Week 3: Matrices: manipulation and computation

Algorithmic Data Science

2025-26

Dr Adam Barrett

A puzzle for you

- The probability of it being sunny is 2/9
- The probability of it raining is 4/9
- The probability of it being overcast is 1/3
- The probability that I go to the park if it is sunny is 3/4
- The probability that I go to the park if it is raining is ½
- The probability that I go to the park if it is overcast is 2/3

- What is the probability that I go to the park?

- The probability of it being sunny is 2/9
- The probability of it raining is 4/9
- The probability of it being overcast is 1/3
- The probability that I go to the park if it is sunny is 3/4
- The probability that I go to the park if it is raining is ½
- The probability that I go to the park if it is overcast is 2/3

- What is the probability that I go to the park?

$$P(p) = P(p|s)P(s) + P(p|r)P(r) + P(p|o)P(o)$$

$$= \frac{1}{6} \qquad + \frac{2}{9} \qquad + \frac{2}{9} \qquad = \frac{11}{18}$$

This session

- Empirical computation of time complexity.

- Another set of notes on the elementary matrix operations.
- An application of matrices
- Algorithms for matrix multiplication

Time complexity O Notation: Loose definition.

A run-time is **O(g(n))** if:

For sufficiently large data, **const x g(n)** approximates the run-time well,

and the approximation generally gets better and better the bigger the data.

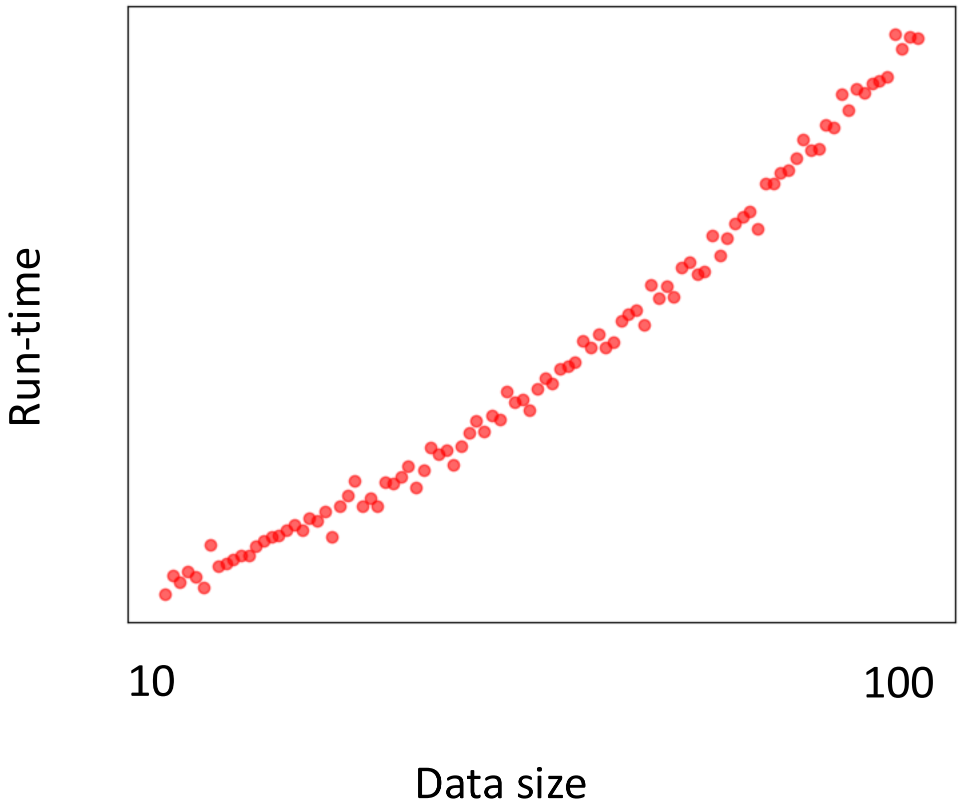
Example: $17n^6 + 15n^4$ is $\boldsymbol{O(n^6)}$

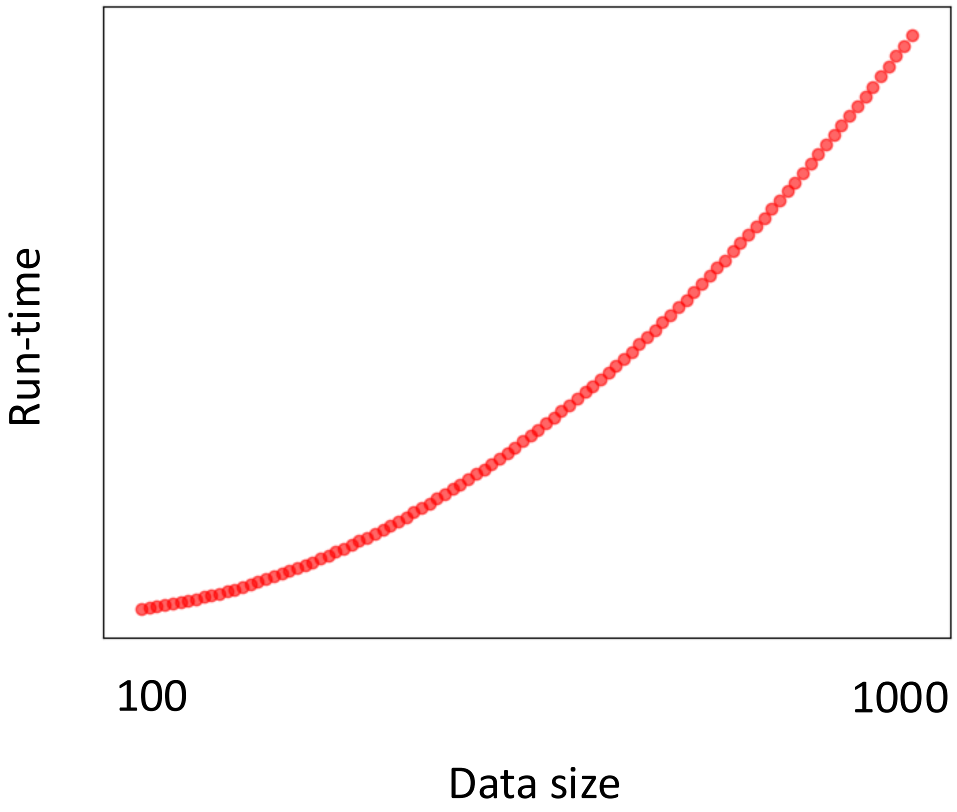
Here g(n) is n^6 and the constant is 17:

The bigger n gets,
the better $17n^6$ = 17g(n) approximates
the run-time.

Empirically testing the run-time

- In the labs you will plot run-time against data size.
- Here is one that for small n looks maybe linear, but for larger n, we see that it is not-linear.





- If we assume this is $O(n^{\alpha})$, how do we find α?

Empirically testing the run-time

If t is $O(n^{\alpha})$ then the run-time is the following, for some constant c:

$$t = cn^{\alpha}$$

Take logs of both sides of the equation:

$$\log(t) = \log(cn^{\alpha})$$

Using log(xy)=log(x)+log(y)
And $\log(a^b) = b \log(a)$

$$\log(t) = \log(c) + \alpha\log(n)$$

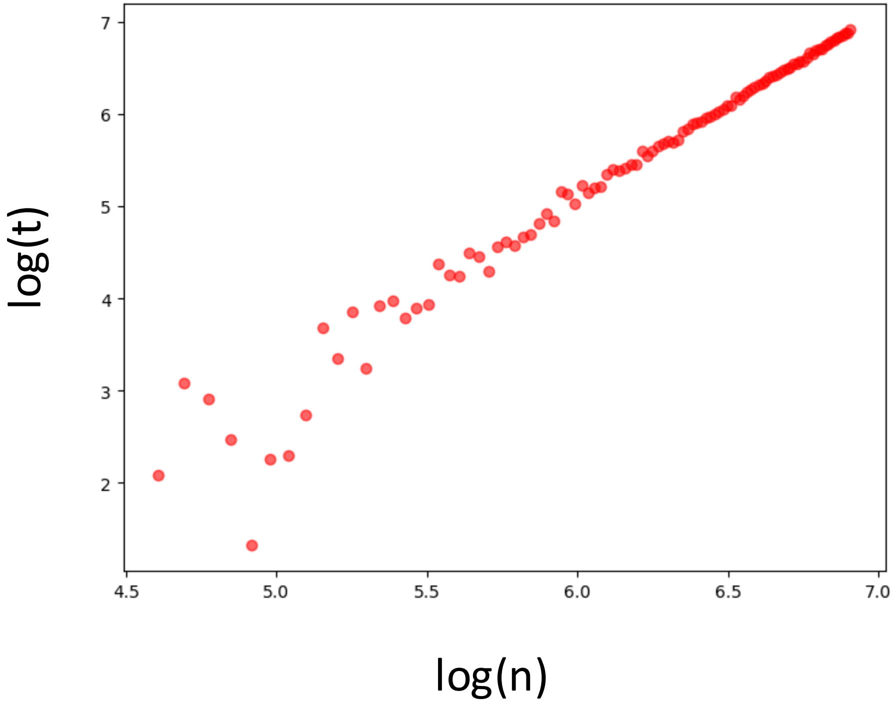
This means that if I plot y=log(t) against x=log(n), I will get a straight line, with α being the gradient (slope) and $\log(c)$ being the intercept:

$$\mathrm{y} = \log(c) + \alpha x$$

You can then use a stats library to find the gradient and intercept and hence α and the constant.

Example

Doing this for the example:

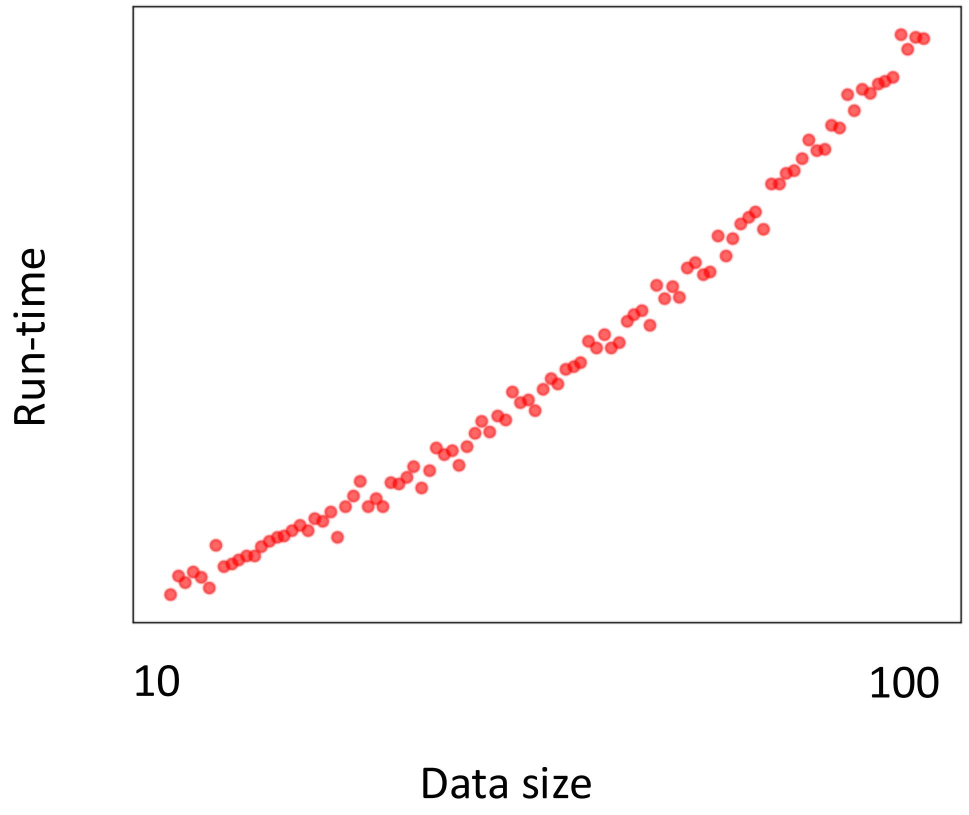


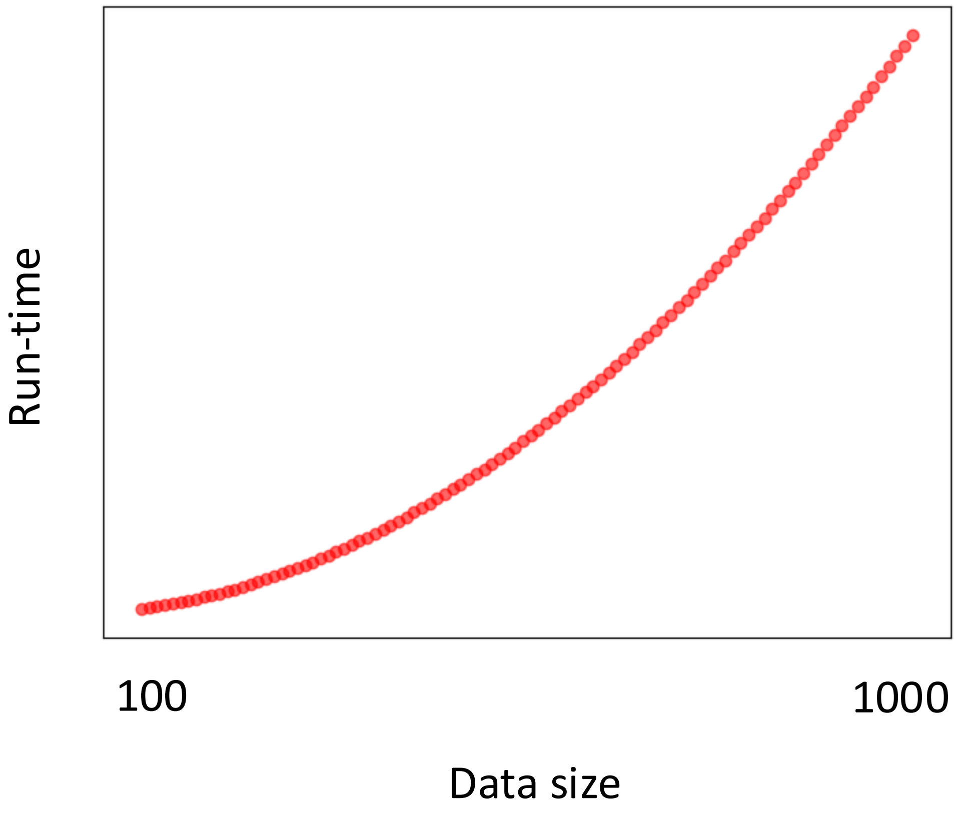
Use a stats library to find the slope and the intercept.

Can tell by eye that the slope is roughly 2 for large n, which is what we're interested in. So this is $O(n^2)$.

Empirically testing the run-time

- In the labs you will plot run-time against data size.
- Here is one that for small n looks maybe linear, but for larger n, we see that it is not-linear.





- If we assume this is $O(n^{\alpha})$, how do we find α?

Empirically testing the run-time

If t is $O(n^{\alpha})$ then the run-time is the following, for some constant c:

$$t = cn^{\alpha}$$

Take logs of both sides of the equation:

$$\log(t) = \log(cn^{\alpha})$$

Using log(xy)=log(x)+log(y)
And $\log(a^{b}) = b\log(a)$

$$\log(t) = \log(c) + \alpha\log(n)$$

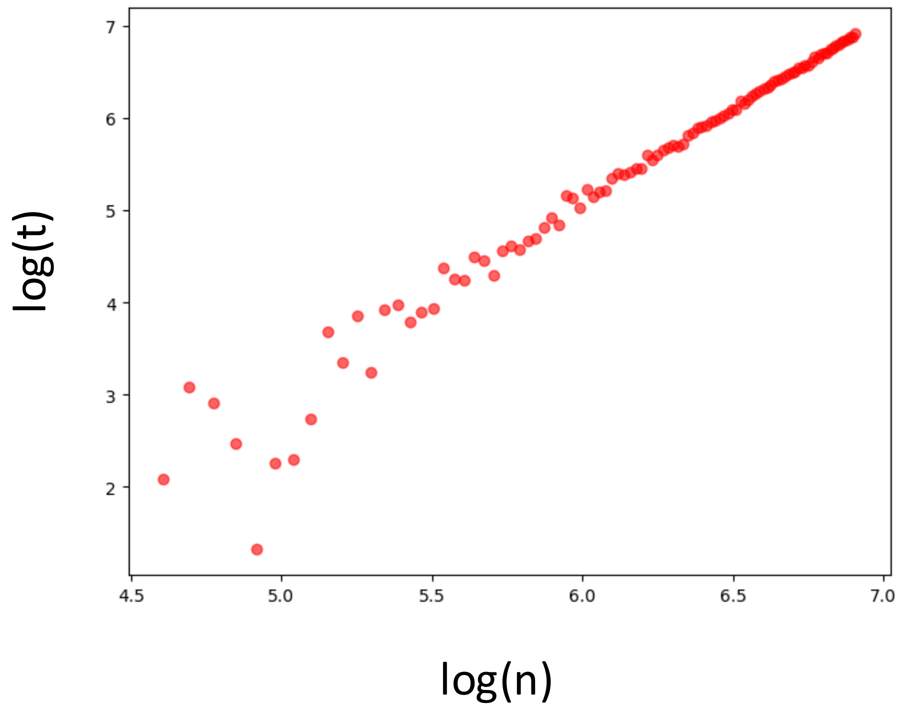
This means that if I plot y=log(t) against x=log(n), I will get a straight line, with α being the gradient (slope) and $\log(c)$ being the intercept:

$$\mathrm{y} = \log(c) + \alpha x$$

You can then use a stats library to find the gradient and intercept and hence α and the constant.

Example

Doing this for the example:



Use a stats library to find the slope and the intercept.

Can tell by eye that the slope is roughly 2 for large n, which is what we're interested in. So this is $O(n^2)$.

Main topics per week

Week	Topic
1	Data structures and data formats
2	Algorithmic complexity. Sorting.
3	**Matrices: Manipulation and computation**
4	Similarity analysis
5	Processes and concurrency
6	Distributed computation
7	Map/reduce
8	Graphs/networks
9	Graphs/networks, PageRank algorithm
10	Databases
11	*independent study*

A matrix is ….

- a structured collection of numbers, e.g.,

$$A = \begin{pmatrix} 0 & 3 \\ -2 & 5 \\ 0.2 & 10 \end{pmatrix}$$

- matrix *A* has 3 rows and 2 columns. Its **dimensionality** is 3x2
- Individual elements, a_{ij}, can be referred to by subscripts.
- *i* refers to the row
- *j* refers to the column
- So here, a_{21} = -2

Matrix terminology

- A **vector** is a 1 dimensional matrix (dimensionality = 1xn or nx1)
- A **row vector** is 1xn whereas a **column vector** is nx1
- A **zero matrix** is a matrix where every entry is 0
- A **square** matrix has dimensionality *n*x*n*
- A **diagonal** matrix is a square matrix with a_{ij} =0 if i≠j
- An **identity** matrix, *I*, is a diagonal matrix with a_{ij} = 1 if i=j

- Let's see an example of each of these.

- A **row vector:** (5 2 3)
- A **column vector:** $\begin{pmatrix} 1 \\ 4 \\ 2 \end{pmatrix}$
- A **zero matrix** $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
- A **square** matrix $\begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix}$
- A **diagonal** matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{pmatrix}$
- An **identity** matrix, *I* $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Matrix operations: transpose

- The transpose of a matrix A is the matrix A^T obtained by exchanging the rows and columns of A.

$$A = \begin{pmatrix} 0 & 3 \\ -2 & 5 \\ 0.2 & 10 \end{pmatrix} \rightarrow A^T = \begin{pmatrix} 0 & -2 & 0.2 \\ 3 & 5 & 10 \end{pmatrix}$$

- A **symmetric** matrix satisfies the condition A = A^T

- Write down a symmetric matrix. $\begin{pmatrix} 1 & 4 \\ 4 & 2 \end{pmatrix}$

Matrix operations: addition

- Addition can only be carried out for matrices which have the same dimensions
- Addition is defined component-wise:

$$C = A + B \leftrightarrow \forall_{ij}\left(c_{ij} = a_{ij} + b_{ij}\right)$$

- For example:

$$\begin{pmatrix} 1 & 3 & 1 \\ 1 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+0 & 1+5 \\ 1+7 & 0+5 & 0+0 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 6 \\ 8 & 5 & 0 \end{pmatrix}$$

- For an m x n matrix, what is the asymptotic run-time of matrix addition in O notation?

$$O(m\, n)$$

Matrix operations: multiplication by a scalar

- If λ is a scalar and A = (a_{ij}) is a matrix, then λA = (λ a_{ij}) is the scalar multiple of A obtained by multiplying each of its elements by λ.
- The **negative** of a matrix is defined as $-A = -1.A$
- Hence $-A = (-a_{ij})$
- Hence, $A+(-A) = 0 = (-A) + A$

- For an n x n matrix, what is the asymptotic run-time of multiplication by a scalar?

$$O(n^2)$$

Matrix operations: matrix subtraction

- **matrix subtraction** is defined as the addition of the negative of a matrix: *B*-*A* = *B*+(-*A*)

1. $\begin{pmatrix} 10 & 0 \\ 3 & -2 \\ 5 & -9 \end{pmatrix} - \begin{pmatrix} 0 & 8 \\ -2 & -2 \\ 5 & 1 \end{pmatrix} =$ $\begin{pmatrix} 10 & -8 \\ 5 & 0 \\ 0 & -10 \end{pmatrix}$

2. $\begin{pmatrix} 2 & -1 \\ 5 & 3 \end{pmatrix} - \begin{pmatrix} 0 & 5 & 2 \\ 3 & 1 & -5 \end{pmatrix}$= Doesn't exist (not compatible)

3. $4\begin{pmatrix} 3 & -1 \\ 0 & 2 \end{pmatrix} - 2I =$ $\begin{pmatrix} 10 & -4 \\ 0 & 6 \end{pmatrix}$

Matrix operations: matrix multiplication

- Two matrices, *A* and *B*, can only be multiplied if they are **compatible**: the number of columns of *A* equals the number of rows of *B*.
- If *A* = (a_{ij}) is an *mxn* matrix and *B* = (b_{jk}) is an *nxp* matrix, then their matrix product *C* = *AB* is the *mxp* matrix *C* = (c_{ik}) where: $$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$
- For example: $\begin{pmatrix} 2 & 0 & 1 \\ 0 & -1 & 3 \end{pmatrix} \times \begin{pmatrix} 3 & -2 \\ 1 & 5 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 6 & -4 \\ -1 & -5 \end{pmatrix}$
- Note that matrix multiplication is **not commutative**: $\begin{pmatrix} 3 & -2 \\ 1 & 5 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 & 1 \\ 0 & -1 & 3 \end{pmatrix} = \begin{pmatrix} 6 & 2 & -3 \\ 2 & -5 & 16 \\ 0 & 0 & 0 \end{pmatrix}$
- For n x n matrices, what is the asymptotic run-time of (naïve method) matrix multiplication?

$$O(n^3)$$

Identity matrix is multiplicative identity

- Check for yourselves by example, that for any square matrix *A*, and identity matrix *I* of the same dimensions as *A*,

$$AI = IA = A.$$

- *Exercise: Can you prove this in general, using algebra?*

$$A = \begin{pmatrix} 5 & 6 \\ 1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Matrix operations: matrix division?

- How do you find B such that $AB = C$?
- There is no ‘division’ operator for matrices.
- However, we define the **inverse** of an nxn matrix A to be the nxn matrix, denoted A^{-1} (if it exists), such that $AA^{-1} = I = A^{-1}A$
- Hence, in the above example where $AB = C$, it follows that $B = A^{-1}C$

Matrix inverses

- We can test whether *B* is the inverse of *A* using matrix multiplication e.g.,

$$AB = \begin{pmatrix} 4 & 7 \\ 2 & 6 \end{pmatrix}\begin{pmatrix} 0.6 & -0.7 \\ -0.2 & 0.4 \end{pmatrix} = \begin{pmatrix} 4 \times 0.6 + 7 \times -0.2 & 4 \times -0.7 + 7 \times 0.4 \\ 2 \times 0.6 + 6 \times -0.2 & 2 \times -0.7 + 6 \times 0.4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

- So $A = B^{-1}$ and $B = A^{-1}$
- Many nonzero square matrices do not have inverses. A matrix without an inverse is called **noninvertible** or **singular**.
- If a matrix has an inverse, it is called **invertible** or **non-singular.**
- The transpose operation commutes with the inverse operation:

$$(A^{-1})^T = (A^T)^{-1}$$

- Test this for yourself.

Finding inverses: 2x2 matrices

1. Find the determinant. For a 2x2 matrix A this is:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

2. If (and only if) the determinant is 0 (which it will be if any row or column contains only 0's), then A is singular. Otherwise:

$$A^{-1} = \frac{1}{|A|}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

3. It is straightforward to prove that this is the inverse for a 2x2 matrix.

Example:

$$A = \begin{pmatrix} 4 & 3 \\ 1 & 2 \end{pmatrix}$$

Finding inverses: larger matrices

- The *ij*th **minor** of an *n*x*n* matrix *A*, for *n>1*, is the *(n-1)x(n-1*) matrix $A_{[ij]}$ obtained by deleting the *i*th row and the *j*th column of *A.*
- The determinant of A is given by the recursive procedure:

$$|A| = \begin{cases} a_{11} \text{ if } \quad n = 1 \\ a_{11}|A_{[11]}| - a_{12}|A_{[12]}| + \cdots + (-1)^{n+1} a_{1n}|A_{[1n]}| \quad \text{if } n > 1 \end{cases}$$

- Make a matrix where each element is replaced by the determinant of its minor
- Change the signs of alternate cells (this is called the matrix of cofactors)
- Transpose this matrix (this is called the adjugate or adjoint)
- Multiply by the reciprocal of the determinant.

Example

$$\begin{vmatrix} 4 & 3 & 5 \\ 1 & 2 & 3 \\ 4 & 1 & 2 \end{vmatrix} = 4\begin{vmatrix} 2 & 3 \\ 1 & 2 \end{vmatrix} - 3\begin{vmatrix} 1 & 3 \\ 4 & 2 \end{vmatrix} + 5\begin{vmatrix} 1 & 2 \\ 4 & 1 \end{vmatrix}$$

= 4x(2x2-3x1)-3x(1x2-3x4)+5x(1x1-2x4)

=4+30-35=-1

- For an n x n matrix, what is the asymptotic run-time of naïve computation of the determinant?

$O(n^2)$
$O(n^3)$
$O(n!)$ ✓
$O(2^n)$

Applications of matrices: solving systems of linear equations

Imagine we have a set of 3 simultaneous linear equations:

$$3x + 2y \; - z = 10$$
$$-x + 5y \; - 3z = -2$$
$$2x \; - y + 2z = 0$$

This can be written as a matrix equation:

$$\begin{pmatrix} 3 & 2 & -1 \\ -1 & 5 & -3 \\ 2 & -1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 10 \\ -2 \\ 0 \end{pmatrix}$$

Therefore the solution can be found (if there is one) by calculating:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 3 & 2 & -1 \\ -1 & 5 & -3 \\ 2 & -1 & 2 \end{pmatrix}^{-1} \begin{pmatrix} 10 \\ -2 \\ 0 \end{pmatrix}$$

Applications of matrices: calculating marginal distributions

The example from the beginning of the lecture can be written:

$$\begin{pmatrix} P(park|sunny) & P(park|raining) & P(park|overcast) \end{pmatrix} \begin{pmatrix} P(sunny) \\ P(raining) \\ P(overcast) \end{pmatrix} = \begin{pmatrix} P(park) \end{pmatrix}$$

We can easily extend this:

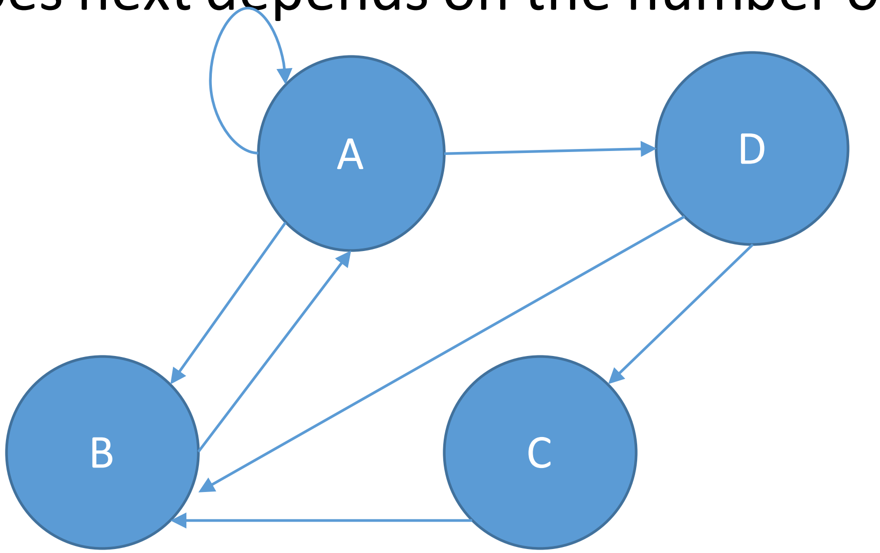
This is **stochastic** if each column sums to 1, i.e., represents a complete probability distribution over the variables (variables must be mutually exclusive and exhaustive)

$$\begin{pmatrix} P(p|s) & P(p|r) & P(p|o) \\ P(b|s) & P(b|r) & P(b|o) \\ P(h|s) & P(h|r) & P(h|o) \end{pmatrix} \begin{pmatrix} P(s) \\ P(r) \\ P(o) \end{pmatrix} = \begin{pmatrix} P(p) \\ P(b) \\ P(h) \end{pmatrix}$$

$$\begin{pmatrix} 3/4 & 1/2 & 2/3 \\ 1/8 & 0 & 1/9 \\ 1/8 & 1/2 & 2/9 \end{pmatrix} \begin{pmatrix} 2/9 \\ 4/9 \\ 1/3 \end{pmatrix} = \begin{pmatrix} 66/108 \\ 7/108 \\ 35/108 \end{pmatrix}$$

The PageRank Algorithm

- Ranks pages on the web by their perceived importance
- Pages are considered more important if they have more links TO them from other more important pages ….
- Imagine a random surfer on a web with 4 pages. If he is truly random, then there is a uniform probability of him starting anywhere. The probability of where he goes next depends on the number of outlinks from a page



At time 0, t_0: $P(A) = P(B) = P(C) = P(D) = 1/4$

At time 1, t_1:

$P(A \mid A_0) = 1/3$	$P(A \mid B_0) = 1$	
$P(B \mid A_0) = 1/3$	$P(B \mid B_0) = 0$	….
$P(C \mid A_0) = 0$	$P(C \mid B_0) = 0$	
$P(D \mid A_0) = 1/3$	$P(D \mid B_0) = 0$	

The PageRank Algorithm

Transition matrix: T

At t_1:

$$\begin{pmatrix} P(A) \\ P(B) \\ P(C) \\ P(D) \end{pmatrix} = \begin{pmatrix} 1/3 & 1 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} = \begin{pmatrix} 8/24 \\ 11/24 \\ 3/24 \\ 2/24 \end{pmatrix}$$

At t_n:

$$\begin{pmatrix} P(A) \\ P(B) \\ P(C) \\ P(D) \end{pmatrix} = T^n \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$$

- This tells us where the random surfer is likely to be after n steps.
- So matrix-vector multiplication is the foundation of the PageRank algorithm, named after Larry Page, one of the co-founders of Google.
- Without it, we would still be using Yahoo, Altavista and other search engines which have now all but vanished ....
- And the matrices are very large at Google ....

Addition and multiplication, simple exercise

$$\begin{pmatrix} a & 9 \\ 4 & 7 \end{pmatrix} + \begin{pmatrix} 5 & 3 \\ 4 & 2 \end{pmatrix} = \begin{pmatrix} & \\ & \end{pmatrix}$$

$$\begin{pmatrix} 4 & 7 \\ 2 & 6 \end{pmatrix} \begin{pmatrix} 3 & 1 \\ b & 2 \end{pmatrix} = \begin{pmatrix} & \\ & \end{pmatrix}$$

Algorithms for matrix multiplication: naïve method

```
Matrix-Multiply (A,B):
  if A and B are nxn matrices:
    let C be an nxn matrix
    for i from 1 to n:
            for j from 1 to n:
          c_ij = 0
          for k from 1 to n:
            c_ij += a_ik * b_kj
  return C
```

The number of multiplications is n^3.
The number of additions is n^3

So it is straightforward to see that an upper bound on the running time of this algorithm is $O(n^3)$

Strassen's Method

First we note that any multiplication of 2 *n*x*n* matrices where n is a power of 2 can be broken down recursively into the multiplication of (*n/2*) x (*n/2*) matrices.

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & \cdot\cdot & \cdot\cdot \\ k & l & \cdot\cdot & \cdot\cdot \end{pmatrix}\begin{pmatrix} \alpha & \beta & \gamma & \delta \\ \varepsilon & \zeta & \eta & \theta \\ \lambda & \mu & \cdot\cdot & \cdot\cdot \\ \nu & \rho & \cdot\cdot & \cdot\cdot \end{pmatrix} = \begin{pmatrix} a\alpha + b\varepsilon + c\lambda + d\nu & a\beta + b\zeta + c\mu + d\rho & \cdot\cdot & \cdot\cdot \\ e\alpha + f\varepsilon + g\lambda + h\nu & e\beta + f\zeta + g\mu + h\rho & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \end{pmatrix}$$

This can alternatively be written as:

$$\begin{pmatrix} \begin{pmatrix} a & b \\ e & f \end{pmatrix} & \begin{pmatrix} c & d \\ g & h \end{pmatrix} \\ \begin{pmatrix} i & j \\ k & l \end{pmatrix} & \begin{pmatrix} \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \cdot\cdot \end{pmatrix} \end{pmatrix}\begin{pmatrix} \begin{pmatrix} \alpha & \beta \\ \varepsilon & \zeta \end{pmatrix} & \begin{pmatrix} \gamma & \delta \\ \eta & \theta \end{pmatrix} \\ \begin{pmatrix} \lambda & \mu \\ \nu & \rho \end{pmatrix} & \begin{pmatrix} \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \cdot\cdot \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} a & b \\ e & f \end{pmatrix}\begin{pmatrix} \alpha & \beta \\ \varepsilon & \zeta \end{pmatrix} + \begin{pmatrix} c & d \\ g & h \end{pmatrix}\begin{pmatrix} \lambda & \mu \\ \nu & \rho \end{pmatrix} & \cdot\cdot \\ \cdot\cdot & \cdot\cdot \end{pmatrix}$$

Strassen’s Method

So for any *nxn* matrix, where *n* is a power of 2, it is straightforward to write matrix multiplication as a recurrence, where the components of the matrices may be numbers or matrices:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & g \\ f & h \end{pmatrix} = \begin{pmatrix} r & s \\ t & u \end{pmatrix}$$

$$r = ae + bf$$
$$s = ag + bh$$
$$t = ce + df$$
$$u = cg + dh$$

This is an example of a divide-and-conquer strategy. We split the problem into smaller problems, solve each smaller problem and then combine the results. Here the smaller problem is of size n/2. There are 2^3 of them to solve. Combining requires 4 matrix additions.

Strassen's method

- The running time for this basic recursive approach is given by solving the recurrence formula:

Since matrix addition is $O(n^2)$. Doing 4 of them only affects the constant.

$$T(n) = 8T\left({}^{n}/_{2}\right) + O(n^2)$$

- The solution of this recurrence formula is: $T(n) = O\left(8^{log_2 n}\right) = O\left(n^{log_2 8}\right) = O(n^3)$
- This is no faster than the naïve method for matrix multiplication
- However Strassen discovered a recursive method which requires only 7 recursive multiplications at each step (but many more additions and subtractions)

$$T(n) = 7T\left({}^{n}/_{2}\right) + O(n^2) = O\left(n^{log_2 7}\right)$$

Strassen’s Method

Remember:
$r = ae + bf$
$s = ag + bh$
$t = ce + df$
$u = cg + dh$

Calculate (recursively):
$P_1 = a(g - h)$
$P_2 = (a + b)h$
$P_3 = (c + d)\, e$
$P_4 = d(f - e)$
$P_5 = (a + d)(e + h)$
$P_6 = (b - d)(f + h)$
$P_7 = (a - c)(e + g)$

These are equivalent to:
$P_1 = ag - ah$
$P_2 = ah + bh$
$P_3 = ce + de$
$P_4 = df - de$
$P_5 = ae + ah + de + dh$
$P_6 = bf + bh - df - dh$
$P_7 = ae + ag - ce - cg$

7 multiplications and 10 additions/subtractions

Strassen's Method

Remember:
$r = ae + bf$
$s = ag + bh$
$t = ce + df$
$u = cg + dh$

Calculate (recursively):
$P_1 = a(g - h)$
$P_2 = (a + b)h$
$P_3 = (c + d)\, e$
$P_4 = d(f - e)$
$P_5 = (a + d)(e + h)$
$P_6 = (b - d)(f + h)$
$P_7 = (a - c)(e + g)$

These are equivalent to:
$P_1 = ag - ah$
$P_2 = ah + bh$
$P_3 = ce + de$
$P_4 = df - de$
$P_5 = ae + ah + de + dh$
$P_6 = bf + bh - df - dh$
$P_7 = ae + ag - ce - cg$

$$\underbrace{P_1 + P_2}_{\boxed{\text{1 addition}}} = ag - \cancel{ah} + \cancel{ah} + bh = s$$

Strassen's Method

Remember:
$r = ae + bf$
$s = ag + bh$
$t = ce + df$
$u = cg + dh$

Calculate (recursively):
$P_1 = a(g - h)$
$P_2 = (a + b)h$
$P_3 = (c + d)\, e$
$P_4 = d(f - e)$
$P_5 = (a + d)(e + h)$
$P_6 = (b - d)(f + h)$
$P_7 = (a - c)(e + g)$

These are equivalent to:
$P_1 = ag - ah$
$P_2 = ah + bh$
$P_3 = ce + de$
$P_4 = df - de$
$P_5 = ae + ah + de + dh$
$P_6 = bf + bh - df - dh$
$P_7 = ae + ag - ce - cg$

$$\underbrace{P_3 + P_4}_{\boxed{\text{1 addition}}} = ce + \cancel{de} + df - \cancel{de} = t$$

Strassen's Method

Remember:

$r = ae + bf$

$s = ag + bh$

$t = ce + df$

$u = cg + dh$

Calculate (recursively):

$P_1 = a(g - h)$

$P_2 = (a + b)h$

$P_3 = (c + d)\, e$

$P_4 = d(f - e)$

$P_5 = (a + d)(e + h)$

$P_6 = (b - d)(f + h)$

$P_7 = (a - c)(e + g)$

These are equivalent to:

$P_1 = ag - ah$

$P_2 = ah + bh$

$P_3 = ce + de$

$P_4 = df - de$

$P_5 = ae + ah + de + dh$

$P_6 = bf + bh - df - dh$

$P_7 = ae + ag - ce - cg$

$$\underbrace{P_5 + P_4 - P_2 + P_6}_{\text{3 additions / subtractions}} = ae + \cancel{ah} + \cancel{de} + \cancel{dh} + \cancel{df} - \cancel{de} - \cancel{ah} - \cancel{bh} + bf + \cancel{bh} - \cancel{df} - \cancel{dh} = r$$

Strassen's Method

Remember:

$r = ae + bf$

$s = ag + bh$

$t = ce + df$

$u = cg + dh$

Calculate (recursively):

$P_1 = a(g - h)$

$P_2 = (a + b)h$

$P_3 = (c + d)\, e$

$P_4 = d(f - e)$

$P_5 = (a + d)(e + h)$

$P_6 = (b - d)(f + h)$

$P_7 = (a - c)(e + g)$

These are equivalent to:

$P_1 = ag - ah$

$P_2 = ah + bh$

$P_3 = ce + de$

$P_4 = df - de$

$P_5 = ae + ah + de + dh$

$P_6 = bf + bh - df - dh$

$P_7 = ae + ag - ce - cg$

$$\underbrace{P_5 - P_3 - P_7 + P_1}_{\boxed{\text{3 additions / subtractions}}} = \cancel{ae} + \cancel{ah} + \cancel{de} + dh - \cancel{ce} - \cancel{de} - \cancel{ae} - \cancel{ag} + \cancel{ce} + cg + \cancel{ag} - \cancel{ah} = u$$

Strassen's Method

- So we can carry out matrix multiplication with just 7 recursive multiplications of matrices size *n/2* (but we now have 18 additions / subtractions rather than 4). So:

$$T(n) = 7T\left({}^{n}/_{2}\right) + O(n^2) = O\left(n^{log_2 7}\right) = \mathrm{O}(n^{2.81})$$

- It is possible to modify Strassen's algorithm to work when n is not a power of 2
- In practice, the large constant hidden in the running time makes Strassen's algorithm impractical unless n is large (>45) and dense (few zero entries).
- For sparse matrices, there are special sparse-matrix algorithms which can beat this.
- There are even more advanced techniques which can beat Strassen for dense matrices - $O(n^{2.376})$ is achievable, maybe even better.

Preview of an application: All Pairs Similarity

- If A is an *nxm* matrix containing *the n*-dimensional vectors (purchase histories) for *m* customers, we can compute all pairs similarity very straightforwardly.
- First compute all of the dot products using $A^T.A$ e.g.

$$\begin{pmatrix} 1 & 2 & 3 \\ -1 & 0 & 3 \end{pmatrix}\begin{pmatrix} 1 & -1 \\ 2 & 0 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 14 & 8 \\ 8 & 10 \end{pmatrix}$$

This is almost the covariance matrix – BUT we haven't subtracted the means of individual variables before doing the dot products.

- Then take every element on the leading diagonal and divide all of the elements in its containing row and column by it's square root:

$$\begin{pmatrix} {}^{14}/_{\sqrt{14 \times 14}} & {}^{8}/_{\sqrt{14 \times 10}} \\ {}^{8}/_{\sqrt{10 \times 14}} & {}^{10}/_{\sqrt{10 \times 10}} \end{pmatrix} = \begin{pmatrix} 1 & {}^{4}/_{\sqrt{35}} \\ {}^{4}/_{\sqrt{35}} & 1 \end{pmatrix}$$

[Divide i,j component by (length of vector i times length of vector j)]

All Pairs similarity

- Matrix multiplication can be done in O(m^2n), but can use Strassen's algorithm for large data.
- Dividing every element is O(m^2) so...
- All pairs similarity can be done in less than O(m^2n)
- Very important if we want to find clusters of similar objects (where objects are represented by vectors of real-valued features).

Summary

- Empirical computation of time complexity.

- Another set of notes on the elementary matrix operations.
- An application of matrices
- Algorithms for matrix multiplication