization
- Pre-computing The fastest code that never runs.

The Power of Ordered Data

- Layout data in a way that enables the fastest processing
- Sorting the order of your data enables the use algorithms that can exploit this order.

The Power of Ordered Data

Why Pre-Sorting Helps

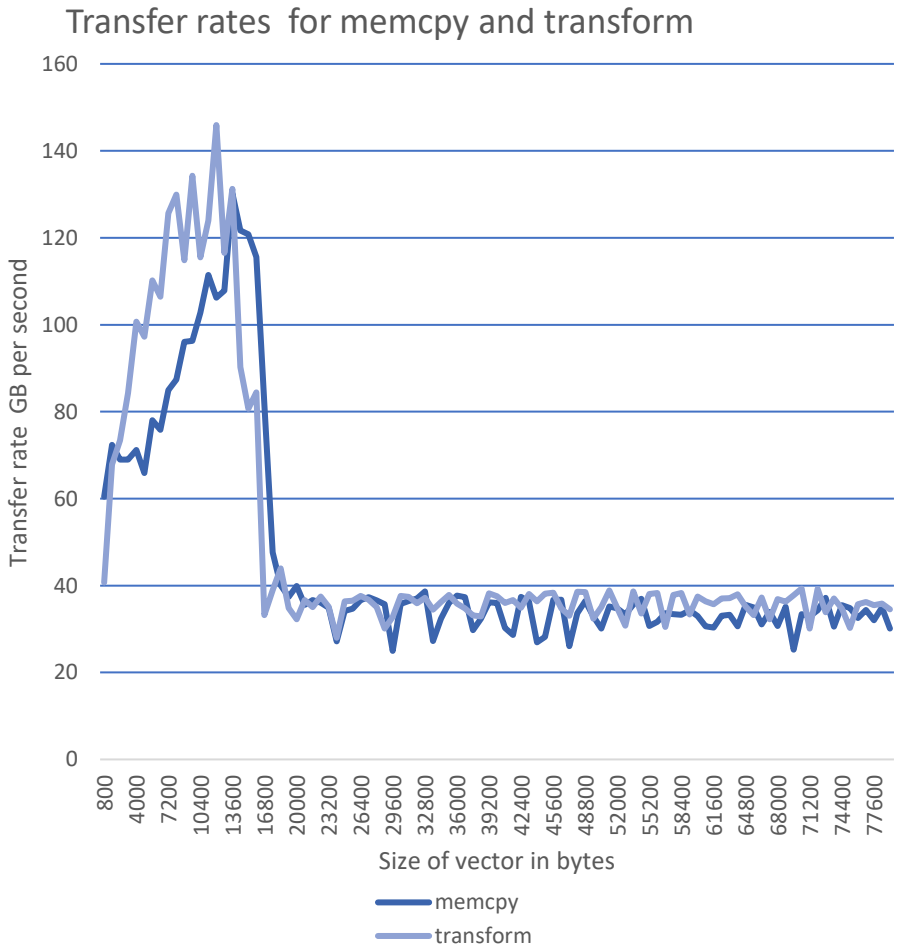
Use Case	Without Sorting	With Pre-Sorting	Speed Gain
Binary Search	O(N)	O(log N)	✅ **Exponentially Faster**
Finding Pairs (Two-Sum)	$O(N^2)$	O(N log N)	✅ **Much Faster**
Set Intersection	O(NM)	O(N + M)	✅ **Huge Speedup**
Removing Duplicates	$O(N^2)$	O(N log N)	✅ **Better Scaling**
Merging Sorted Lists	O(N log N)	O(N)	✅ **Linear Time**

The Power of Contiguous Data

- 1) Predictability of memory access pattern
- 2) Density of data representation.
- 3) When using homogenous primitive types (e.g. doubles) we can load multiple values into a vectorised register in one instruction

Moving contiguous data with SIMD

```
vmovupd     zmm0,zmmword ptr [rdx+rcx-40h]
vmovupd     zmmword ptr [rcx-40h],zmm0
vmovupd     zmm1,zmmword ptr [rdx+rcx]
vmovupd     zmmword ptr [rcx],zmm1
vmovupd     zmm0,zmmword ptr [rdx+rcx+40h]
vmovupd     zmmword ptr [rcx+40h],zmm0
vmovupd     zmm1,zmmword ptr [rdx+rcx+80h]
vmovupd     zmmword ptr [rcx+80h],zmm1
lea         rcx,[rcx+100h]
sub         r9,1
jne         ApplyTransformUR_X<Vec8d,
  <lambda_37c86e8c431697a3720d3d806be773b5> >+0C0h (07FF770A71F20h)
```



The Power of Contiguous Access

Which is Fastest?

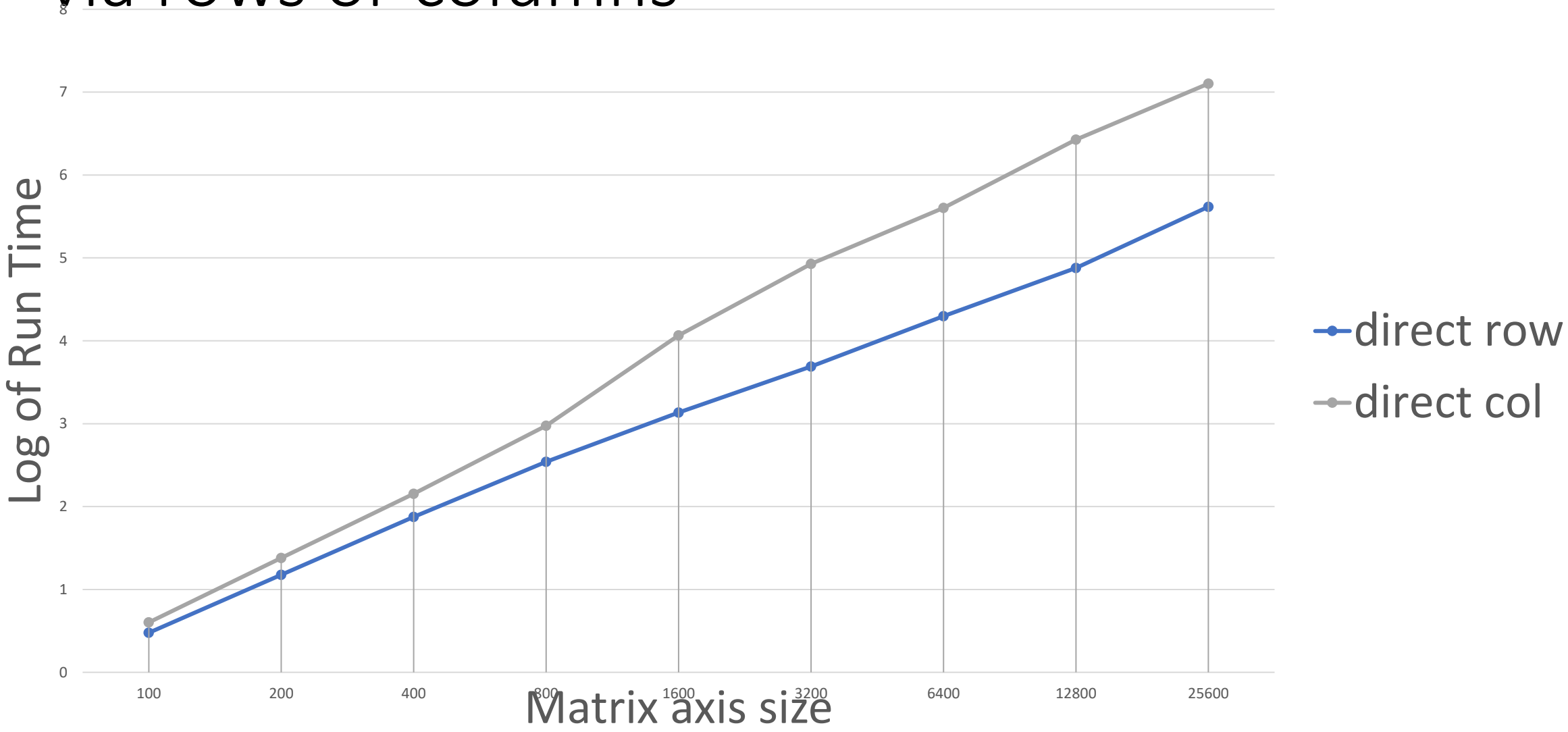
```cpp
// Function to sum elements in a 2D array using row-major indexing (direct loop indexing)
long long sumRowMajorDirect(const vector<vector<int>>& arr) {
    long long sum = 0;
    int size = arr.size();
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            sum += arr[i][j];
        }
    }
    return sum;
}

// Function to sum elements in a 2D array using column-major indexing (direct loop indexing)
long long sumColumnMajorDirect(const vector<vector<int>>& arr) {
    long long sum = 0;
    int size = arr.size();
    for (int j = 0; j < size; ++j) {
        for (int i = 0; i < size; ++i) {
            sum += arr[i][j];
        }
    }
    return sum;
}
```

Its Not Just Loop Interchange

- Physical access to strided V contiguous memory a common optimisation or performance bug

Log run time for summing using direct access via rows or columns

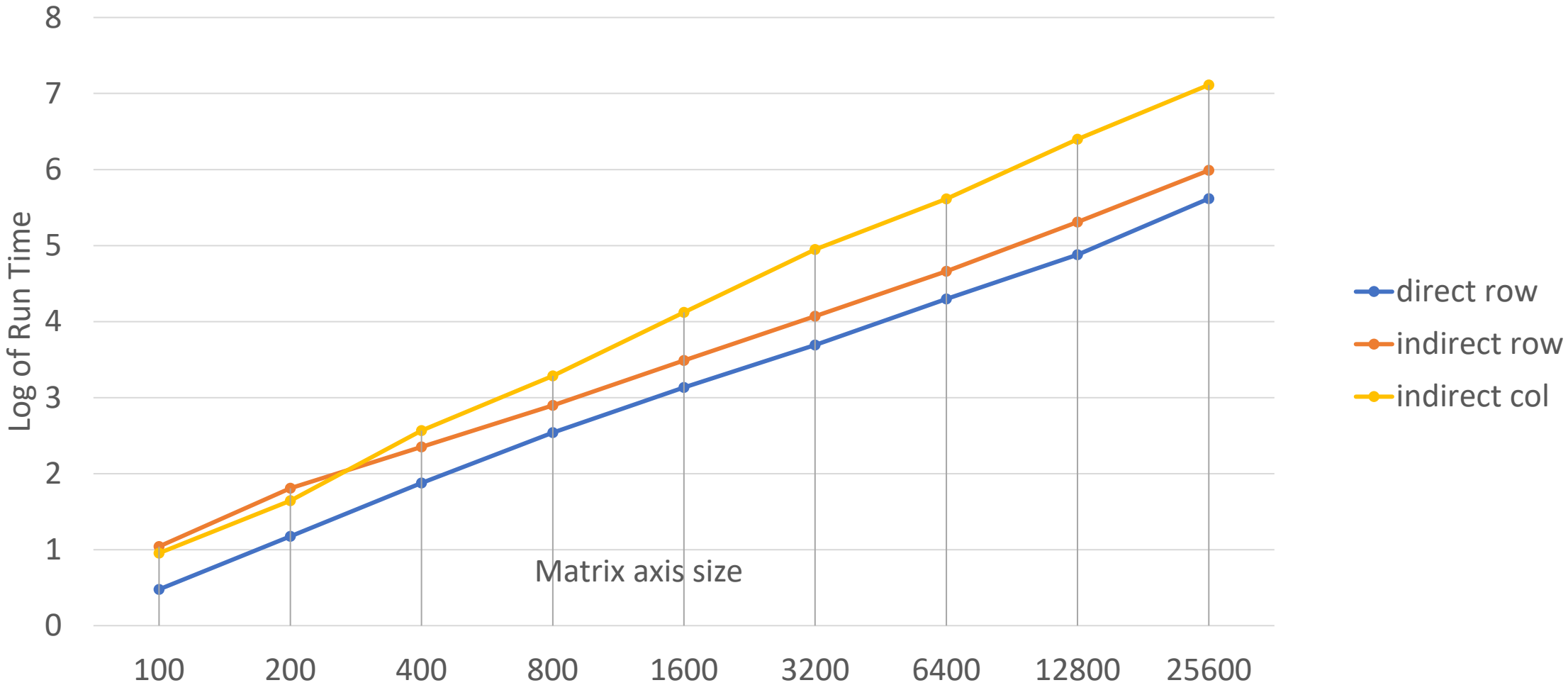


Add a level of indirection

```cpp
// Function to generate the row-major access path (coordinates).
vector<pair<int, int>> generateRowMajorAccessPath(int size) {
    vector<pair<int, int>> access_path;
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            access_path.push_back({ i, j });
        }
    }
    return access_path;
}
```

```cpp
// Function to sum elements in a 2D array using a pre-calculated access path
long long sumWithAccessPath(const vector<vector<int>>& arr, const vector<pair<int, int>>& access_path) {
    long long sum = 0;
    for (const auto& coord : access_path) {
        sum += arr[coord.first][coord.second];
    }
    return sum;
}
```

The access path is not knowable: Still, row access is better.

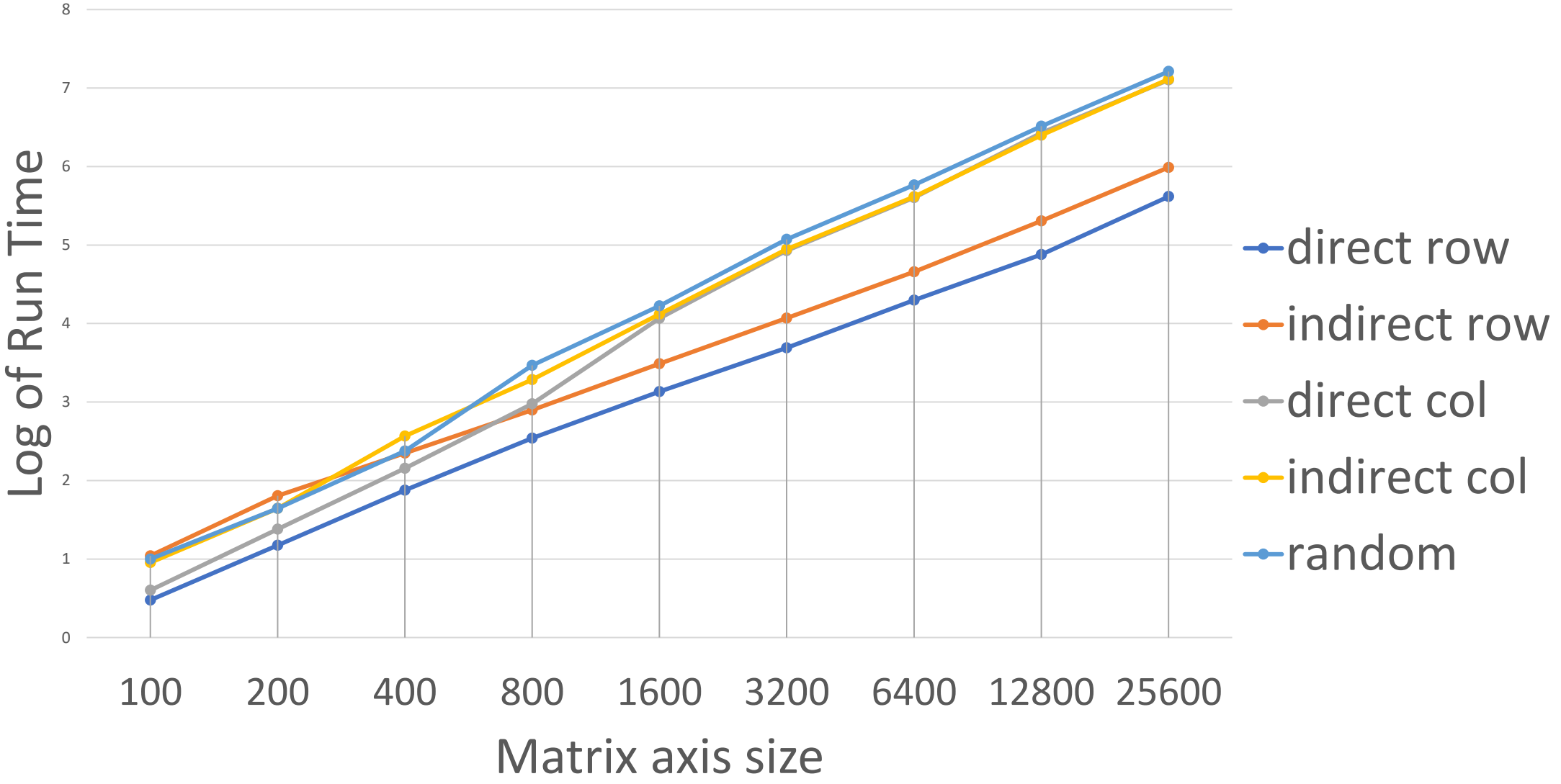


- Sequential access to contiguous memory is fastest, even when we don’t know that we will be accessing contiguous memory at compile time

- Shuffle indirect data, to give random access

```cpp
// Function to shuffle the column-major access path to create a "shuffled" path
vector<pair<int, int>> generateShuffledAccessPath(const vector<pair<int, int>>& column_major_access_path) {
    vector<pair<int, int>> shuffled_path = column_major_access_path;
    random_device rd;
    mt19937 g(rd());
    shuffle(shuffled_path.begin(), shuffled_path.end(), g);
    return shuffled_path;
}
```

Predictable access paths V random paths



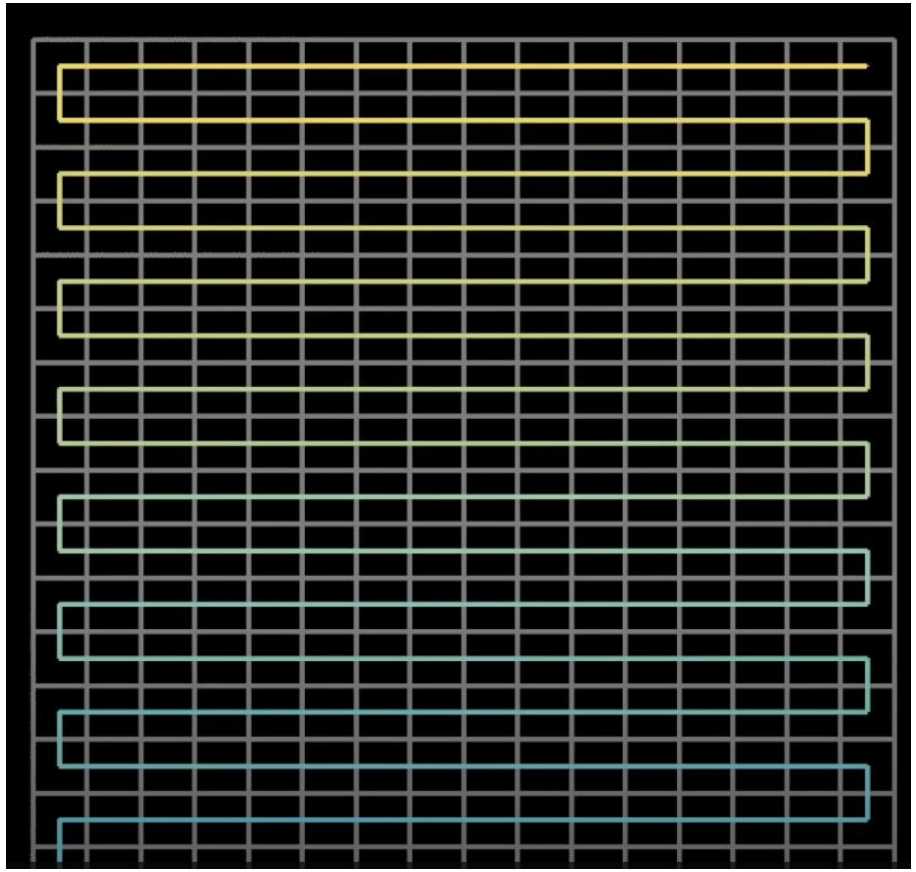
Random Access Strategies

- 1) Avoid: Use ordered/ sequential access sorted
- 2) Minimise: Shared look-up (look up vectors of results)
  - Control flow
  - Cost of look-up amortised over vector size + contiguous read.
- 3) Constrain: minimise the range searched.
- 4) Multi-dimensional locality of reference space-filling curves

Higher Dimensional Data

How do we maintain locality of reference when we have 2D data?

Rectangular layout : Typical array style

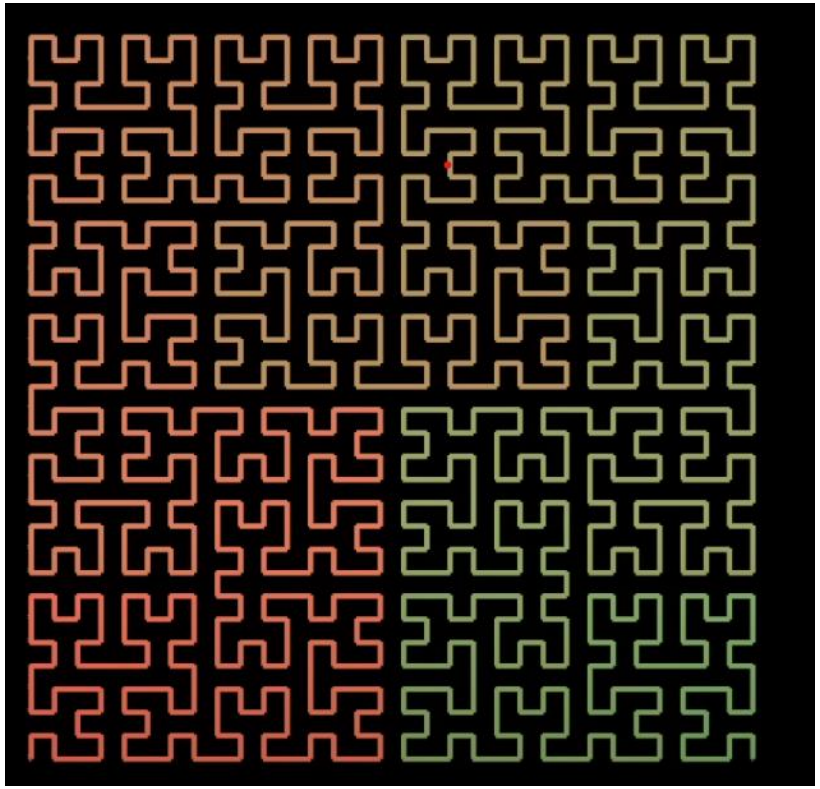


Distance between vertically separated points is the width of the rectangle.

Space Filling Curves

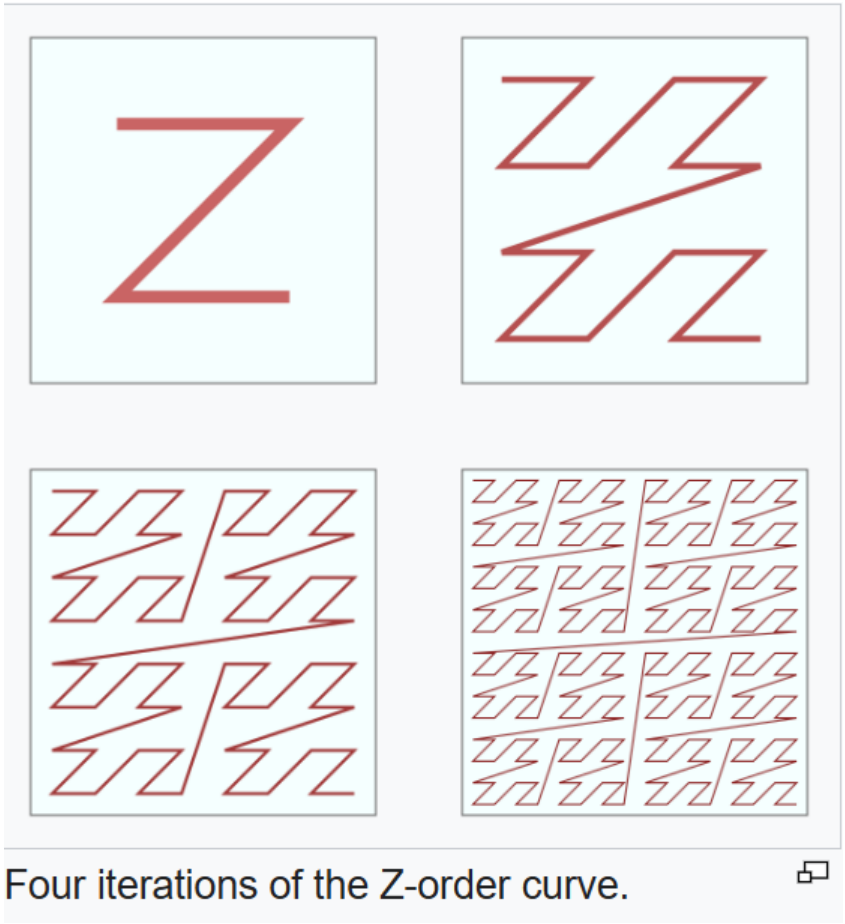
- A space-filling curve maps all points in a higher dimension to one dimension but preserves high dimensional locality in one dimension (memory)

Hilbert Curve



Vertically separated points are much closer together on when arranged on the Hilbert curve layout.

Morten Curve

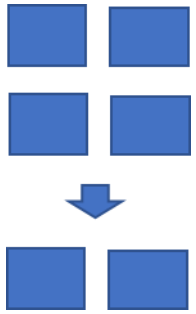


Four iterations of the Z-order curve.

Experiment

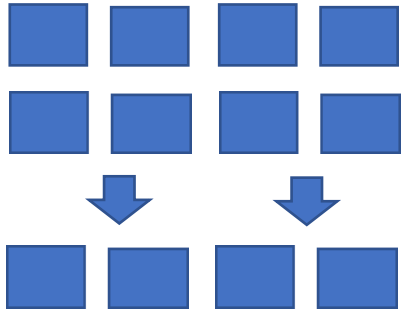
- Random Access to matrix data when stored in space-filling curve
- https://godbolt.org/z/jabx1b6zv Morton
- Note, don’t see so much benefit on the Godbolt servers, machines with smaller cache can show the effects. Can’t run large enough examples without process kill.

The power of using vectorised instructions
Single Instruction Multiple Data



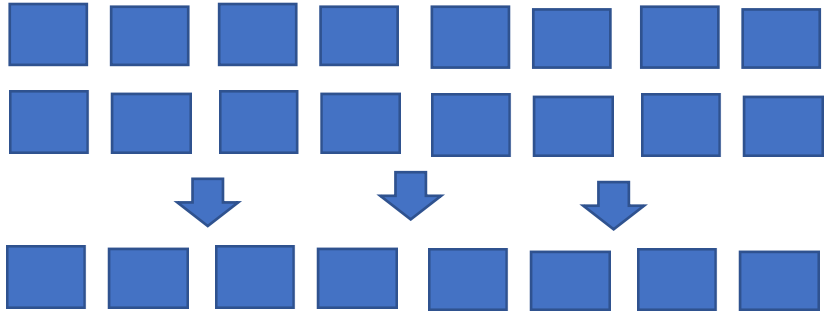
XMM register

SSE2



YMM register

AVX2



ZMM register

AVX512

Array of Structures


```
struct StudentData
{
    long m_ID;
    double m_GPA;
    int m_age;
};

std::vector<StudentData>    physics_students_data_AOS;
```

Structure Of Arrays


```
struct physics_students_data_SOA
{
    std::vector<long>   m_IDs;
    std::vector<double> m_GPAs;
    std::vector<int>    m_Ages;
};
```

Execution Orchestration – Data Partitioning

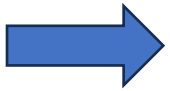
- Partition the data so that values that will be used by the same function are stored together
- We can load more easily into a register and apply the same sequence of transformations in a vectorised way.
- Different derived types may have different virtual function implementations -> we would have to partition the data on basis of derived class type

Loop Hoisting and Branching Conditions

- Membership of the Array Of Structs vector, can also be governed by some conditional or look-up logic

Partitioned /split SoA

```
for (const auto& item : Items)
{
    if (item.value > big)
    {
        doBigStuff(item.other_value);
    }
    else
    {
        doSmallStuff(item.other_value);
    }
}
```



```
//partition Items ->  BigItems  + SmallItems
for (const auto& item : BigItems)
{
    doBigStuff(item.other_value);
}

for (const auto& item : SmallItems)
{
    doSmallStuff(item.other_value);
}
```

No branch prediction or masking , just applying the function

Looking up a vector of values once rather than looking up a set of scalars multiple times

- For maps, Instead of a key to a scalar value, we can have a key into a contiguous vector of values (aligned and padded for SIMD use)
- When the objects we manage are vectors (not scalars) the cost of branching and lookups can be amortised over all elements in the vector
- When we have large vectors the cost of look-ups falls away
- We could even consider introducing more sophisticated caching strategies that we would not have considered for say a single double

Compile Time Pre-compute with constexpr

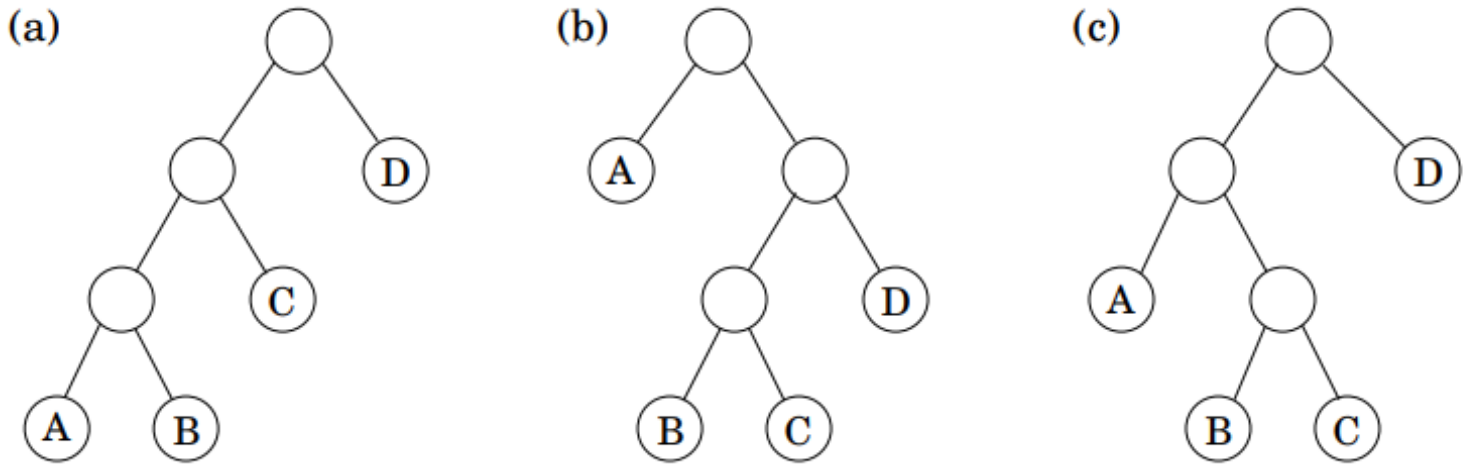
What is Matrix Chain Multiplication?

- Given a sequence of matrices **A1, A2, ..., An**, we want to multiply them.
- **Matrix multiplication is associative**:
  - Example: (A × B) × C = A × (B × C)
- **Goal**: Find the best order to multiply matrices to **minimize scalar multiplications**.

Why Order Matters

- **Example: Different Orders, Different Costs**
- Given matrices:
  - A1 (10 × 100), A2 (100 × 5), A3 (5 × 50)
- Two ways to multiply:
  1. **(A1 × A2) × A3** → (10 × 100) × (100 × 5) → 5000 + (10 × 5 × 50) → **7500 ops**
  2. **A1 × (A2 × A3)** → (100 × 5) × (5 × 50) → 25000 + (10 × 100 × 50) → **75000 ops**
- **Conclusion:** Different orders → Different computation costs!

Matrix Chain Multiplication as a Binary Tree



(a) ((A × B) × C) × D;
(b) (b) A × ((B × C) × D);
(c) (c) (A × (B × C)) × D.

A vast number of combinations for large trees

Dynamic Programming Approach

- Define cost table **C[i][j]** = Minimum number of operations to multiply matrices **Ai to Aj**.
- Fill the table **bottom-up**, solving for **small chains first**.
- C(i, j) = min {C(i, k) + C(k + 1, j) + $m_{i-1} \cdot m_k \cdot m_j$} // cost to split @k cost after split + cost at k
- Store **split points** to reconstruct optimal order.
- **Time Complexity: O(N^3), Space Complexity: O(N^2)**.
- https://godbolt.org/z/9a9TP6aos

constexpr dynamic programming to compute optimal order at compile time

- https://godbolt.org/z/b3x3ao14K performance example.
- "Algorithms" Gupta et al, Chapter 6 .5
  https://people.eecs.berkeley.edu/~vazirani/algorithms/chap6.pdf

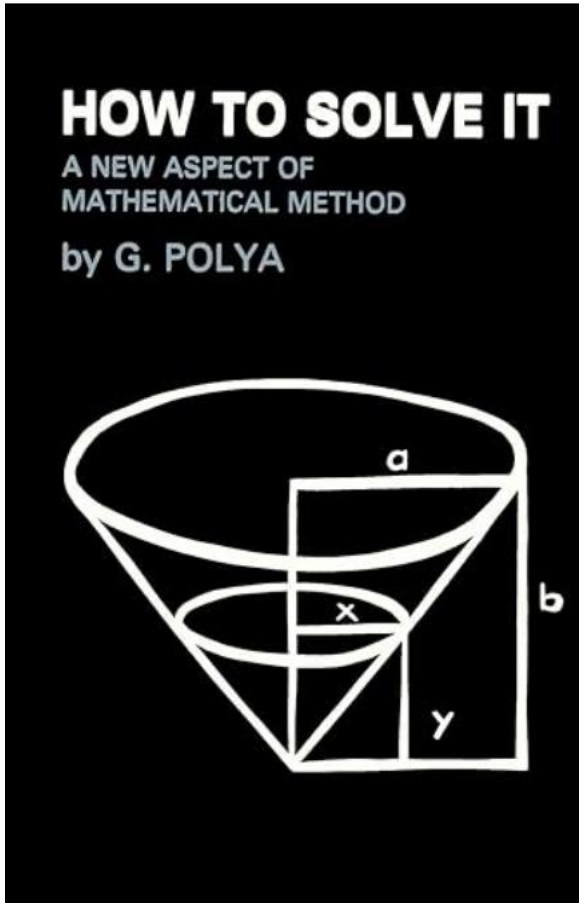
Solving the Design Problem

A Little Help from George Polya

George Pólya was a Hungarian-American mathematician. He was a professor of mathematics from 1914 to 1940 at ETH Zürich and from 1940 to 1953 at Stanford University. He made fundamental contributions to combinatorics, number theory, numerical analysis and probability theory.

He is also noted for his work in heuristics and mathematics education.[2] He has been described as one of The Martians,[3] an informal category which included one of his most famous students at ETH Zurich, John von Neumann.

'Everyone should know the work of George Polya on how to solve problems' Marvin Minsky



Polya In A NutShell

- George Pólya's method encourages you to:

1. **Understand** the context and clarify the problem. Restate the problem, ensure that you understand a terms definitions, conditions and data
2. **Plan** by looking for analogous problems or solutions and deciding on the right tools. break down into manageable parts; consider working backward; simplify the problem.
3. **Carry out** the plan with careful experimentation or calculation, checking the results of each step
4. **Reflect** on the result to improve future understanding and methods.

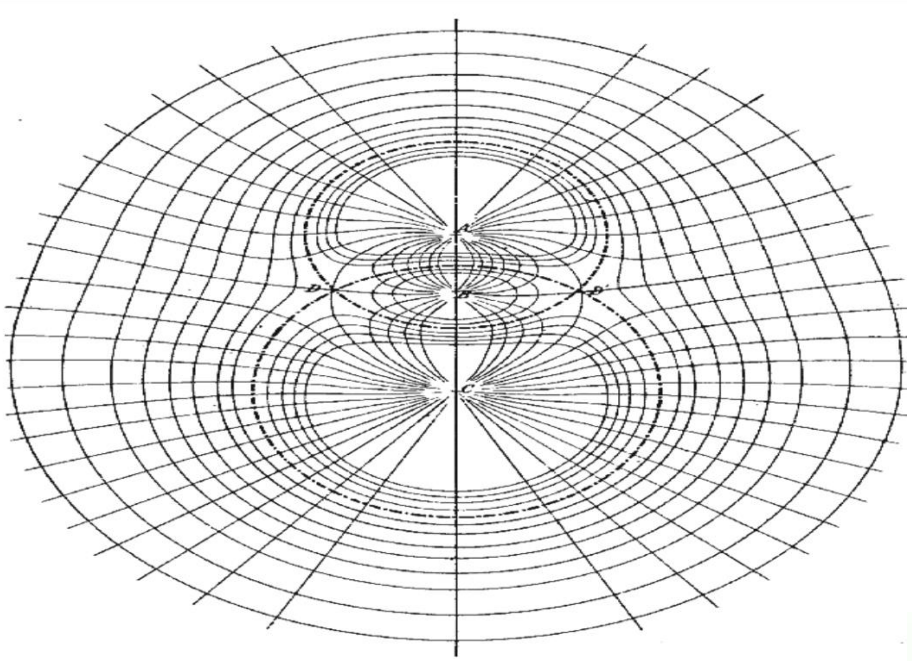
Understand The Problem Skiena

- What is the input
- What is the output
- Can I work an example by hand
- How accurate/optimal must my result be
- Data size .. All the data type questions
- How important is performance
- What type of problem is it? Numeric? Graph? String? Set? Geometric?

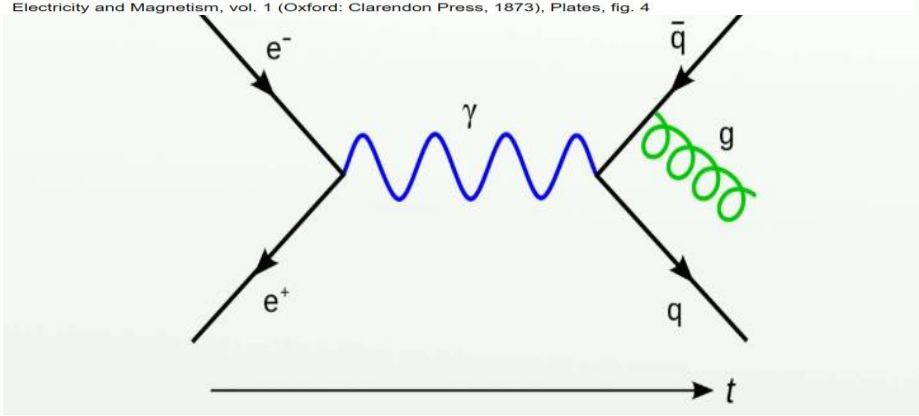
Understand the problem (Polya)

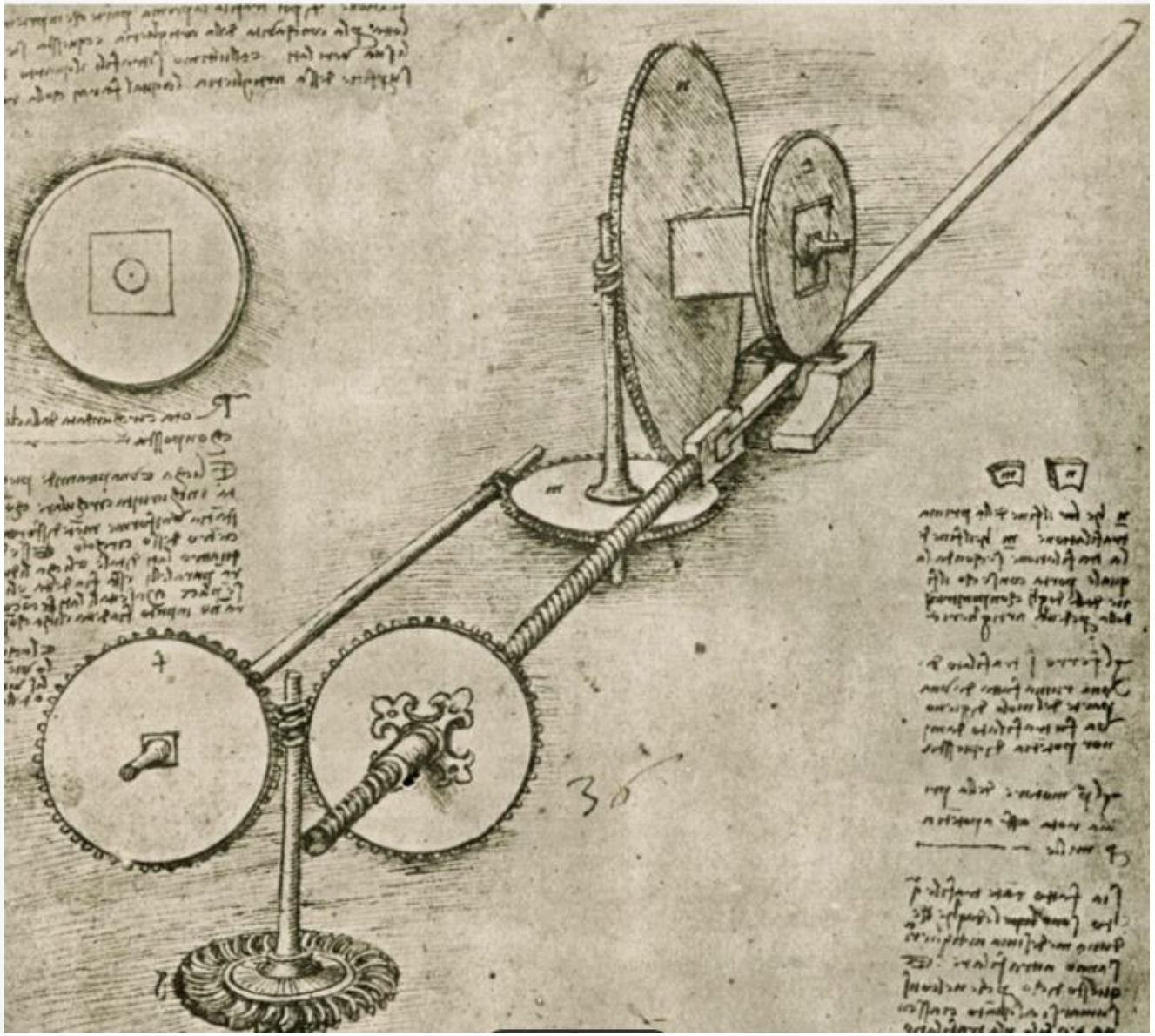
- What is the unknown? What are the data? What is the condition?
- Is it possible to satisfy the condition? Is it sufficient? Or redundant? Or contradictory?
- Separate various parts of the condition. Can I write them down?
- Draw a picture.

Draw a Diagram

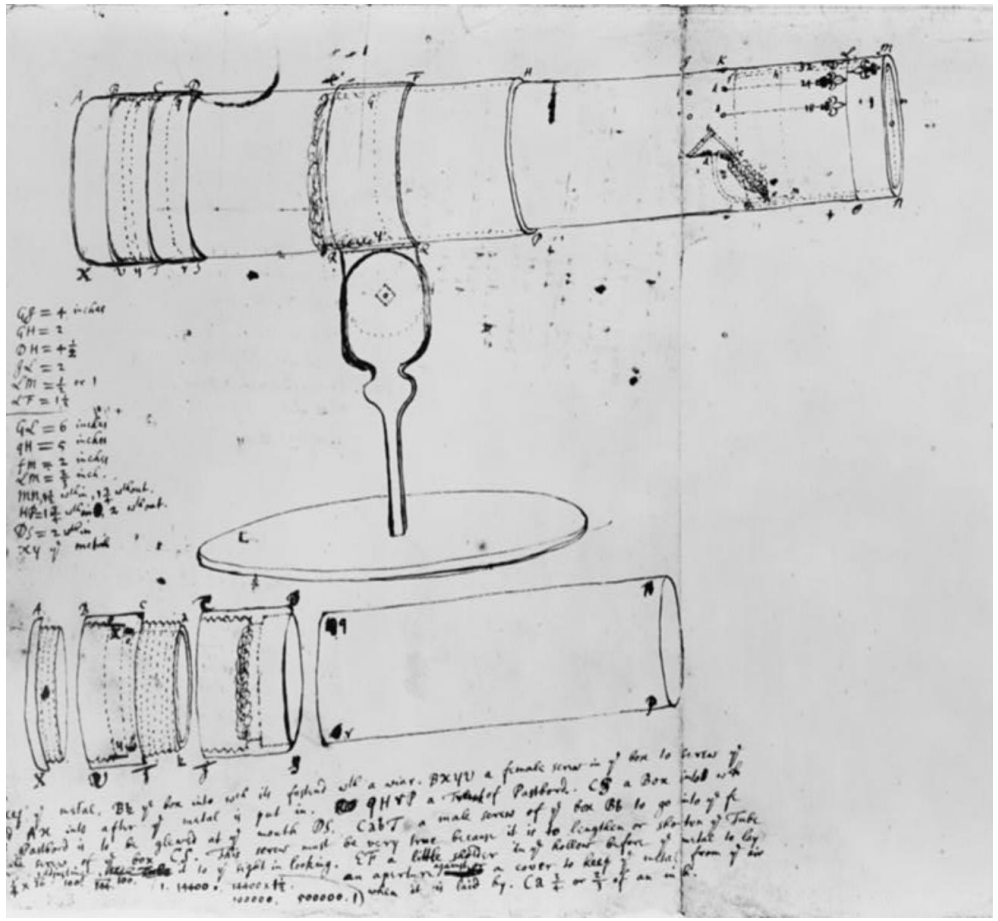


One of James Clerk Maxwell's examples for drawings of lines of force. From James Clerk Maxwell, A Treatise on Electricity and Magnetism, vol. 1 (Oxford: Clarendon Press, 1873), Plates, fig. 4

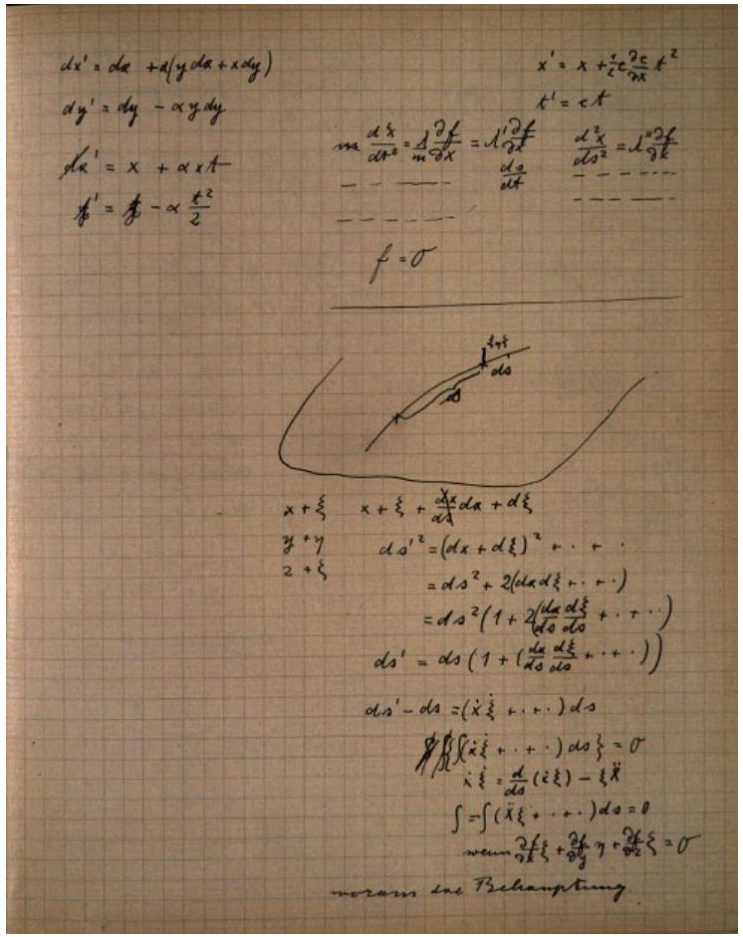




Newton



Einstein



Devise a Plan

- Find the connection between the unknown and the data. Possibly solve auxiliary problems if no immediate connection.
- Have you seen and solved the problem before ?
- Do you know a related problem?
- Look at the unknown. Do you know problems giving the same or similar unkowns? Could the same solution approach be used?
- Can you restate the problem differently? Dig back into problem definition in minutia.

Devise a Plan 2

- Try to solve a related or easier (auxiliary problem)
- Relax constraints, consider a more general or specific similar problem?
- What happens if you change the data, the unknown? Does this bring you closer to a solution?
- Did you use all the data? Using all the information in our problem space might give us conditions to exploit.
- Have you taken into account all essential notions involved in the problem?

Devise a Plan 3 – consider heuristics/tactics

- **Key Heuristic Strategies**:
- Analogy
- Decomposition
- Generalization and Specialization
- Working Backwards
- Auxiliary Elements (constructions, diagrams, notation, intermediate goals)
- Reduction: mapping problem to known problem

3 Carry Out The Plan

- Work through the tasks in the plan. Checking/testing the results of intermediate findings.

- Prototyping, measuring performance , checking that the faster code generates the correct results.

4 Reflection

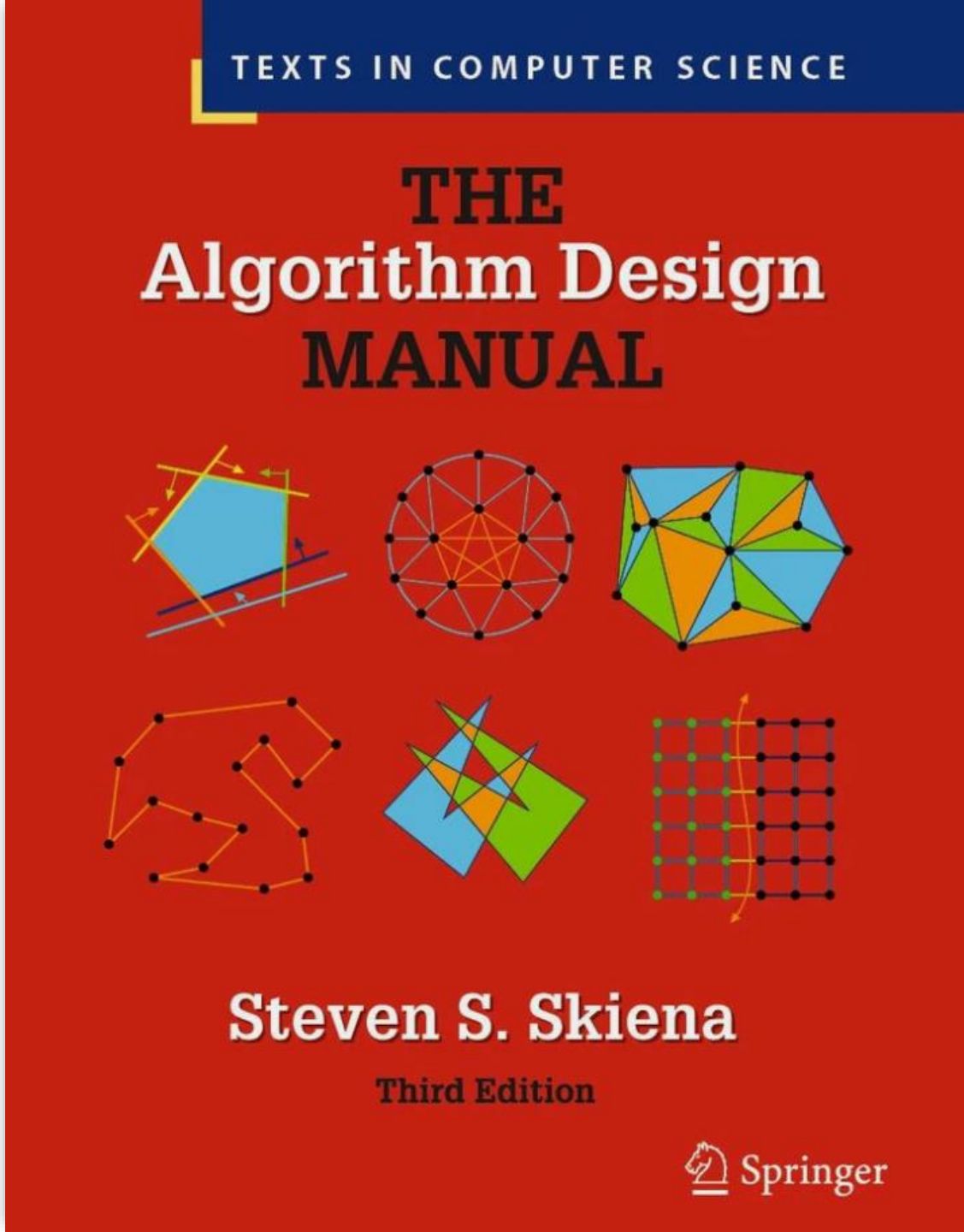
- Check the results
- Could I get to the same results via a different route
- Can I see the answer/ solution at a glance
- Can this effort/result solve other problems

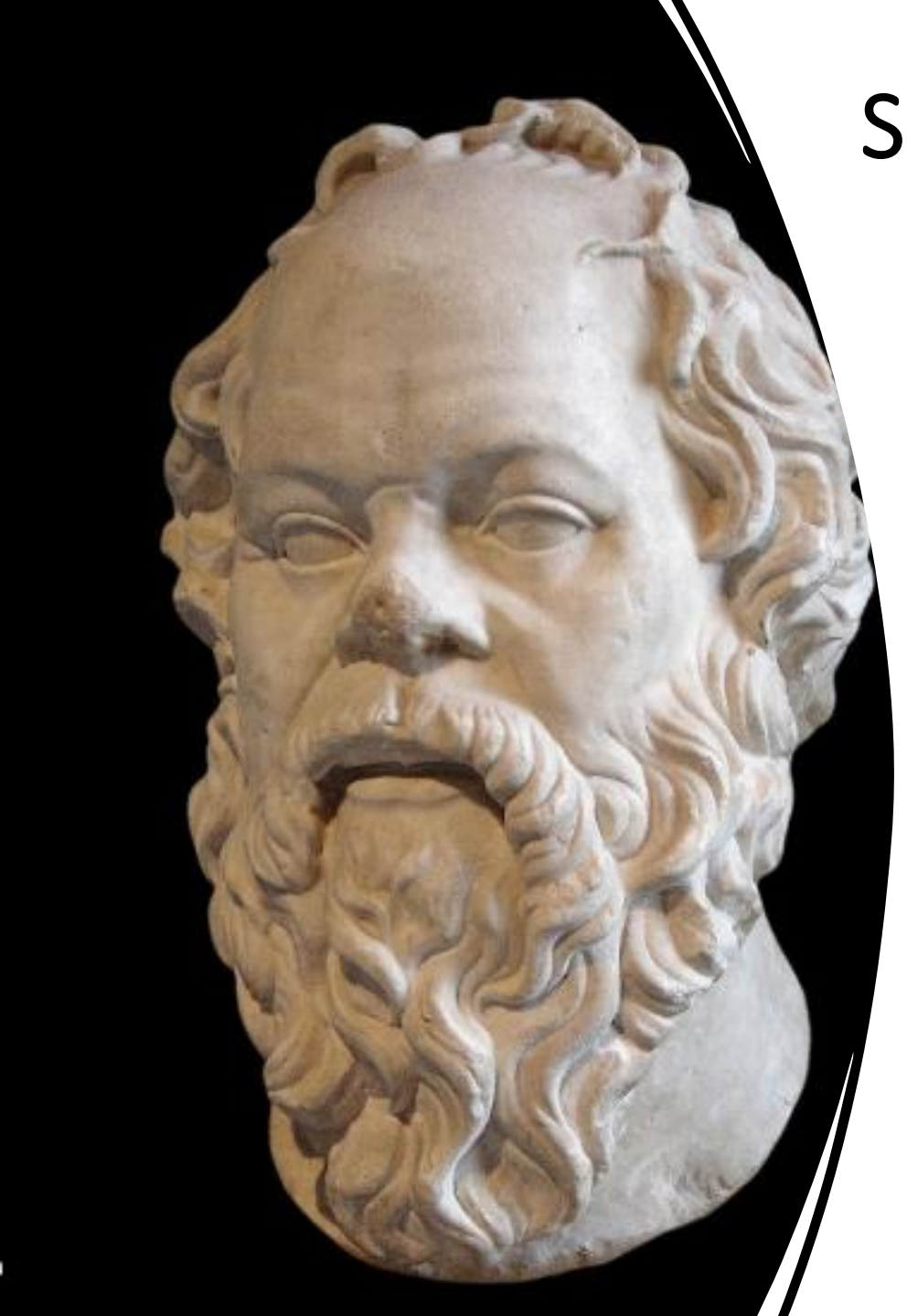
Diagnosis

- **Incomplete understanding of the problem:** The most common cause of failure. Lack of concentration on understanding the problem in the initial phase.
- **Planning Failure:** Two opposite faults
  - Rushing into calculation and construction without any plan or general idea
  - Clumsy waiting for an idea to come and taking no action to make it come quicker.
- **Execution Failure:** Carelessness, lack of checking each step. Failure to check the result is very common. (faster code giving the wrong answer ?)

Steven S Skiena

- The Algorithm Design Manual
- Has an ordered approach to working the problem.
- At the end credit refers the reader to Polya
- A very useful catalogue of problem/solutions





Socratic Questions on DOD

Type	Purpose	Example
Clarification	Define key concepts	"What do we mean by 'cache efficiency'?"
Probing Assumptions	Challenge beliefs	"Why do we assume OOP is the best approach?"
Probing Evidence	Check reasoning	"What benchmarks prove this design is faster?"
Alternative Views	Consider other perspectives	"What would an OOP advocate say about DOD?"
Implications	Explore consequences	"How will DOD affect maintainability?"
Questioning the Question	Reflect on our inquiry	"Are we focusing on the right problem?"

Essential Elements for Success

- Develop discovery through thoughtful questions: Socratic Questioning?
- Positive Mindset: curiosity, persistence and a growth mindset.
- Notes: Keep notes on thoughts, attempts, and processes.

Feeling Stuck is natural state

- Feeling stuck is part of the process
- Unless the problem is trivial and known to you, work is required
- Example of a Polya conducting a problem-solving session with students

- https://www.youtube.com/watch?v=h0gbw-Ur_do

EXAMPLES

Example Leave One Out Regression

Example Leave One Out Regression

- Machine Learning
- Trying to find predictive factors for simple linear relationships.
- Assessing feature vector/model quality.
- For each data point we forecast the value at that point using a model built from the whole data set minus that point. The difference between forecast and observed gives the error at each point.
- A final step of estimating quantiles or inter-quantile range as a metric for the quality of the fit, and suitability of the relationship as a predictor.
- Picture of distribution, noisy and ranges, robust against outliers/noise

The Application

- Use simple linear low (single dimension)
- Generally, will be an $O(N^2)$
- For each (N) data points
  - Fit (N -1) data points and compute residual

Regularised Linear Regression

- Makes more stable by adding a penalty for larger slopes

Understanding The problem

- Questions about data/ size
- Performance requirements
- Accuracy requirements
- Have I solved it before?

- Known solutions, call least squares fit many times with permuted leave one out data

Understanding the problem

- What is the algorithm used
- Why is it so expensive / slow

Model and Objective function

Model $\hat{y}_i = \beta_0 + \beta_1 x_i, \quad i = 1, \ldots$

Ridge Objective $\min_{\beta_0, \beta_1} \sum_{i=1} [y_i - (\beta_0 + \beta_1 x_i)]^2 + \lambda \beta_1^2$

- λ: regularization parameter
- β_0: intercept (not penalized)
- β_1: slope (penalized)

The Design Matrix

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

We include an intercept by adding a column of 1's:

Regularisation

We penalize only β_1. Hence, our penalty matrix is:

$$\Lambda = \begin{bmatrix} 0 & 0 \\ 0 & \lambda \end{bmatrix}.$$

Interpretation:

- Top-left entry = 0 $\Longrightarrow$ do **not** penalize β_0.
- Bottom-right entry = λ $\Longrightarrow$ penalize β_1 with strength λ.

The Normal Equations

The **Normal Equation** for ridge with intercept (unpenalized) is:

$$\left(X^\top X + \Lambda\right)\beta = X^\top \mathbf{y}$$

$$\beta = \left(X^\top X + \Lambda\right)^{-1} X^\top \mathbf{y}$$

The Essence of the Regression Calculation

$$X^\top X = \begin{bmatrix} \mathsf{N} & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} \mathsf{N} & S_x \\ S_x & S_{xx} \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} 0 & 0 \\ 0 & \lambda \end{bmatrix}$$

$$X^\top X + \Lambda = \begin{bmatrix} \mathsf{N} & S_x \\ S_x & S_{xx} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} \mathsf{N} & S_x \\ S_x & S_{xx} + \lambda \end{bmatrix}$$

$$X^\top \mathbf{y} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix} = \begin{bmatrix} S_y \\ S_{xy} \end{bmatrix}$$

Final Form of β

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \mathrm{N} & S_x \\ S_x & S_{xx} + \lambda \end{bmatrix}^{-1} \begin{bmatrix} S_y \\ S_{xy} \end{bmatrix}.$$

2×2 Inverse

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}^{-1} = \frac{1}{ac - b^2} \begin{bmatrix} c & -b \\ -b & a \end{bmatrix}.$$

Here $a = 5,\ b = S_x,\ c = S_{xx} + \lambda,\ \Delta = ac - b^2.$

Explicit β_0 and β_1

$$\beta_0 = \frac{(S_{xx} + \lambda)\, S_y - S_x\, S_{xy}}{\mathsf{N}\,(S_{xx} + \lambda) - (S_x)^2}$$

$$\beta_1 = \frac{\mathsf{N}\; S_{xy} - S_x\, S_y}{\mathsf{N}\,(S_{xx} + \lambda) - (S_x)^2}$$

Planning

- Identify slow /expensive parts
- Explore auxiliary problems that reflect the slow parts
- Draw some pictures
- Generalise solutions to auxiliary problems
- Construct a new algorithm

The slow bit (repeated N times)

$$X^\top X = \begin{bmatrix} \mathsf{N} & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} \mathsf{N} & S_x \\ S_x & S_{xx} \end{bmatrix}$$

- But we are repeating this N times with slightly different data sets
- It's the first X^TX term that introduces the O(N) dependence into the fitting
- Also, the last term in X^TY

The slow bit (repeated N times)

$$X^\top X = \begin{bmatrix} \mathsf{N} & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} \mathsf{N} & S_x \\ S_x & S_{xx} \end{bmatrix}$$

How are we going to make this go faster?

The slow bit (repeated N times)

$$X^\top X = \begin{bmatrix} \mathsf{N} & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} \mathsf{N} & S_x \\ S_x & S_{xx} \end{bmatrix}$$

How are we going to make this go faster?

Unsequenced reduction and transform reduce!
Scale with threads and SIMD (if we are lucky)

If only this was dereferencing a nullptr!

STOP

If only this was dereferencing a nullptr!

- We have applied an answer we know, to a problem we recognise.
- We have not considered the **context** fully, and have only considered what we might do on an existing inner loop.
- Huge restrictions in the scope of solutions we might consider

Expand scope

- Consider speeding up the whole set of X^TX not just each perturbed version.

- This is a key element

Consider a simpler auxiliary problem

$$X^\top X = \begin{bmatrix} \mathrm{N} & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} \mathrm{N} & S_x \\ S_x & S_{xx} \end{bmatrix}$$

How does this single element compute, vary over all the different leave-one-out summations we will do?

Alternatives for auxiliary problem

- We have aux problem selection
- Example, we went for leave one out sum over x_i rather than sum of 1 or sum of $x_i y_i$ x_i^2
- Looked non-trivial but illustrative
- Also, we don't care too much we want just something that has the most relevance
- We can always try a different problem if we don't get the benefit from this one.
- We are choosing problems that we think will teach us something. Not necessarily or even, be a component of the solution.

Consider a simpler auxiliary problem

- One of the summations in the X^TX matrix, for all the leave-one-out perturbations
- Pick leave one out sum of X_i
- Hey, why not draw a picture.

Summation rows leaving out an element

1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1	11.11	
1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1		12.12
1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9		11.11	12.12
1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8		10.1	11.11	12.12
1.1	2.2	3.3	4.4	5.5	6.6	7.7		9.9	10.1	11.11	12.12
1.1	2.2	3.3	4.4	5.5	6.6		8.8	9.9	10.1	11.11	12.12
1.1	2.2	3.3	4.4	5.5		7.7	8.8	9.9	10.1	11.11	12.12
1.1	2.2	3.3	4.4		6.6	7.7	8.8	9.9	10.1	11.11	12.12
1.1	2.2	3.3		5.5	6.6	7.7	8.8	9.9	10.1	11.11	12.12
1.1	2.2		4.4	5.5	6.6	7.7	8.8	9.9	10.1	11.11	12.12
1.1		3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1	11.11	12.12

Lots of repetition

Summation rows leaving out an element

1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1	11.11	
1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1		12.12

Can we re-use the sum for the first row to calculate the second row ?

Summation rows leaving out an element

1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1	11.11	
1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1		12.12

Can we re-use the sum for the first row to calculate the second row ?

$$Sum\_loo\_1 = Sum\_loo\_0 + x\_0 - x\_1$$
$$Sum\_loo\_2 = Sum\_loo\_0 + x\_0 - x\_2$$

Lots of repetition

Summation rows leaving out an element

1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1	11.11	
1.1	2.2	3.3	4.4	5.5	6.6	7.7	8.8	9.9	10.1		12.12

But Sum_loo_0 + x_0 is just the sum of all the elements
So each leave one out sum (i) = Sum_all – x_i

So compute the sum over the whole row
Then compute leave one out sums by subtracting the left-out-element

The leave one out sum $S_k = S_N - x_k$

Where $S_N = \sum_{i=o}^{i=N} f(x)$

Generalisation ?

- Does this generalise to all our leave one out sums ?

What are the costs for this solution?

- First sum over whole row to get Sn cost N
- Then create each leave one out sum Sk SN –xi cost N operations
- We can compute all our leave one out sums in 2 N operations

The New Algorithm : Summations over set

- For all leave one out sums, sum over the complete set S_N
- $S_N(x)$
- $S_N(1)$
- $S_N(x^2)$
- $S_N(xy)$
- $S_N(y)$

- Generate Leave one out at index i, by subtracting $f(x_i)$ vector from the sum
- For each set of Loo values N ops for reduce, N ops to create N Loo sums
- O(2N)

Leave one out vector sums

- For all leave-one-out sums, sum over the complete set S_N
- Leave one out $Sx\_i = S_N(x) - x\_i$ vector of $(x\_i * -1) + Sx\_i$
- Leave one out $Sn\_i = S_N(1) = N-1$
- Leave one out $Sxx\_i = S_N(x^2) - (x\_i * x\_i)$ element-wise multiply $x\_i * x\_i$
- Leave one out $Sxy\_i = S_N(xy) = -(x\_i * y\_i)$ element-wise multiply $x\_i\ y\_i$
- Leave one out $S_{y\_i} = S_N(y) - y\_i$ vector of $(y\_i * -1) + Sy\_i$

New Algorithm

- Generate the new fits using the closed form expressions for Betas
- 1) Generate sums over whole data set Sx, Sxx, Sy, Sxy
- 2) Generate leave one out sums by subtracting vectors of x,xx,y,xy from corresponding sum
- 3) Evaluate closed form solution using vector operations

Hardware Memory and Vectorisation

- Use DR3

- Contiguous memory layout
- Memory pool so efficient memory allocation
- Vectorised math operations

Create a vectorized version using DR3

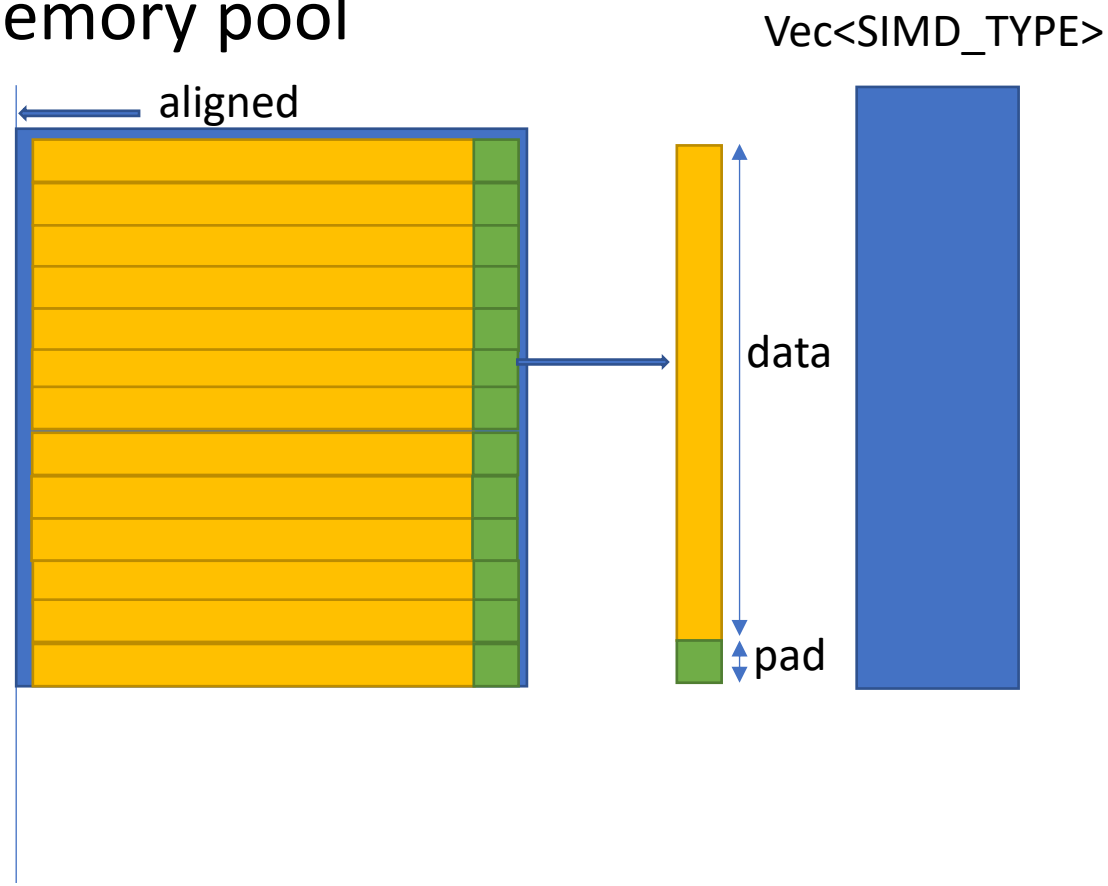
- Code available on https://github.com/andyD123/DR3
- Implementation using vectorised library.
- Using contiguous memory layout and vectorised instructions
- We can transform scalar code into vector code.
- Using auto to avoid explicitly indicating vector or scalar typing.

VecXX

- Memory managed vector type
- Supports math functions and operations
- Contiguous, aligned and padded
- Can change the scalar type and instruction set
- Substitutable for scalar type so we drop into existing code to make it vectorised

VecXX Utility

Memory pool



Math Operators and Functions

vec_A = vec_B+ vec_C

vec_A >= vec_B

vec_A = sin(vec_C)

Explicit β_0 and β_1

After computing the inverse, we get:

$$\beta_0 = \frac{(S_{xx} + \lambda)\, S_y - S_x\, S_{xy}}{\mathsf{N}(S_{xx} + \lambda) - (S_x)^2}, \qquad \beta_1 = \frac{\mathsf{N} S_{xy} - S_x\, S_y}{\mathsf{N}(S_{xx} + \lambda) - (S_x)^2}.$$

Note the denominators are the same !

```
auto MULT = [](auto x, auto y) { return x * y; };
auto SUM = [](auto x, auto y) { return x + y; };
auto SQR = [](auto x) {return x * x; };

//compute reductions for Sx,Sy,Sxx,Sxy
auto S_x = reduce(data_X, SUM);
auto S_y = reduce(data_Y, SUM);
auto S_xx = transformReduce(data_X, SQR, SUM);
auto S_xy = transformReduce(data_X, data_Y, MULT, SUM);

// compute leave one out vectors

auto SX_loo = S_x - data_X;   //leave one out SX
auto SY_loo = S_y - data_Y;   //leave one out SY

auto data_X_squared = data_X * data_X;
auto SXX_loo = S_xx - data_X_squared; //leave one out SXX

auto data_X_Y = data_X * data_Y;
auto SXY_loo = S_xy - data_X_Y;  //leave one out SXY

double lambda = 0.0;// 0.1;  //regularisation parameter
double Sz = data_X.size() - 1.0;

// Compute the fit parameters
auto denominator = (Sz * (SXX_loo + lambda)) - (SX_loo * SX_loo);

auto Beta_0_numerator = (SXX_loo + lambda) * SY_loo - SX_loo * SXY_loo;
auto Beta_0 = Beta_0_numerator / denominator; //vector of fits for Beta 0 offsets

auto Beta_1_numerator = Sz * SXY_loo - (SX_loo * SY_loo);
auto Beta_1 = Beta_1_numerator / denominator;  //vector of Beta_1   slopes
```

$$\beta_0 = \frac{(S_{xx} + \lambda)\, S_y - S_x\, S_{xy}}{\mathsf{N}(S_{xx} + \lambda) - (S_x)^2}$$

$$\beta_1 = \frac{\mathsf{N} S_{xy} - S_x\, S_y}{\mathsf{N}(S_{xx} + \lambda) - (S_x)^2}$$

Leave One Out Regression- Performance Run

- 1 million elements LOO

```
18.0961 milli seconds per fit
D:\online25\DR3\x64\ICC2023\LeaveOneOutRegression.exe (process 58192) exited with code 0.
Press any key to close this window . . .
```

```
generating data set size 1000000
setting data
fitting data
1.46681e+07 milli seconds per fit
D:\online25\DR3\x64\ICC2023\LeaveOneOutRegression.exe (process 79504) exited with code 0.
Press any key to close this window . . .
```

Blocking – Tiling

- Keeping all the vector sums in L1 cache

- Use some kind of Zip Iterator and lambda. Break the large vectors down into a sequence of small vectors that fit in L1 cache.

Using float or double

- Speed up x2
- Loss of accuracy ?

Use Pairwise Reduction

- Vectorized SIMD pairwise reduction can increase accuracy for large data sets

- Can help if we transition from double to float

Generalisation Reflection

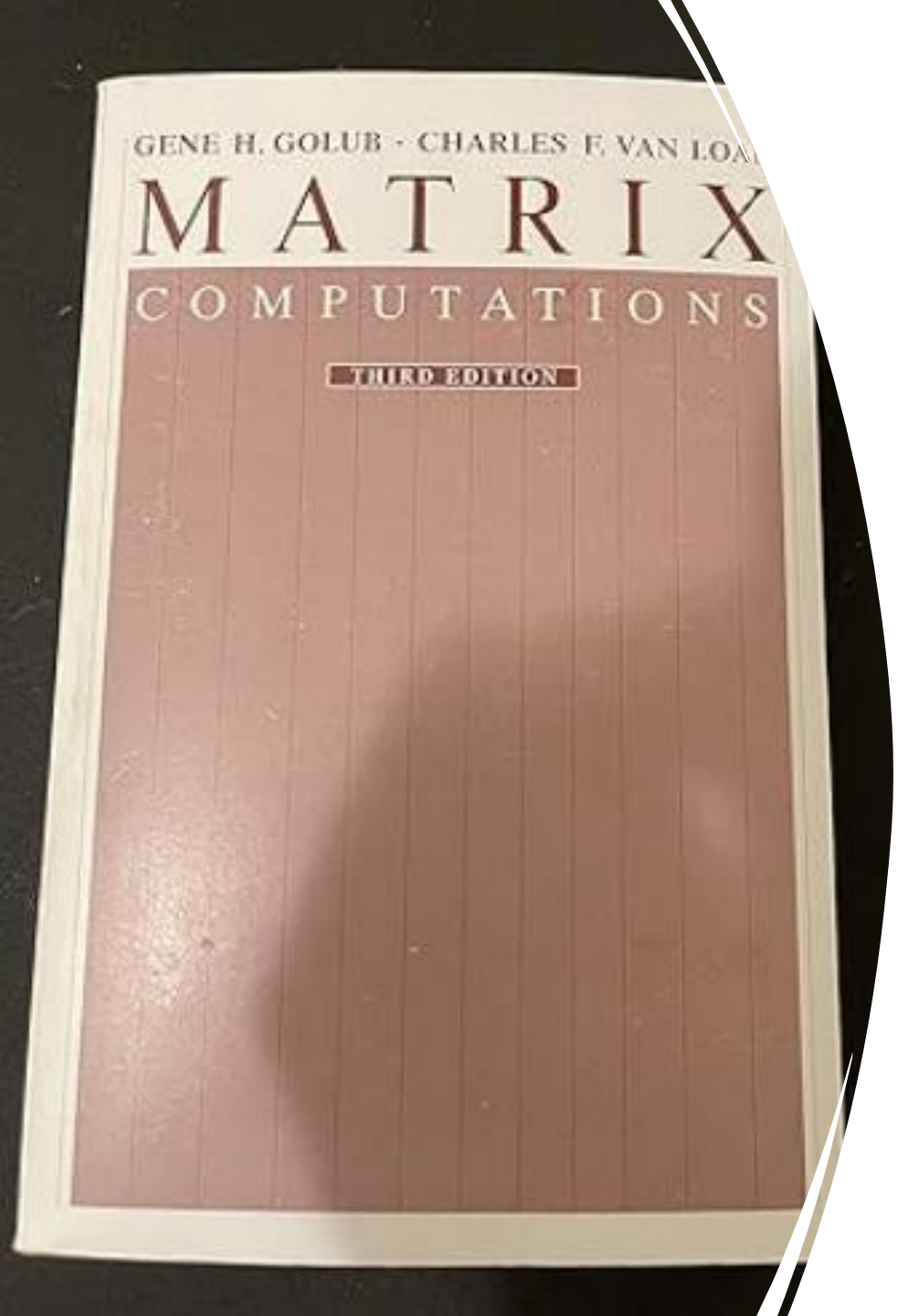
- We can generalise our Trick.
- If we are using reduction to calculate N values on perturbations of a set of data of size N. And there is an inverse operation to the reduction operation

- Generate O(N) perturbed sets
  - Generate Reduction value for each O(N)
  - -> $O(N^2)$

- Generate reduction on whole set O(N)
- Apply inverse operation on whole set result for each permutation O(1)
  - -> O(N)

Reflection

- This was an important illustration of the benefit of not just trying to optimise the inner loop.
- The benefits of expanding context
- Looking at a simpler, auxiliary sub-problem
- We might also look at calculating an example by hand



Reflection

- Had we studied some post-grad course in computational matrix methods
- We might have seen it as some matrix update problem

Take Aways

- Data-oriented design addresses a highly dimensional problem.
  - **Logical** algorithm design
  - **Physical** optimising spatiotemporal memory use patterns
  - **Idiosyncratic** aspects of the actual problem itself.

- Working through the problem space in a structured way by using approaches such as Polya's can help

- Always look to expand the context of your thinking around problem areas. This can be very useful in keeping brilliant solutions in play.

- Drawing pictures and solving easier ancillary problems is unlikely to be wasted time

Final Thoughts

- Hopefully you will find yourself in a situation where you have an increased understanding of the problem and afford yourself the time to innovate. With some success.

- code available on https://github.com/andyD123/DR3

- Contact e mail - andreedrakeford@hotmail.com

Appendix : Another example

Algorithm Reduction

- Our starting point is actually the end point of an excellent problem solving approach. Reduction

- By transforming inputs and outputs to a known problem and then apply the known solution and map solution back to problem domain.

- Black Scholes equation is used to price options on an asset price

- The average price is still a random variable and by transforming the moments we can apply it to pricing the option on the average price .

Example

- A troublesome function in a moment matching pricing algorithm:
- Sum over all elements of a square matrix, where the axis has sets of equal values and monotonic increasing time indices

Norm = $\sum_{i=0}^{N} \sum_{j=0}^{N} F(t, T1_i, T1_j)$

Example

- A troublesome function in a moment matching pricing algorithm:
- Sum over all elements of a square matrix, where the axis has sets of equal values and monotonic increasing time indices

Norm = $\sum_{i=0}^{N} \sum_{j=0}^{N} F(t, T1_i, T1_j)$

However inside of expensive function F, F_Impl just takes a single argument, of value $\min(t, T1_i, T1_j)$

What we calculate

Norm = $\sum_{i=0}^{N} \sum_{j=0}^{N} F\_impl(\min(t, T1_i, T1_j))$

This defines an N*N square matrix, with expensive to populate values

What would Polya say ? What are the good questions?

- Have you seen it before ?
- Tell me about this matrix,
- draw a picture
- Make it simpler only T1

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

What would Polya say ? What are the good questions?

- Have you seen it before?
- Tell me about this matrix,
- draw a picture
- Make it simpler only T1, T2
- Does it have any special properties or symmetries, Yes
- Values in T1 & T2 axes are the same
- How big is it (12*3, monthly time points going out to 3 years)
- However, we might need to consider hourly time points

What would Polya say ? What are the good questions?

- Values in T1 & T2 axes are the same, ordered in increasing value
- Do I get the same answer if I swap values of T1 and T2?
- Draw this on the matrix

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Exploiting

- Exploiting symmetry
- We calculate and re-use the result

$$R(i,j) = f(t,T1,T2)$$

$$R(j,i) = R(i,j)$$

All off axis calculations can be cut in half.

Or graphically we can see this as an upper triangular matrix

Draw Matrix, we don’t need to calculate values for these cells

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

What would Polya say ? What are the good questions?

- “What is the Condition?”

Polya questions

- What is the condition
- Have you used all the data

- “What is the Condition?”
- Thanks George P its this minimum

- Draw it ? . min(T1,T2)

- “What is the Condition?”
- Thanks George P its this minimum

- Draw it ? . min(T1,T2)

- Special case T1 =T2 on the diagonal
- Concrete example (3,3)   min is 3

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Special case T1 =T2 on the diagonal
Concrete example (3,3)
min is 3

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Special case T1 =T2 on the diagonal
Concrete example (3,3)
min is 3

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

What if we increase the column to 4 ie (3,4)?

min =3

Draw Matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

What if we increase the column to 5 ie (3,5)?

still min =3

Draw Matrix the min(T1,T2) condition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

What if we extend across to the end of the row? column to 7 ie (3,7)?

Yep we still have a min =3

Draw Matrix the min(T1,T2) condition

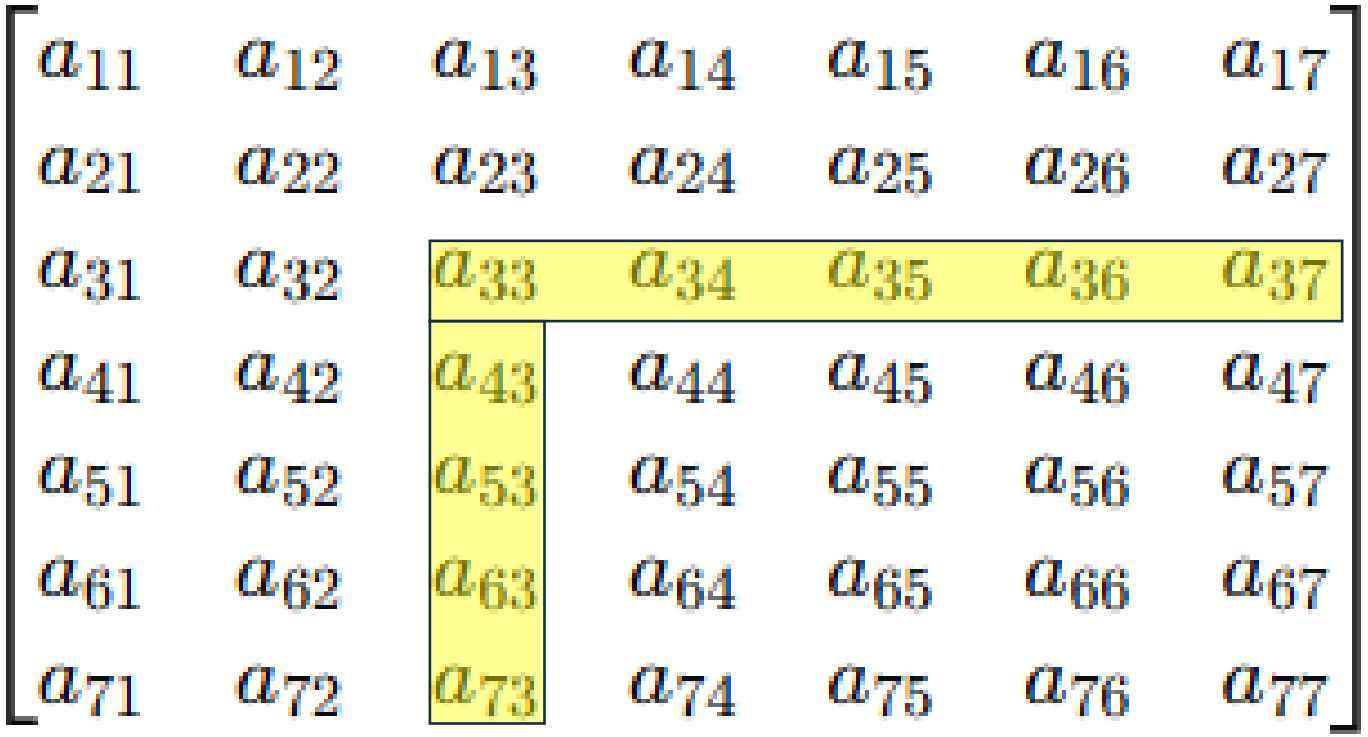
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

What if there is symmetry
What happens if we go down the column from the central diagonal

Point (4,3)

Yep we still have a min =3

Draw Matrix the min(T1,T2) condition



What if there is symmetry
What happens if we go down the column from the central diagonal

Point (4,3)
Point (5,3)
Point (6,3)
Point (7,3)

Yep we still have a min =3

Draw Matrix the min(T1,T2) condition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Does this Generalize?

Is this the same for every diagonal?

Draw Matrix the min(T1,T2) condition

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Does this Generalize?

Is this the same for every diagonal?

Top Row OK

Draw Matrix the min(T1,T2) condition

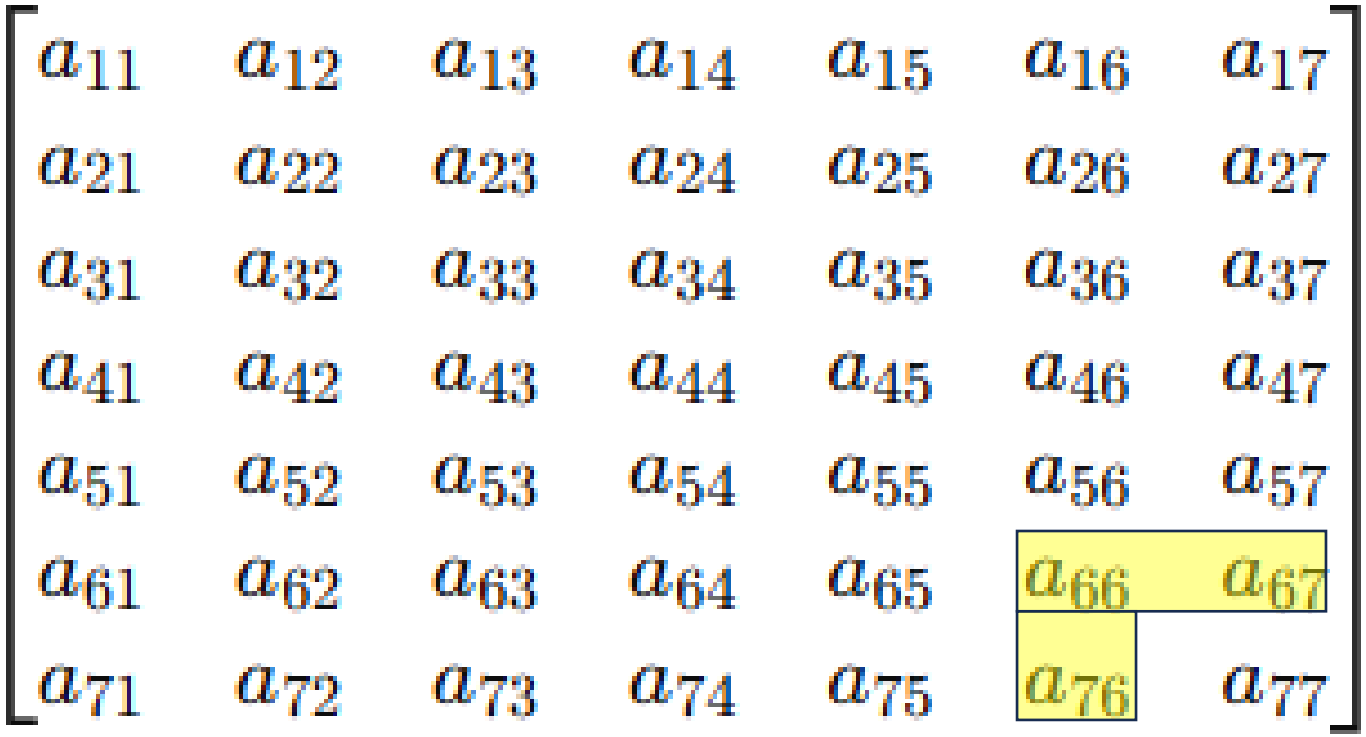
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Does this Generalize?

Is this the same for every diagonal?

Bottom Row OK

Draw Matrix the min(T1,T2) condition



Does this Generalize?

Is this the same for every diagonal?

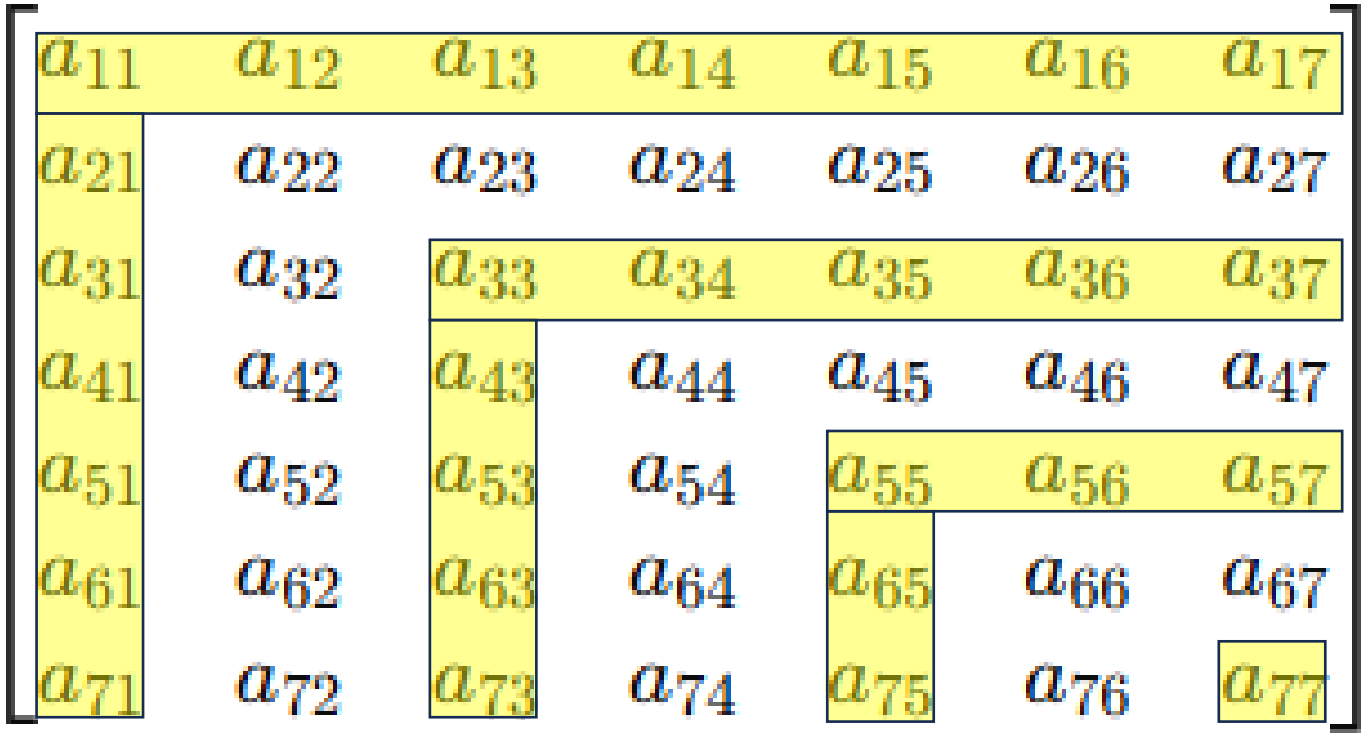
Even Row (6,6)

OK

- We have a collection of elements that have the same input value to the expensive function and will consequently generate the same value.
- We can apply the same logic to all other elements on the diagonal.

- We get N distinct regions 1 for each element on the diagonal.

Draw Matrix


a_{11} a_{12} a_{13} a_{14} a_{15} a_{16} a_{17}
a_{21} a_{22} a_{23} a_{24} a_{25} a_{26} a_{27}
a_{31} a_{32} a_{33} a_{34} a_{35} a_{36} a_{37}
a_{41} a_{42} a_{43} a_{44} a_{45} a_{46} a_{47}
a_{51} a_{52} a_{53} a_{54} a_{55} a_{56} a_{57}
a_{61} a_{62} a_{63} a_{64} a_{65} a_{66} a_{67}
a_{71} a_{72} a_{73} a_{74} a_{75} a_{76} a_{77}

- We have a collection of elements that have the same input value to the expensive function and will consequently generate the same value.
- We can apply the same logic to all other elements on the diagonal.

- We get N distinct regions 1 for each element on the diagonal.

- $size\ of\ ith\ element \quad S_i = 2(N - i)+1$

Reduces to a single summation
O(N^2) -> O(N)

$$Norm = \sum_{i=1}^{N} F(Ti) * (2(N - i) + 1)$$

- Walk along the diagonal elements call the function and multiply by scale factor (number of elements) and add to running sum

Adding the extra variable t, to the min condition

- Three variants
- **t < T1**
- **t > TN**
- **T1 < t < TN**

Draw Matrix adding extra t min(t,min(Ti,Tj))

t < T1

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

Elements follow the angled partition scheme

All elements = t

$$Norm = F(t) * N^2$$

Draw Matrix adding extra t min(t,min(T1,T2))

t > TN

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} \end{bmatrix}$$

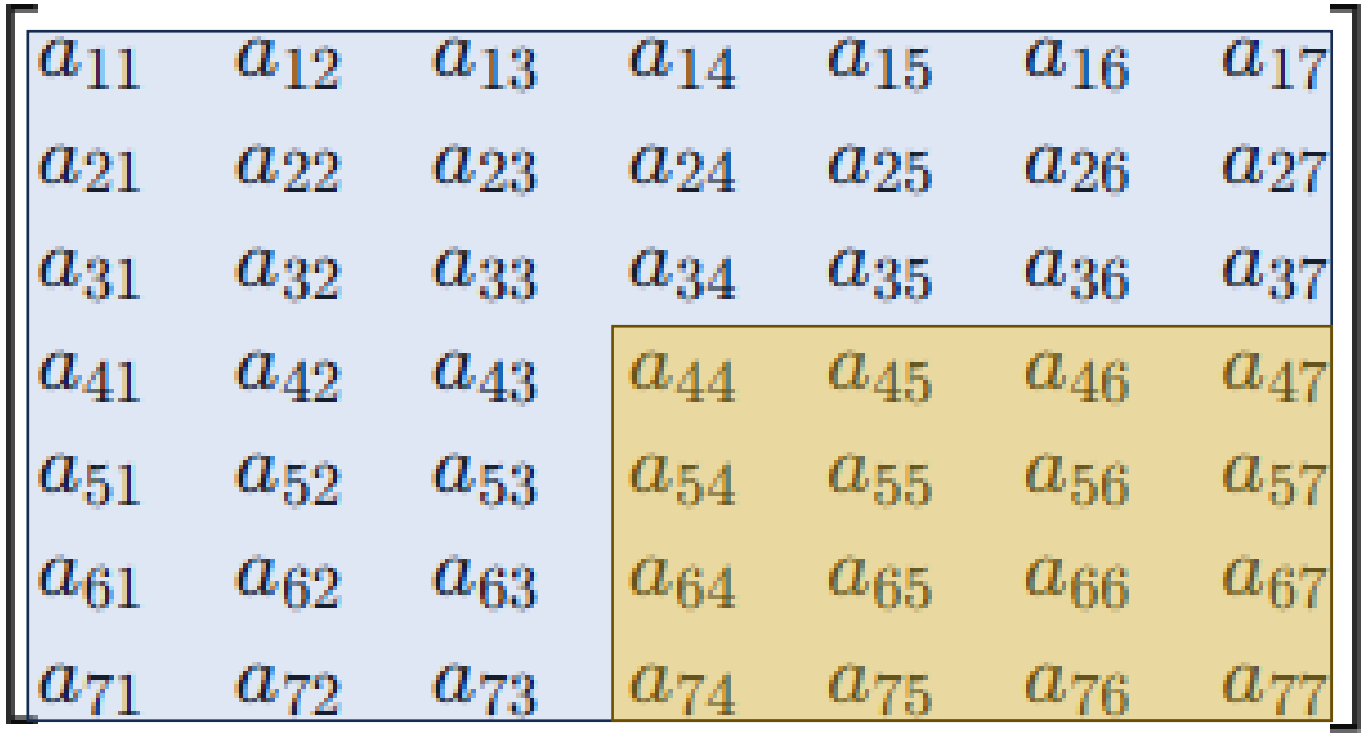
Elements follow the angled partition scheme

All elements = t

$$Norm = \sum_{i=1} F(Ti) * (2(N - i) + 1)$$

Draw Matrix adding extra t min(t,min(T1,T2))

T1 < t < TN



Elements follow the angled partition scheme

All elements = t

$$Norm = \sum_{i=1}^{K} F(Ti) * (2(N-i)+1) \; + F(t) * (N-K)^2$$

Result

- Our new approach gives us between 1 and N calls to the expensive function, as opposed to N^2

- Strategic win, particularly when we move to assets which have matrix elements on an hourly basis instead of monthly

Reflection

- We looked for and found special conditions that helped us.
- We took an unusual path through the elements of our matrix. (not just linear) and ultimately, we lost a dimension (inner loop)
- Finding these conditions and exploiting them
- Drawing the diagram was particularly helpful to me
- Considering the even simpler case and the special cases