Fundamental Algorithms
Chapter 8: Matrices and Scientific Computing

Sevag Gharibian

Universität Paderborn
WS 2019

Outline

1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

References

- CLRS Chapters 28.1, 28.2, 30.1, 30.2
- M. Mahoney lecture notes: `https://www.stat.berkeley.edu/~mmahoney/f13-stat260-cs294/Lectures/lecture02.pdf`
- T. Leighton and T. Rubinfeld lecture notes: `http://web.mit.edu/neboat/Public/6.042/randomwalks.pdf`
- O. Levin: `http://discrete.openmathbooks.org/dmoi2/sec_recurrence.html`
- M. Nielsen lectures on Google technology: http://michaelnielsen.org/blog/lectures-on-the-google-technology-stack-1-introduction-to-pagerank/
- History of complex numbers https://www.cut-the-knot.org/arithmetic/algebra/HistoricalRemarks.shtml

Outline

1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

Matrices

Motivation - why matrices?

- Applications in most technical fields
- Physics: Classical mechanics, optics, electromagnetism, quantum mechanics
- Computer Science: Graphics, randomized algorithms, big data (e.g. Google's PageRank algorithm), quantum computing
- Mathematics: Graph theory, geometry, linear systems of equations, optimization
- Economics, game theory

Matrices

Motivation - why matrices?

- Applications in most technical fields
- Physics: Classical mechanics, optics, electromagnetism, quantum mechanics
- Computer Science: Graphics, randomized algorithms, big data (e.g. Google's PageRank algorithm), quantum computing
- Mathematics: Graph theory, geometry, linear systems of equations, optimization
- Economics, game theory

Note:

- Throughout these notes, we assume all operations are done over the field of real numbers, $\mathbb{R}$.
- We ignore issues of precision (which is an important topic).

Basics

Recall a 2×3 matrix M is given (e.g.) by:

$$M = \begin{pmatrix} 0 & 3 & -1 \\ 2 & 2 & 1 \end{pmatrix}.$$

The *transpose* of M is

$$M^T = \begin{pmatrix} 0 & 2 \\ 3 & 2 \\ -1 & 1 \end{pmatrix}.$$

The set of all $m \times n$ matrices over $\mathbb{R}$ is denoted $\mathbb{R}^{m\times n}$.

The *entry at position* (i, j) of M is denoted $M(i, j)$ or M_{ij}.

Basics

Recall: The set of all $m \times n$ matrices over $\mathbb{R}$ is denoted $\mathbb{R}^{m\times n}$.

Special cases of matrices:

- (Vectors) For $n = 1$ (resp. $m = 1$), have *column* (resp. *row*) vector:

$$\mathbf{v} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \qquad \mathbf{v}^T = \begin{pmatrix} 5 & 3 \end{pmatrix}.$$

Basics

Recall: The set of all $m \times n$ matrices over $\mathbb{R}$ is denoted $\mathbb{R}^{m\times n}$.

Special cases of matrices:

- (Vectors) For $n = 1$ (resp. $m = 1$), have *column* (resp. *row*) vector:

$$\mathbf{v} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \qquad \mathbf{v}^T = \begin{pmatrix} 5 & 3 \end{pmatrix}.$$

- (Square matrix) Set $m = n$.

Basics

Recall: The set of all $m \times n$ matrices over $\mathbb{R}$ is denoted $\mathbb{R}^{m\times n}$.

Special cases of matrices:

- (Vectors) For $n = 1$ (resp. $m = 1$), have *column* (resp. *row*) vector:

$$\mathbf{v} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \qquad \mathbf{v}^T = \begin{pmatrix} 5 & 3 \end{pmatrix}.$$

- (Square matrix) Set $m = n$.
- (Diagonal matrix) A square matrix M with $M_{ij} = 0$ if $i \neq j$.

Basics

Recall: The set of all $m \times n$ matrices over $\mathbb{R}$ is denoted $\mathbb{R}^{m\times n}$.

Special cases of matrices:

- (Vectors) For $n = 1$ (resp. $m = 1$), have *column* (resp. *row*) vector:

$$\mathbf{v} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \qquad \mathbf{v}^T = \begin{pmatrix} 5 & 3 \end{pmatrix}.$$

- (Square matrix) Set $m = n$.
- (Diagonal matrix) A square matrix M with $M_{ij} = 0$ if $i \neq j$.
- (Identity matrix) The $n \times n$ (diagonal) matrix ($n = 2$ below)

$$I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Matrix operations

- (Matrix addition) For any $M, N \in \mathbb{R}^{m \times n}$, $(M + N)_{ij} = M_{ij} + N_{ij}$.

  Ex. What is $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$?

Matrix operations

- (Matrix addition) For any $M, N \in \mathbb{R}^{m \times n}$, $(M + N)_{ij} = M_{ij} + N_{ij}$.

  Ex. What is $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$?

- (Scalar multiplication) For any $c \in \mathbb{R}$, $(cM)_{ij} = c \cdot M_{ij}$.

Matrix operations

- (Matrix addition) For any $M, N \in \mathbb{R}^{m\times n}$, $(M+N)_{ij} = M_{ij} + N_{ij}$.

  Ex. What is $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$?

- (Scalar multiplication) For any $c \in \mathbb{R}$, $(cM)_{ij} = c \cdot M_{ij}$.
- (Vector inner product) For any column vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$,

$$\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{n} v_i w_i \in \mathbb{R}.$$

  The inner product "measures" the overlap between **v** and **w**. When $\mathbf{v} \cdot \mathbf{w} = 0$, we say **v** and **w** are *orthogonal*.

  Ex. For $\mathbf{v} = (1 \;\; 0)^T$, $\mathbf{w} = (0 \;\; 1)^T$, what is $\mathbf{v} \cdot \mathbf{w}$? $\mathbf{v} \cdot \mathbf{v}$? Draw **v** and **w** on the 2D Euclidean plane to visualize the dot product.

Matrix Multiplication

- Since we are working over $\mathbb{R}$, can be defined using inner product[1].
- For any $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$:

$$(MN)_{ij} = M_{(i)}^T \cdot N^{(j)} = \sum_{k=1}^{n} M_{i,k}N_{k,j},$$

  where $M_{(i)}$ (resp. $M^{(i)}$) is the ith row of M (resp. ith column) of M.

  Ex. What is dimension of MN, i.e. what values are allowed for i, j?

- Examples:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}$$

[1] The analogous claim over $\mathbb{C}$ would not quite be correct.

Matrix Multiplication

- Since we are working over $\mathbb{R}$, can be defined using inner product[1].
- For any $M \in \mathbb{R}^{m \times n}$, $N \in \mathbb{R}^{n \times p}$:

$$(MN)_{ij} = M_{(i)}^T \cdot N^{(j)} = \sum_{k=1}^{n} M_{i,k} N_{k,j},$$

where $M_{(i)}$ (resp. $M^{(i)}$) is the ith row of M (resp. ith column) of M.

Ex. What is dimension of MN, i.e. what values are allowed for i, j?

- Examples:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} b \\ a \end{pmatrix}.$$

Q: In 2D plane, what operation does last equation encode?

[1] The analogous claim over $\mathbb{C}$ would not quite be correct.

Matrix Multiplication

More properties:

- For all $M \in \mathbb{R}^{m \times n}$, $I_m M = M I_n = M$.
- For any triple A, B, C (with appropriate dimensions):
  - (associativity) $A(BC) = (AB)C$
  - (distributivity) $A(B + C) = AB + AC$ and $(B + C)D = BD + CD$.
  - (commutativity) Does $AB = BA$ necessarily?

Ex. Let $M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $N = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Does MN equal NM?

[2] https://www.math.ucla.edu/~gyueun.lee/writing/stability_GSO.pdf

Matrix Multiplication

More properties:

- For all $M \in \mathbb{R}^{m\times n}$, $I_m M = M I_n = M$.
- For any triple A, B, C (with appropriate dimensions):
  - (associativity) $A(BC) = (AB)C$
  - (distributivity) $A(B + C) = AB + AC$ and $(B + C)D = BD + CD$.
  - (commutativity) Does $AB = BA$ necessarily?

Ex. Let $M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $N = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Does MN equal NM?

Life lesson

That matrix multiplication is non-commutative is *not* just an academic question! The structure of the world around us depends on this property — it gives rise to the *uncertainty principle* in quantum mechanics, which in turn is used[2] to explain why matter is stable (i.e. why doesn't an electron just crash into the nucleus of the atom?).

[2] https://www.math.ucla.edu/~gyueun.lee/writing/stability_GSO.pdf

Bit versus operation complexity

Q: Naive worst-case “runtime” for multiplying $M, N \in \mathbb{R}^{n\times n}$?

Bit versus operation complexity

Q: Naive worst-case "runtime" for multiplying $M, N \in \mathbb{R}^{n\times n}$?

- Need to compute $O(n^2)$ entries for MN.
- Each entry MN_{ij} is the inner product of two n-dimensional vectors.
- Therefore, total "cost" $O(n^3)$. But... what does "cost" mean?

Bit versus operation complexity

Q: Naive worst-case "runtime" for multiplying $M, N \in \mathbb{R}^{n\times n}$?

- Need to compute $O(n^2)$ entries for MN.
- Each entry MN_{ij} is the inner product of two n-dimensional vectors.
- Therefore, total "cost" $O(n^3)$. But... what does "cost" mean?
- More accurate: $O(n^3)$ *field operations* over $\mathbb{R}$, i.e. additions and multiplications over $\mathbb{R}$. (We assume each field operation costs $O(1)$.)

Bit versus operation complexity

Q: Naive worst-case "runtime" for multiplying $M, N \in \mathbb{R}^{n\times n}$?

- Need to compute $O(n^2)$ entries for MN.
- Each entry MN_{ij} is the inner product of two n-dimensional vectors.
- Therefore, total "cost" $O(n^3)$. But... what does "cost" mean?
- More accurate: $O(n^3)$ *field operations* over $\mathbb{R}$, i.e. additions and multiplications over $\mathbb{R}$. (We assume each field operation costs $O(1)$.)

Can also do "low-level" analysis by factoring in cost of each field op:

- E.g. How many steps to actually implement n-bit addition of integers on a Turing machine? (Answer: $O(n)$.)
- This cost model is called *bit complexity*.

Here, we focus on operation complexity, i.e. we will not worry about the low-level details of implementing addition, multiplication etc over $\mathbb{R}$.

Q: Can we beat the naive $O(n^3)$ matrix multiplication time?

Q: Can we beat the naive $O(n^3)$ matrix multiplication time?

A: (If the answer was *no*, would you be sitting here?)

Q: Can we beat the naive $O(n^3)$ matrix multiplication time?

A: (If the answer was *no*, would you be sitting here?)

Strassen's Algorithm

- Strassen, Volker. Gaussian Elimination is not Optimal, *Numer. Math.* 13, p. 354–356, 1969.
- Requires $O(n^{2.808})$ operations.
- Recursive, divide-and-conquer approach.
- Quite a surprise to the research community!

Outline

Goals of section

- Practice working with matrices
- Practice working with randomization
- Study a mix of classic and modern algorithms

Outline

1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

Warmup

Note: For simplicity, we assume n is a power of 2, where $M, N \in \mathbb{R}^{n \times n}$.

Write M, N, MN in block form. For $a, b, c, d, e, f, g, h, r, s, t, u \in \mathbb{R}^{\frac{n}{2} \times \frac{n}{2}}$:

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad N = \begin{pmatrix} e & f \\ g & h \end{pmatrix}, \quad MN = \begin{pmatrix} r & s \\ t & u \end{pmatrix}.$$

Warmup

Note: For simplicity, we assume n is a power of 2, where $M, N \in \mathbb{R}^{n\times n}$.

Write M, N, MN in block form. For $a, b, c, d, e, f, g, h, r, s, t, u \in \mathbb{R}^{\frac{n}{2}\times\frac{n}{2}}$:

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad N = \begin{pmatrix} e & f \\ g & h \end{pmatrix}, \quad MN = \begin{pmatrix} r & s \\ t & u \end{pmatrix}.$$

Naive algorithm:

- Compute each block of MN independently as follows.

$$r = ae + bg \qquad s = af + bh \qquad t = ce + dg \qquad u = cf + dh.$$

- Recursively compute each $n/2 \times n/2$ product ae, bg, etc. . .

Warmup

Note: For simplicity, we assume n is a power of 2, where $M, N \in \mathbb{R}^{n\times n}$.

Write M, N, MN in block form. For $a, b, c, d, e, f, g, h, r, s, t, u \in \mathbb{R}^{\frac{n}{2}\times\frac{n}{2}}$:

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad N = \begin{pmatrix} e & f \\ g & h \end{pmatrix}, \quad MN = \begin{pmatrix} r & s \\ t & u \end{pmatrix}.$$

Naive algorithm:

- Compute each block of MN independently as follows.

$$r = ae + bg \qquad s = af + bh \qquad t = ce + dg \qquad u = cf + dh.$$

- Recursively compute each $n/2 \times n/2$ product ae, bg, etc. . . .

Cost: For $M, N \in \mathbb{R}^{n\times n}$, recurrence relation for multiplication costs $T(n)$:

$$T(n) = 8T(n/2) + \Theta(n^2) \in \Theta(n^{\log_2 8}) \in \Theta(n^3) \ldots \text{ (why?)}$$

. . . no improvement!

Cost: For $M, N \in \mathbb{R}^{n\times n}$, have recurrence

$$T(n) = 8T(n/2) + \Theta(n^2) \in \Theta(n^{\log_2 8}) \in \Theta(n^3)\ldots$$

. . . no improvement!

This cost was too large because we needed 8 recursive calls per level...

Q: Can we do it with 7 recursive calls?

Cost: For $M, N \in \mathbb{R}^{n \times n}$, have recurrence

$$T(n) = 8T(n/2) + \Theta(n^2) \in \Theta(n^{\log_2 8}) \in \Theta(n^3) \ldots$$

...no improvement!

This cost was too large because we needed 8 recursive calls per level...

Q: Can we do it with 7 recursive calls?

- Remarkably, yes!
- We hence get runtime $\Theta(n^{\log_2 7}) \in \Theta(n^{2.808})$, as claimed.
- Ok, so how do we do it?

Strassen's algorithm - a bit of magic

1. Compute the following 7 products (recursively):

$$
\begin{array}{lll}
P_1 = a(f-h) & P_2 = (a+b)h & P_3 = (c+d)e \\
P_4 = d(g-e) & P_5 = (a+d)(e+h) & P_6 = (b-d)(g+h) \\
P_7 = (a-c)(e+f) & &
\end{array}
$$

Strassen's algorithm - a bit of magic

1. Compute the following 7 products (recursively):

$$
\begin{array}{lll}
P_1 = a(f-h) & P_2 = (a+b)h & P_3 = (c+d)e \\
P_4 = d(g-e) & P_5 = (a+d)(e+h) & P_6 = (b-d)(g+h) \\
P_7 = (a-c)(e+f) & &
\end{array}
$$

2. Recall we wish to compute each block of MN, i.e.:

$$r = ae + bg \qquad s = af + bh \qquad t = ce + dg \qquad u = cf + dh.$$

Magically, we have:

$$
\begin{array}{ll}
r = P_5 + P_4 - P_2 + P_6 & s = P_1 + P_2 \\
t = P_3 + P_4 & u = P_5 + P_1 - P_3 - P_7.
\end{array}
$$

Strassen's algorithm - a bit of magic

1. Compute the following 7 products (recursively):

$$\begin{array}{lll} P_1 = a(f-h) & P_2 = (a+b)h & P_3 = (c+d)e \\ P_4 = d(g-e) & P_5 = (a+d)(e+h) & P_6 = (b-d)(g+h) \\ P_7 = (a-c)(e+f) & & \end{array}$$

2. Recall we wish to compute each block of MN, i.e.:

$$r = ae + bg \qquad s = af + bh \qquad t = ce + dg \qquad u = cf + dh.$$

Magically, we have:

$$\begin{array}{ll} r = P_5 + P_4 - P_2 + P_6 & s = P_1 + P_2 \\ t = P_3 + P_4 & u = P_5 + P_1 - P_3 - P_7. \end{array}$$

Cost: For $M, N \in \mathbb{R}^{n\times n}$, have recurrence

$$T(n) = 7T(n/2) + \Theta(n^2) \in \Theta(n^{\log_2 7}) \in \Theta(n^{2.808})!$$

Two questions you should always ask yourself:

1. Is this asymptotic improvement useful in practice?

[3] The precise definition of "numerically stable" depends on context. Roughly, it means one wants the algorithm to "behave well" even on "bad inputs/edge cases".

Two questions you should always ask yourself:

1. Is this asymptotic improvement useful in practice?
   - Constant factor hidden by Big-Oh notation is large for Strassen's method. In practice, for small inputs cheaper to run naive method.

[3] The precise definition of "numerically stable" depends on context. Roughly, it means one wants the algorithm to "behave well" even on "bad inputs/edge cases".

Two questions you should always ask yourself:

1. Is this asymptotic improvement useful in practice?
   - Constant factor hidden by Big-Oh notation is large for Strassen's method. In practice, for small inputs cheaper to run naive method.
   - If matrices have special structure (e.g. *sparse*, meaning have few non-zero entries), faster methods exist.

[3] The precise definition of "numerically stable" depends on context. Roughly, it means one wants the algorithm to "behave well" even on "bad inputs/edge cases".

Two questions you should always ask yourself:

1. Is this asymptotic improvement useful in practice?
   - Constant factor hidden by Big-Oh notation is large for Strassen's method. In practice, for small inputs cheaper to run naive method.
   - If matrices have special structure (e.g. *sparse*, meaning have few non-zero entries), faster methods exist.
   - Strassen's algorithm is less *numerically stable*[3] than naive method for some applications.

[3] The precise definition of "numerically stable" depends on context. Roughly, it means one wants the algorithm to "behave well" even on "bad inputs/edge cases".

Two questions you should always ask yourself:

1. Is this asymptotic improvement useful in practice?
   - Constant factor hidden by Big-Oh notation is large for Strassen's method. In practice, for small inputs cheaper to run naive method.
   - If matrices have special structure (e.g. *sparse*, meaning have few non-zero entries), faster methods exist.
   - Strassen's algorithm is less *numerically stable*[3] than naive method for some applications.
   - As stated, Strassen's algorithm uses space for recursions on submatrices (there are ways around this).

[3] The precise definition of "numerically stable" depends on context. Roughly, it means one wants the algorithm to "behave well" even on "bad inputs/edge cases".

Two questions you should always ask yourself:

1. Is this asymptotic improvement useful in practice?
   - Constant factor hidden by Big-Oh notation is large for Strassen's method. In practice, for small inputs cheaper to run naive method.
   - If matrices have special structure (e.g. *sparse*, meaning have few non-zero entries), faster methods exist.
   - Strassen's algorithm is less *numerically stable*[3] than naive method for some applications.
   - As stated, Strassen's algorithm uses space for recursions on submatrices (there are ways around this).
2. Can we do better?

[3] The precise definition of "numerically stable" depends on context. Roughly, it means one wants the algorithm to "behave well" even on "bad inputs/edge cases".

Can we do better?

Lower bounds

- Naive lower bound of $\Omega(n^2)$. (Why?)

Can we do better?

Lower bounds

- Naive lower bound of $\Omega(n^2)$. (Why?)
- Embarrassingly, unknown whether optimal is $\omega(n^2)$ (after 50 years!)

Can we do better?

Lower bounds

- Naive lower bound of $\Omega(n^2)$. (Why?)
- Embarrassingly, unknown whether optimal is $\omega(n^2)$ (after 50 years!)
- If we restrict the type of circuit computing the matrix product, then a lower bound of $\Omega(n^2 \log n)$ can be shown [Raz, 2003]

Can we do better?

Upper bounds

- Strassen (1969): $O(n^{2.808})$.
- Pan (1978): $o(n^{2.796})$
- Bini, Capovani, Romani, Lotti using *border rank* (1979): $o(n^{2.78})$
- Schönhage via τ*-theorem* (1981): $o(n^{2.548})$
- Romani (1982): $o(n^{2.517})$
- Coppersmith, Winograd (1981): $o(n^{2.496})$
- Strassen via *laser method* (1986): $o(n^{2.479})$
- Coppersmith, Winograd (1989): $o(n^{2.376})$
- V. V. Williams (2013): $O(n^{2.3729})$
- Le Gall (2014): $O(n^{2.3728639})$

The more advanced these algorithms get, the less useful they tend to be in practice. . .

What if we want something more useful in practice? Say for machine learning or big data?

The more advanced these algorithms get, the less useful they tend to be in practice. . .

What if we want something more useful in practice? Say for machine learning or big data?

Common tool: Randomization

Tradeoff: Time/space versus accuracy

SIMONS INSTITUTE for the Theory of Computing

Berkeley University of California

Search

HOME ABOUT PEOPLE PROGRAMS & EVENTS VISITING SUPPORT WATCH / READ CALENDAR

‹ Workshops & Symposia ‹ Programs & Events ‹ Home

- Overview
- Programs
- Workshops & Symposia
  - › Upcoming Workshops & Symposia
  - › Past Workshops & Symposia
- Internal Program Activities
- Public Lectures
- Participate

Workshops | Fall 2018

Randomized Numerical Linear Algebra and Applications

Sep. 24 – Sep. 27, 2018

Program: Foundations of Data Science

Add to Calendar

View schedule & video » View abstracts »

Organizers:

Petros Drineas (Purdue University; chair), Ken Clarkson (IBM Almaden), Prateek Jain (Microsoft Research India), Michael Mahoney (International Computer Science Institute and UC Berkeley)

The focus of this workshop will be on recent developments in randomized linear algebra, with an emphasis on how algorithmic improvements from the theory of algorithms interact with statistical, optimization, inference, and related perspectives. One focus area of the workshop will be the broad use of sketching techniques developed in the data stream literature for solving optimization problems in linear and multi-linear algebra. The workshop will also consider the impact of theoretical developments in randomized linear algebra on (i) numerical analysis as a method for constructing preconditioners; (ii) applications as a principled feature selection method; and (iii) implementations as a way to avoid communication rather than computation. Another goal of this workshop is thus to bridge the theory-practice gap by trying to understand the needs of practitioners when working on real datasets.

Outline

1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

Basic probability theory

Let X be a discrete random variable taking values from $S = \{1, \ldots, n\}$.

- The *probability* that X takes value $x \in S$ is $\Pr(X = x)$, or $\Pr(x)$.

Basic probability theory

Let X be a discrete random variable taking values from $S = \{1, \ldots, n\}$.

- The *probability* that X takes value $x \in S$ is $\Pr(X = x)$, or $\Pr(x)$.
- The *expected value* of X is

$$E[X] = \sum_{x \in S} \Pr(x) \cdot x.$$

Note: Expected value is a *linear* function, i.e. $E[X + Y] = E[X] + E[Y]$.

Basic probability theory

Let X be a discrete random variable taking values from $S = \{1, \ldots, n\}$.

- The *probability* that X takes value $x \in S$ is $\Pr(X = x)$, or $\Pr(x)$.
- The *expected value* of X is

$$E[X] = \sum_{x \in S} \Pr(x) \cdot x.$$

  Note: Expected value is a *linear* function, i.e. $E[X + Y] = E[X] + E[Y]$.

- The *variance* of X is $\mathrm{Var}[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$.

Basic probability theory

Let X be a discrete random variable taking values from $S = \{1, \ldots, n\}$.

- The *probability* that X takes value $x \in S$ is $\Pr(X = x)$, or $\Pr(x)$.
- The *expected value* of X is

$$E[X] = \sum_{x \in S} \Pr(x) \cdot x.$$

  Note: Expected value is a *linear* function, i.e. $E[X + Y] = E[X] + E[Y]$.

- The *variance* of X is $\mathrm{Var}[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$.

Ex. Let $X \in \{1, -1\}$ be a random variable corresponding to a sampling experiment in which a fair coin is flipped, and if the coin lands HEADS (resp. TAILS), you gain (resp. lose) 1 EUR. What is $E[X]$? What is $\mathrm{Var}[X]$?

Back to matrix multiplication

Recall: Over $\mathbb{R}$, matrix multiplication can be viewed as *inner products* over rows of M and columns of N.

For any $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$:

$$(MN)_{ij} = M_{(i)}^T \cdot N^{(j)} = \sum_{k=1}^{n} M_{i,k} N_{k,j},$$

where $M_{(i)}$ (resp. $M^{(i)}$) is the ith row of M (resp. ith column) of M.

Outer products

Inner product of $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ multiplies row vector by column vector:

$$\mathbf{v} \cdot \mathbf{w} = \mathbf{v}^T \mathbf{w} = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \in \mathbb{R}.$$

Outer products

Inner product of $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ multiplies row vector by column vector:

$$\mathbf{v} \cdot \mathbf{w} = \mathbf{v}^T\mathbf{w} = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \in \mathbb{R}.$$

Outer product of $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ multiplies column vector by row vector:

$$\mathbf{v}\mathbf{w}^T = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \begin{pmatrix} w_1 & w_2 & \cdots & w_n \end{pmatrix} \in \mathbb{R}^{?}.$$

Outer products

Inner product of $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ multiplies row vector by column vector:

$$\mathbf{v} \cdot \mathbf{w} = \mathbf{v}^T \mathbf{w} = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \in \mathbb{R}.$$

Outer product of $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^n$ multiplies column vector by row vector:

$$\mathbf{v}\mathbf{w}^T = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \begin{pmatrix} w_1 & w_2 & \cdots & w_n \end{pmatrix} = \begin{pmatrix} v_1 w_1 & v_1 w_2 & \cdots & v_1 w_n \\ v_2 w_1 & v_2 w_2 & \cdots & v_2 w_n \\ \vdots & \vdots & \ddots & \vdots \\ v_n w_1 & v_n w_2 & \cdots & v_n w_n \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

Q: What dimensions does the outer product of $\mathbf{v} \in \mathbb{R}^m$ and $\mathbf{w} \in \mathbb{R}^n$ have?

Ex: Let $\mathbf{v} = (1 \;\; 0)^T$, $\mathbf{w} = (0 \;\; 1)^T$. What are inner/outer products of $\mathbf{v}$ and $\mathbf{w}$?

Back to matrix multiplication

Inner product view: For any $M \in \mathbb{R}^{m \times n}$, $N \in \mathbb{R}^{n \times p}$:

$$(MN)_{ij} = M_{(i)}^T \cdot N^{(j)} = \begin{pmatrix} \quad \text{Row } i \text{ of } M \quad \end{pmatrix} \begin{pmatrix} \text{Column} \\ j \\ \text{of } N \end{pmatrix} \in \mathbb{R}.$$

Back to matrix multiplication

Inner product view: For any $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$:

$$(MN)_{ij} = M_{(i)}^T \cdot N^{(j)} = \begin{pmatrix} \quad \text{Row } i \text{ of } M \quad \end{pmatrix} \begin{pmatrix} \text{Column} \\ j \\ \text{of } N \end{pmatrix} \in \mathbb{R}.$$

Outer product view: For any $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$:

$$MN = \sum_{k=1}^{n} M^{(k)} N_{(k)} = \sum_{k=1}^{n} \begin{pmatrix} \text{Column} \\ k \\ \text{of } M \end{pmatrix} \begin{pmatrix} \quad \text{Row } k \text{ of } M \quad \end{pmatrix} \in \mathbb{R}^{m\times p}.$$

Q: What differences can you spot between the inner and outer product views?

Ex: Prove that the outer product view is correct.

So MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

So MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Let's take inspiration from sums of real numbers

Suppose wish to approximate sum $\sum_{k=1}^{n} a_i$ over $a_i \in \mathbb{R}$ *without* adding all a_i.

So MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Let's take inspiration from sums of real numbers

Suppose wish to approximate sum $\sum_{k=1}^{n} a_i$ over $a_i \in \mathbb{R}$ *without* adding all a_i.

Idea:

1. Uniformly & independently sample s terms (with replacement) from $\{a_i\}$.

So MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Let's take inspiration from sums of real numbers

Suppose wish to approximate sum $\sum_{k=1}^{n} a_i$ over $a_i \in \mathbb{R}$ *without* adding all a_i.

Idea:

1. Uniformly & independently sample s terms (with replacement) from $\{a_i\}$.
2. Add all the samples; call this sum q.

So MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Let's take inspiration from sums of real numbers

Suppose wish to approximate sum $\sum_{k=1}^{n} a_i$ over $a_i \in \mathbb{R}$ *without* adding all a_i.

Idea:

1. Uniformly & independently sample s terms (with replacement) from $\{a_i\}$.
2. Add all the samples; call this sum q.
3. Output αq for appropriate rescaling factor α. (Why need α?)

So MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Let's take inspiration from sums of real numbers

Suppose wish to approximate sum $\sum_{k=1}^{n} a_i$ over $a_i \in \mathbb{R}$ *without* adding all a_i.

Idea:

1. Uniformly & independently sample s terms (with replacement) from $\{a_i\}$.
2. Add all the samples; call this sum q.
3. Output αq for appropriate rescaling factor α. (Why need α?)

Sampling Lemma (Arora, Karger, Karpinski, 1999)

Suppose $\forall i$, $|a_i| \leq M$ for fixed M. If $s = g \log n$ samples are drawn, then

$$\sum_{i=1}^{n} a_i - nM\sqrt{\frac{f}{g}} \leq \alpha q \leq \sum_{i=1}^{n} a_i + nM\sqrt{\frac{f}{g}}$$

with probability at least $1 - n^{-f}$, for $\alpha = \frac{n}{s}$ and $f, g > 0$.

Digesting the Sampling Lemma

Sampling Lemma (Arora, Karger, Karpinski, 1999)

Suppose $\forall i$, $|a_i| \leq M$ for fixed M. If $s = g \log n$ samples are drawn, then

$$\sum_{i=1}^{n} a_i - nM\sqrt{\frac{f}{g}} \leq \alpha q \leq \sum_{i=1}^{n} a_i + nM\sqrt{\frac{f}{g}}$$

with probability at least $1 - n^{-f}$, for $\alpha = \frac{n}{s}$ and $f, g > 0$.

Note that the error:

- is *additive*, i.e. of form $\pm\epsilon$,

Digesting the Sampling Lemma

Sampling Lemma (Arora, Karger, Karpinski, 1999)

Suppose $\forall i$, $|a_i| \leq M$ for fixed M. If $s = g \log n$ samples are drawn, then

$$\sum_{i=1}^{n} a_i - nM\sqrt{\frac{f}{g}} \leq \alpha q \leq \sum_{i=1}^{n} a_i + nM\sqrt{\frac{f}{g}}$$

with probability at least $1 - n^{-f}$, for $\alpha = \frac{n}{s}$ and $f, g > 0$.

Note that the error:

- is *additive*, i.e. of form $\pm\epsilon$,
- scales with the number of terms in the sum, n,

Digesting the Sampling Lemma

Sampling Lemma (Arora, Karger, Karpinski, 1999)

Suppose $\forall i$, $|a_i| \leq M$ for fixed M. If $s = g \log n$ samples are drawn, then

$$\sum_{i=1}^{n} a_i - nM\sqrt{\frac{f}{g}} \leq \alpha q \leq \sum_{i=1}^{n} a_i + nM\sqrt{\frac{f}{g}}$$

with probability at least $1 - n^{-f}$, for $\alpha = \frac{n}{s}$ and $f, g > 0$.

Note that the error:

- is *additive*, i.e. of form $\pm\epsilon$,
- scales with the number of terms in the sum, n,
- scales with the magnitude bound, M,

Digesting the Sampling Lemma

Sampling Lemma (Arora, Karger, Karpinski, 1999)

Suppose $\forall i$, $|a_i| \leq M$ for fixed M. If $s = g \log n$ samples are drawn, then

$$\sum_{i=1}^{n} a_i - nM\sqrt{\frac{f}{g}} \leq \alpha q \leq \sum_{i=1}^{n} a_i + nM\sqrt{\frac{f}{g}}$$

with probability at least $1 - n^{-f}$, for $\alpha = \frac{n}{s}$ and $f, g > 0$.

Note that the error:

- is *additive*, i.e. of form $\pm\epsilon$,
- scales with the number of terms in the sum, n,
- scales with the magnitude bound, M,
- scales inversely with coefficient in the number of samples, g.

Digesting the Sampling Lemma

Sampling Lemma (Arora, Karger, Karpinski, 1999)

Suppose $\forall i$, $|a_i| \leq M$ for fixed M. If $s = g \log n$ samples are drawn, then

$$\sum_{i=1}^{n} a_i - nM\sqrt{\frac{f}{g}} \leq \alpha q \leq \sum_{i=1}^{n} a_i + nM\sqrt{\frac{f}{g}}$$

with probability at least $1 - n^{-f}$, for $\alpha = \frac{n}{s}$ and $f, g > 0$.

Note that the error:

- is *additive*, i.e. of form $\pm\epsilon$,
- scales with the number of terms in the sum, n,
- scales with the magnitude bound, M,
- scales inversely with coefficient in the number of samples, g.

Obvious question: Can we do something similar for matrix multiplication?

Drineas-Kannan-Mahoney algorithm

Recall:

- $M \in \mathbb{R}^{m \times n}$, $N \in \mathbb{R}^{n \times p}$.
- MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Drineas-Kannan-Mahoney algorithm

Recall:

- $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$.
- MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Algorithm:

1. Set C to the $m \times p$ zero matrix. //C will store estimate for MN

Drineas-Kannan-Mahoney algorithm

Recall:

- $M \in \mathbb{R}^{m \times n}$, $N \in \mathbb{R}^{n \times p}$.
- MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Algorithm:

1. Set C to the $m \times p$ zero matrix. //C will store estimate for MN
2. For $t = 1 \ldots s$ do: //draw s samples
   1. Pick $k_t \in \{1, \ldots, n\}$ uniformly at random.

Drineas-Kannan-Mahoney algorithm

Recall:

- $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$.
- MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Algorithm:

1. Set C to the $m \times p$ zero matrix. //C will store estimate for MN
2. For $t = 1 \ldots s$ do: //draw s samples
   1. Pick $k_t \in \{1, \ldots, n\}$ uniformly at random.
   2. Set $C = C + \frac{n}{s} M^{(k_t)} N_{(k_t)}$.
3. Output C.

Drineas-Kannan-Mahoney algorithm

Recall:

- $M \in \mathbb{R}^{m \times n}$, $N \in \mathbb{R}^{n \times p}$.
- MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Algorithm:

1. Set C to the $m \times p$ zero matrix. //C will store estimate for MN
2. For $t = 1 \dots s$ do: //draw s samples
    1. Pick $k_t \in \{1, \dots, n\}$ uniformly at random.
    2. Set $C = C + \frac{n}{s} M^{(k_t)} N_{(k_t)}$.
3. Output C.

Q: How "close" is C to MN? Specifically, how do we define an "absolute value function" $|C - MN|$ for matrices C, M, N?

Norms

What properties does absolute value function (on $\mathbb{R}$) have? $\forall a, b \in \mathbb{R}$:

1. (Non-negativity) $|a| \geq 0$.
2. (Subadditivity) $|a+b| \leq |a| + |b|$.
3. (Multiplicativity) $|ab| = |a|\,|b|$.
4. (Positive definiteness) $|a| = 0$ iff $a = 0$.

Norms

What properties does absolute value function (on $\mathbb{R}$) have? $\forall a, b \in \mathbb{R}$:

1. (Non-negativity) $|a| \geq 0$.
2. (Subadditivity) $|a + b| \leq |a| + |b|$.
3. (Multiplicativity) $|ab| = |a|\,|b|$.
4. (Positive definiteness) $|a| = 0$ iff $a = 0$.

A *norm* $\|\cdot\| : V \mapsto \mathbb{R}_{\geq 0}$ generalizes this to vector spaces V over a field $F = \mathbb{R}$.

Any norm, by definition, satisfies that for all $c \in F$, $\mathbf{v}, \mathbf{w} \in V$:

1. (Non-negativity) $\|v\| \geq 0$.
2. (Subadditivity) $\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|$.
3. (Absolute scalability) $\|c\mathbf{v}\| = |c|\,\|\mathbf{v}\|$.
4. (Positive definiteness) $\|\mathbf{v}\| = 0$ iff $\mathbf{v} = 0$ (i.e. $\mathbf{v}$ is zero vector).

Recall: A vector space can refer to a space of *vectors* or *matrices*.

Constructing norms

Like absolute value function, a norm should "measure the size" of its input.

Q: How to construct functions $\|\cdot\|$ satisfying properties 1-4 of a norm?

Constructing norms

Like absolute value function, a norm should "measure the size" of its input.

Q: How to construct functions $\|\cdot\|$ satisfying properties 1-4 of a norm?
A: Infinite number of ways!

But you already know one way...let's use that.

Euclidean norm for "vectors"

Let $V = \mathbb{R}^n$. Then, Euclidean norm of $\mathbf{v} \in V$ is $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$.

Constructing norms

Like absolute value function, a norm should "measure the size" of its input.

Q: How to construct functions $\|\cdot\|$ satisfying properties 1-4 of a norm?
A: Infinite number of ways!

But you already know one way... let's use that.

Euclidean norm for "vectors"

Let $V = \mathbb{R}^n$. Then, Euclidean norm of $\mathbf{v} \in V$ is $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^n v_i^2}$.

Frobenius norm for "matrices"

Let $V = \mathbb{R}^{m\times n}$. Then, Frobenius norm of $M \in V$ is $\|M\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n M_{ij}^2}$.

Constructing norms

Like absolute value function, a norm should “measure the size” of its input.

Q: How to construct functions $\|\cdot\|$ satisfying properties 1-4 of a norm?
A: Infinite number of ways!

But you already know one way... let’s use that.

Euclidean norm for “vectors”

Let $V = \mathbb{R}^n$. Then, Euclidean norm of $\mathbf{v} \in V$ is $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^n v_i^2}$.

Frobenius norm for “matrices”

Let $V = \mathbb{R}^{m\times n}$. Then, Frobenius norm of $M \in V$ is $\|M\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n M_{ij}^2}$.

Note: These two are actually the same thing if you “reshape” M into a vector $\mathbf{v}$ by concatenating its columns.

Exercises on norms

1. Define $\mathbf{v} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. What is $\|\mathbf{v}\|_2$?
2. Draw $\mathbf{v}$ in the 2D Euclidean plane. What does $\|\mathbf{v}\|_2$ represent?
3. What does the subadditivity property represent in the 2D plane?
4. Prove that the Euclidean norm is indeed a norm.
5. Let's consider a different norm, the *Taxicab norm* or 1*-norm*:

   $$\|\mathbf{v}\|_1 = \sum_{i=1}^{n} |v_i| .$$

   What is $\|\mathbf{v}\|_1$ for $\mathbf{v}$ from the first exercise above? What does the Taxicab norm represent on the Euclidean plane?
6. Define $M = \begin{pmatrix} -1 & 1 \\ 2 & -4 \end{pmatrix}$. What is $\|M\|_{\mathrm{F}}$?
7. Prove that the Frobenius norm is indeed a norm. (Hint: This should require no additional work.)

Exercises on norms

1. Define $\mathbf{v} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. What is $\|\mathbf{v}\|_2$?
2. Draw $\mathbf{v}$ in the 2D Euclidean plane. What does $\|\mathbf{v}\|_2$ represent?
3. What does the subadditivity property represent in the 2D plane?
4. Prove that the Euclidean norm is indeed a norm.
5. Let's consider a different norm, the *Taxicab norm* or 1*-norm*:

$$\|\mathbf{v}\|_1 = \sum_{i=1}^{n} |v_i| .$$

   What is $\|\mathbf{v}\|_1$ for $\mathbf{v}$ from the first exercise above? What does the Taxicab norm represent on the Euclidean plane?

6. Define $M = \begin{pmatrix} -1 & 1 \\ 2 & -4 \end{pmatrix}$. What is $\|M\|_{\mathrm{F}}$?
7. Prove that the Frobenius norm is indeed a norm. (Hint: This should require no additional work.)

Note: There is more than one way to generalize the 1-norm to matrices.

Returning to our question

Recall:

- $M \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$.
- MN is sum over (rank 1) products $M^{(k)}N_{(k)}$, i.e. $MN = \sum_{k=1}^{n} M^{(k)}N_{(k)}$.

Algorithm:

1. Set C to the $m \times p$ zero matrix. //C will store estimate for MN
2. For $t = 1 \ldots s$ do: //draw s samples
   1. Pick $k_t \in \{1, \ldots, n\}$ uniformly at random.
   2. Set $C = C + \frac{n}{c} M^{(k_t)} N_{(k_t)}$.
3. Output C.

Q: How "close" is C to MN? Specifically, how do we define an "absolute value function" $|C - MN|$ for matrices C, M, N?

Quality of approximation

Lemma (Drineas-Kannan-Mahoney, 2006)

For input matrices M and N, suppose the DKM algorithm makes s samples and outputs matrix C. Then for all indices i, j:

$$E[C_{ij}] = (MN)_{ij} \quad \text{and} \quad \mathrm{Var}[C_{ij}] = \frac{1}{s}\left(n\sum_{k=1}^{n} M_{ik}^2 N_{kj}^2 - (MN)_{ij}^2\right).$$

Quality of approximation

Lemma (Drineas-Kannan-Mahoney, 2006)

For input matrices M and N, suppose the DKM algorithm makes s samples and outputs matrix C. Then for all indices i, j:

$$E[C_{ij}] = (MN)_{ij} \quad \text{and} \quad \mathrm{Var}[C_{ij}] = \frac{1}{s}\left(n\sum_{k=1}^{n} M_{ik}^2 N_{kj}^2 - (MN)_{ij}^2\right).$$

Proof. For iteration t, define $X_t = (\frac{n}{s} M^{(k_t)} N_{(k_t)})_{ij}$ //(i,j)th entry of sample t.

Quality of approximation

Lemma (Drineas-Kannan-Mahoney, 2006)

For input matrices M and N, suppose the DKM algorithm makes s samples and outputs matrix C. Then for all indices i, j:

$$E[C_{ij}] = (MN)_{ij} \quad \text{and} \quad \mathrm{Var}[C_{ij}] = \frac{1}{s}\left(n\sum_{k=1}^{n} M_{ik}^2 N_{kj}^2 - (MN)_{ij}^2\right).$$

Proof. For iteration t, define $X_t = (\frac{n}{s} M^{(k_t)} N_{(k_t)})_{ij}$ //(i, j)th entry of sample t.

Observe that $X_t = \frac{n}{s} M_{ik_t} N_{k_t j}$. So:

Quality of approximation

Lemma (Drineas-Kannan-Mahoney, 2006)

For input matrices M and N, suppose the DKM algorithm makes s samples and outputs matrix C. Then for all indices i, j:

$$E[C_{ij}] = (MN)_{ij} \quad \text{and} \quad \mathrm{Var}[C_{ij}] = \frac{1}{s}\left(n\sum_{k=1}^{n} M_{ik}^2 N_{kj}^2 - (MN)_{ij}^2\right).$$

Proof. For iteration t, define $X_t = (\frac{n}{s}M^{(k_t)}N_{(k_t)})_{ij}$ //(i,j)th entry of sample t.

Observe that $X_t = \frac{n}{s}M_{ik_t}N_{k_tj}$. So:

$$E[X_t] = \sum_{k=1}^{n}\frac{1}{n}\left(\frac{n}{s}M_{ik}N_{kj}\right) = \frac{1}{s}(MN)_{ij} \quad \text{and} \quad E[X_t^2] = \sum_{k=1}^{n}\frac{n}{s^2}M_{ik}^2N_{kj}^2.$$

Quality of approximation

Lemma (Drineas-Kannan-Mahoney, 2006)

For input matrices M and N, suppose the DKM algorithm makes s samples and outputs matrix C. Then for all indices i, j:

$$E[C_{ij}] = (MN)_{ij} \quad \text{and} \quad \mathrm{Var}[C_{ij}] = \frac{1}{s}\left(n\sum_{k=1}^{n} M_{ik}^2 N_{kj}^2 - (MN)_{ij}^2\right).$$

Proof. For iteration t, define $X_t = (\frac{n}{s}M^{(k_t)}N_{(k_t)})_{ij}$ //(i,j)th entry of sample t.

Observe that $X_t = \frac{n}{s}M_{ik_t}N_{k_tj}$. So:

$$E[X_t] = \sum_{k=1}^{n}\frac{1}{n}\left(\frac{n}{s}M_{ik}N_{kj}\right) = \frac{1}{s}(MN)_{ij} \quad \text{and} \quad E[X_t^2] = \sum_{k=1}^{n}\frac{n}{s^2}M_{ik}^2N_{kj}^2.$$

$$E[C_{ij}] = E\left[\sum_{t=1}^{s} X_t\right] = \sum_{t=1}^{s} E[X_t] = (MN)_{ij}$$

$$\mathrm{Var}[C_{ij}] = \mathrm{Var}\left[\sum_{t=1}^{s} X_t\right] = \sum_{t=1}^{s}\mathrm{Var}[X_t] = \sum_{t=1}^{s}\left(\sum_{k=1}^{n}\frac{n}{s^2}M_{ik}^2N_{kj}^2 - \frac{1}{s^2}(MN)_{ij}^2\right).$$

Q: Why do red equalities hold?

Lemma (Drineas-Kannan-Mahoney, 2006)

For input matrices M and N, suppose the DKM algorithm makes s samples and outputs matrix C. Then for all indices i, j:

$$E[C_{ij}] = (MN)_{ij} \quad \text{and} \quad \mathrm{Var}[C_{ij}] = \frac{1}{s}\left(n\sum_{k=1}^{n} M_{ik}^2 N_{kj}^2 - (MN)_{ij}^2\right).$$

We know how each *individual entry* of C deviates from its value in MN.

Q: How “far” then is the full matrix C from MN?

Quality of approximation

Theorem (Drineas-Kannan-Mahoney, 2006)

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \frac{1}{s}\left(n\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2^2\left\|N_{(k)}\right\|_2^2 - \|MN\|_{\mathrm{F}}^2\right).$$

Quality of approximation

Theorem (Drineas-Kannan-Mahoney, 2006)

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \frac{1}{s}\left(n\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2^2\left\|N_{(k)}\right\|_2^2 - \|MN\|_{\mathrm{F}}^2\right).$$

Proof. Observe that

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \sum_{i=1}^{m}\sum_{j=1}^{p} E\left[(MN - C)_{ij}^2\right] = \sum_{i=1}^{m}\sum_{j=1}^{p}\mathrm{Var}[C_{ij}].$$

Quality of approximation

Theorem (Drineas-Kannan-Mahoney, 2006)

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \frac{1}{s}\left(n\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2^2 \left\|N_{(k)}\right\|_2^2 - \|MN\|_{\mathrm{F}}^2\right).$$

Proof. Observe that

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \sum_{i=1}^{m}\sum_{j=1}^{p} E\left[(MN - C)_{ij}^2\right] = \sum_{i=1}^{m}\sum_{j=1}^{p} \mathrm{Var}[C_{ij}].$$

Plugging in the bounds on $\mathrm{Var}[C_{ij}]$ from previous lemma:

$$\begin{aligned} E\left[\|MN - C\|_{\mathrm{F}}^2\right] &= \sum_{i=1}^{m}\sum_{j=1}^{p}\left(\frac{1}{s}\left(n\sum_{k=1}^{n} M_{ik}^2 N_{kj}^2 - (MN)_{ij}^2\right)\right) \\ &= \frac{1}{s}\left(n\sum_{k=1}^{n}\left(\sum_{i=1}^{m} M_{ik}^2\right)\left(\sum_{j=1}^{p} N_{kj}^2\right) - \|MN\|_{\mathrm{F}}^2\right) \end{aligned}$$

from which claim follows.

Optimizing further

Theorem (Drineas-Kannan-Mahoney, 2006)

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \frac{1}{s}\left(n\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2^2\left\|N_{(k)}\right\|_2^2 - \|MN\|_{\mathrm{F}}^2\right) \qquad (**).$$

Q: In iteration t, we *uniformly* sample column/row pair $M^{(k_t)}$ and $N_{(k_t)}$.

Optimizing further

Theorem (Drineas-Kannan-Mahoney, 2006)

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \frac{1}{s}\left(n\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2^2\|N_{(k)}\|_2^2 - \|MN\|_{\mathrm{F}}^2\right) \qquad (**).$$

Q: In iteration t, we *uniformly* sample column/row pair $M^{(k_t)}$ and $N_{(k_t)}$.

Obs: But if a column/row has large norm, it has more "impact" on MN.

Idea: Sample columns/rows with larger norm with larger probability. Set:

Optimizing further

Theorem (Drineas-Kannan-Mahoney, 2006)

$$E\left[\|MN - C\|_F^2\right] = \frac{1}{s}\left(n\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2^2\left\|N_{(k)}\right\|_2^2 - \|MN\|_F^2\right) \qquad (**).$$

Q: In iteration t, we *uniformly* sample column/row pair $M^{(k_t)}$ and $N_{(k_t)}$.

Obs: But if a column/row has large norm, it has more "impact" on MN.

Idea: Sample columns/rows with larger norm with larger probability. Set:

$$\Pr(\text{picking index } k_t \text{ in iteration } t) = \frac{\left\|M^{(k)}\right\|_2\left\|N_{(k)}\right\|_2}{\sum_{l=1}^{n}\left\|M^{(l)}\right\|_2\left\|N_{(l)}\right\|_2}.$$

Optimizing further

Theorem (Drineas-Kannan-Mahoney, 2006)

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \frac{1}{s}\left(n\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2^2 \|N_{(k)}\|_2^2 - \|MN\|_{\mathrm{F}}^2\right) \qquad (**).$$

Q: In iteration t, we *uniformly* sample column/row pair $M^{(k_t)}$ and $N_{(k_t)}$.

Obs: But if a column/row has large norm, it has more "impact" on MN.

Idea: Sample columns/rows with larger norm with larger probability. Set:

$$\Pr(\text{picking index } k_t \text{ in iteration } t) = \frac{\left\|M^{(k)}\right\|_2 \left\|N_{(k)}\right\|_2}{\sum_{l=1}^{n}\left\|M^{(l)}\right\|_2 \left\|N_{(l)}\right\|_2}.$$

This distribution turns out to be *optimal*, i.e. minimizes $E\left[\|MN - C\|_{\mathrm{F}}^2\right]$:

$$E\left[\|MN - C\|_{\mathrm{F}}^2\right] = \frac{1}{s}\left(\sum_{k=1}^{n}\left\|M^{(k)}\right\|_2 \left\|N_{(k)}\right\|_2\right)^2 - \frac{1}{s}\|MN\|_{\mathrm{F}}^2 \qquad (***).$$

Ex. Prove $(***) \leq (**)$. (Hint: Use Cauchy-Schwarz inequality.)

Outline

1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

Goals of section

- More practice with randomization (life lesson: don't gamble)
- Practice solving recurrence relations
- Real world applications of matrices (life lesson: get rich)

Outline

1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

Roulette

- Can bet 1€ per turn on a color, either red or black.
- If ball lands on your color in that turn, win 1€; else, lose 1€.
- Suppose we start with 100€.
- Q: What is the probability we win 100€ before going bankrupt?

Roulette

- Can bet 1€ per turn on a color, either red or black.
- If ball lands on your color in that turn, win 1€; else, lose 1€.
- Suppose we start with 100€.
- Q: What is the probability we win 100€ before going bankrupt?
- Intuition: Since prob. winning in a turn is $18/38 \approx 0.473$ (why?), odds of winning 100€ shouldn't be too far from $1/2$?

Roulette

- Can bet 1€ per turn on a color, either red or black.
- If ball lands on your color in that turn, win 1€; else, lose 1€.
- Suppose we start with 100€.
- Q: What is the probability we win 100€ before going bankrupt?
- Intuition: Since prob. winning in a turn is $18/38 \approx 0.473$ (why?), odds of winning 100€ shouldn't be too far from $1/2$? (Ha ha.)

Let's formalize this

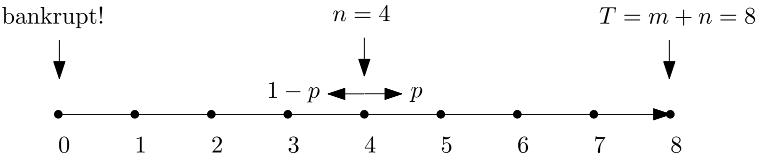
Gambler's ruin

- Start with n€, and make sequence of bets.
- For each bet, win 1€ w.p. p, lose €1 w.p. $1 - p$.
- We *lose* if run out of money, i.e. go bankrupt.
- We *win* if we earn an additional m€, i.e. we stop with $T = n + m$ Euros.

Let's formalize this

Gambler's ruin

- Start with n€, and make sequence of bets.
- For each bet, win 1€ w.p. p, lose €1 w.p. $1 - p$.
- We *lose* if run out of money, i.e. go bankrupt.
- We *win* if we earn an additional m€, i.e. we stop with $T = n + m$ Euros.



- Can be viewed as a 1-dimensional *random walk*.
- Move right 1 step with probability p, left 1 step with probability $1 - p$.

Let's formalize this

Gambler's ruin

- Start with n€, and make sequence of bets.
- For each bet, win 1€ w.p. p, lose 1€ w.p. $1 - p$.
- We *lose* if run out of money, i.e. go bankrupt.
- We *win* if we earn an additional m€, i.e. have $T = n + m$ Euros total.

Let's formalize this

Gambler's ruin

- Start with n€, and make sequence of bets.
- For each bet, win 1€ w.p. p, lose 1€ w.p. $1 - p$.
- We *lose* if run out of money, i.e. go bankrupt.
- We *win* if we earn an additional m€, i.e. have $T = n + m$ Euros total.

- Let W be event that we win before we lose.
- Let D_t be random variable denoting $\#$ of Euros we have at time t.

Let's formalize this

Gambler's ruin

- Start with n€, and make sequence of bets.
- For each bet, win 1€ w.p. p, lose 1€ w.p. $1 - p$.
- We *lose* if run out of money, i.e. go bankrupt.
- We *win* if we earn an additional m€, i.e. have $T = n + m$ Euros total.

- Let W be event that we win before we lose.
- Let D_t be random variable denoting $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

More basic probability theory

A *sample space* Ω is an arbitrary set, the subsets of which are *events*.

Ex. If we flip coin 4 times, what is sample space of all possible outcomes?

More basic probability theory

A *sample space* Ω is an arbitrary set, the subsets of which are *events*.

Ex. If we flip coin 4 times, what is sample space of all possible outcomes?

Definition (Conditional probability ($\Pr(A \mid B)$))

For events A and B from a sample space Ω,

$$\Pr(A \wedge B) = \Pr(A \mid B)\Pr(B),$$

where $\wedge$ denotes *AND*.

More basic probability theory

A *sample space* Ω is an arbitrary set, the subsets of which are *events*.

Ex. If we flip coin 4 times, what is sample space of all possible outcomes?

Definition (Conditional probability ($\Pr(A \mid B)$))

For events A and B from a sample space Ω,

$$\Pr(A \wedge B) = \Pr(A \mid B)\Pr(B),$$

where $\wedge$ denotes *AND*.

Law of total probability

Let $B_1, \ldots, B_n$ partition a sample space Ω. Then for any event A,

$$\Pr(A) = \sum_{i=1}^{n} \Pr(A \mid B_i)\Pr(B_i).$$

Let's formalize this

Gambler's ruin

- Start with n€, and make sequence of bets.
- For each bet, win 1€ w.p. p, lose 1€ w.p. $1 - p$.
- We *lose* if run out of money, i.e. go bankrupt.
- We *win* if we earn an additional m€, i.e. have $T = n + m$ Euros total.

- Let W be event that we win before we lose.
- Let D_t be random variable denoting $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Let's formalize this

Recall W is event we win before we lose, D_t is $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Proof. Cases of $n = 0, n = T$ trivial, so assume $0 < n < T$. Let E_1, E_2 be event that first bet is a win or lose, respectively.

$$P_n \quad = \quad \Pr(W \mid D_0 = n)$$

Let's formalize this

Recall W is event we win before we lose, D_t is $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Proof. Cases of $n = 0, n = T$ trivial, so assume $0 < n < T$. Let E_1, E_2 be event that first bet is a win or lose, respectively.

$$\begin{aligned} P_n &= \Pr(W \mid D_0 = n) \\ &= \Pr(W \wedge E_1 \mid D_0 = n) + \Pr(W \wedge E_2 \mid D_0 = n) \text{ (why?)} \end{aligned}$$

Let's formalize this

Recall W is event we win before we lose, D_t is $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Proof. Cases of $n = 0, n = T$ trivial, so assume $0 < n < T$. Let E_1, E_2 be event that first bet is a win or lose, respectively.

$$\begin{aligned} P_n &= \Pr(W \mid D_0 = n) \\ &= \Pr(W \wedge E_1 \mid D_0 = n) + \Pr(W \wedge E_2 \mid D_0 = n) \text{ (why?)} \\ &= \Pr(E_1 \mid D_0 = n)\Pr(W \mid E_1 \wedge D_0 = n) + \Pr(E_2 \mid D_0 = n)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (?)} \end{aligned}$$

Let's formalize this

Recall W is event we win before we lose, D_t is $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Proof. Cases of $n = 0$, $n = T$ trivial, so assume $0 < n < T$. Let E_1, E_2 be event that first bet is a win or lose, respectively.

$$\begin{aligned} P_n &= \Pr(W \mid D_0 = n) \\ &= \Pr(W \wedge E_1 \mid D_0 = n) + \Pr(W \wedge E_2 \mid D_0 = n) \text{ (why?)} \\ &= \Pr(E_1 \mid D_0 = n)\Pr(W \mid E_1 \wedge D_0 = n) + \Pr(E_2 \mid D_0 = n)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (?)} \\ &= p\Pr(W \mid E_1 \wedge D_0 = n) + (1-p)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (why?)} \end{aligned}$$

Let's formalize this

Recall W is event we win before we lose, D_t is # of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Proof. Cases of $n = 0$, $n = T$ trivial, so assume $0 < n < T$. Let E_1, E_2 be event that first bet is a win or lose, respectively.

$$\begin{aligned} P_n &= \Pr(W \mid D_0 = n) \\ &= \Pr(W \wedge E_1 \mid D_0 = n) + \Pr(W \wedge E_2 \mid D_0 = n) \text{ (why?)} \\ &= \Pr(E_1 \mid D_0 = n)\Pr(W \mid E_1 \wedge D_0 = n) + \Pr(E_2 \mid D_0 = n)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (?)} \\ &= p\Pr(W \mid E_1 \wedge D_0 = n) + (1-p)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (why?)} \\ &= p\Pr(W \mid D_1 = n+1) + (1-p)\Pr(W \mid D_1 = n-1) \text{ (why?)} \end{aligned}$$

Let's formalize this

Recall W is event we win before we lose, D_t is $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Proof. Cases of $n = 0$, $n = T$ trivial, so assume $0 < n < T$. Let E_1, E_2 be event that first bet is a win or lose, respectively.

$$\begin{aligned} P_n &= \Pr(W \mid D_0 = n) \\ &= \Pr(W \wedge E_1 \mid D_0 = n) + \Pr(W \wedge E_2 \mid D_0 = n) \text{ (why?)} \\ &= \Pr(E_1 \mid D_0 = n)\Pr(W \mid E_1 \wedge D_0 = n) + \Pr(E_2 \mid D_0 = n)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (?)} \\ &= p\Pr(W \mid E_1 \wedge D_0 = n) + (1-p)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (why?)} \\ &= p\Pr(W \mid D_1 = n+1) + (1-p)\Pr(W \mid D_1 = n-1) \text{ (why?)} \\ &= p\Pr(W \mid D_0 = n+1) + (1-p)\Pr(W \mid D_0 = n-1) \text{ (why?)} \end{aligned}$$

Let's formalize this

Recall W is event we win before we lose, D_t is $\#$ of Euros we have at time t.

Claim

Let $P_n = \Pr(W \mid D_0 = n)$ be probability of W, *given that* start with n€. Then:

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = T \\ pP_{n+1} + (1-p)P_{n-1} & \text{if } 0 < n < T. \end{cases}$$

Proof. Cases of $n = 0$, $n = T$ trivial, so assume $0 < n < T$. Let E_1, E_2 be event that first bet is a win or lose, respectively.

$$\begin{aligned} P_n &= \Pr(W \mid D_0 = n) \\ &= \Pr(W \wedge E_1 \mid D_0 = n) + \Pr(W \wedge E_2 \mid D_0 = n) \text{ (why?)} \\ &= \Pr(E_1 \mid D_0 = n)\Pr(W \mid E_1 \wedge D_0 = n) + \Pr(E_2 \mid D_0 = n)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (?)} \\ &= p\Pr(W \mid E_1 \wedge D_0 = n) + (1-p)\Pr(W \mid E_2 \wedge D_0 = n) \text{ (why?)} \\ &= p\Pr(W \mid D_1 = n+1) + (1-p)\Pr(W \mid D_1 = n-1) \text{ (why?)} \\ &= p\Pr(W \mid D_0 = n+1) + (1-p)\Pr(W \mid D_0 = n-1) \text{ (why?)} \\ &= pP_{n+1} + (1-p)P_{n-1}. \end{aligned}$$

So if we start with n€, we win with probability $P_n = pP_{n+1} + (1-p)P_{n-1}$, or

$$pP_{n+1} - P_n + (1-p)P_{n-1} = 0.$$

So if we start with n€, we win with probability $P_n = pP_{n+1} + (1-p)P_{n-1}$, or

$$pP_{n+1} - P_n + (1-p)P_{n-1} = 0.$$

This is a *linear homogeneous recurrence* with $P_0 = 0$ and $P_T = 1$.

Let's solve to get closed form for P_n, and determine odds of winning Roulette.

Idea: Use *characteristic root* technique.

Characteristic root technique

Consider recurrence relation $a_n + \alpha a_{n-1} + \beta a_{n-2} = 0$.

Characteristic root technique

Consider recurrence relation $a_n + \alpha a_{n-1} + \beta a_{n-2} = 0$.

Its *characteristic polynomial* is $x^2 + \alpha x + \beta$.

Fact 1

Suppose the characteristic polynomial has roots r_1, r_2 (i.e. solutions to *characteristic equation* $x^2 + \alpha x + \beta = 0$). Then:

Characteristic root technique

Consider recurrence relation $a_n + \alpha a_{n-1} + \beta a_{n-2} = 0$.

Its *characteristic polynomial* is $x^2 + \alpha x + \beta$.

Fact 1

Suppose the characteristic polynomial has roots r_1, r_2 (i.e. solutions to *characteristic equation* $x^2 + \alpha x + \beta = 0$). Then:

- If $r_1 \neq r_2$, there exists constants a, b such that $a_n = ar_1^n + br_2^n$.

Characteristic root technique

Consider recurrence relation $a_n + \alpha a_{n-1} + \beta a_{n-2} = 0$.

Its *characteristic polynomial* is $x^2 + \alpha x + \beta$.

Fact 1

Suppose the characteristic polynomial has roots r_1, r_2 (i.e. solutions to *characteristic equation* $x^2 + \alpha x + \beta = 0$). Then:

- If $r_1 \neq r_2$, there exists constants a, b such that $a_n = ar_1^n + br_2^n$.
- If $r_1 = r_2$, there exists constants a, b such that $a_n = ar_1^n + bnr_2^n$.

Ex. Let $a_n = a_{n-1} + a_{n-2}$, with $a_0 = 0$ and $a_1 = 1$. Which famous recurrence is this? Solve this recurrence.

Characteristic root technique

Consider recurrence relation $a_n + \alpha a_{n-1} + \beta a_{n-2} = 0$.

Its *characteristic polynomial* is $x^2 + \alpha x + \beta$.

Fact 1

Suppose the characteristic polynomial has roots r_1, r_2 (i.e. solutions to *characteristic equation* $x^2 + \alpha x + \beta = 0$). Then:

- If $r_1 \neq r_2$, there exists constants a, b such that $a_n = ar_1^n + br_2^n$.
- If $r_1 = r_2$, there exists constants a, b such that $a_n = ar_1^n + bnr_2^n$.

Ex. Let $a_n = a_{n-1} + a_{n-2}$, with $a_0 = 0$ and $a_1 = 1$. Which famous recurrence is this? Solve this recurrence.

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

Solving the recurrence

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

Solving the recurrence

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

By quadratic formula, $x = \frac{1 \pm \sqrt{1-4p(1-p)}}{2p}$, i.e. have roots $x = \frac{1-p}{p}$ and $x = 1$.

Solving the recurrence

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

By quadratic formula, $x = \frac{1 \pm \sqrt{1-4p(1-p)}}{2p}$, i.e. have roots $x = \frac{1-p}{p}$ and $x = 1$.

- Case 1: $p \neq 1/2$, i.e. distinct roots. By Fact 1, $\exists$ constants a, b s.t.

$$P_n = a\left(\frac{1-p}{p}\right)^n + b \leq \left(\frac{p}{1-p}\right)^m.$$

Ex. Use initial conditions $P_0 = 0, P_T = 1$ to figure out a and b. Then, prove red inequality.

Solving the recurrence

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

By quadratic formula, $x = \frac{1 \pm \sqrt{1-4p(1-p)}}{2p}$, i.e. have roots $x = \frac{1-p}{p}$ and $x = 1$.

- Case 1: $p \neq 1/2$, i.e. distinct roots. By Fact 1, $\exists$ constants a, b s.t.

$$P_n = a\left(\frac{1-p}{p}\right)^n + b \leq \left(\frac{p}{1-p}\right)^m.$$

  Ex. Use initial conditions $P_0 = 0, P_T = 1$ to figure out a and b. Then, prove red inequality.

  Conclusion: For Roulette, $p = \frac{18}{38} \neq \frac{1}{2}$. Thus, $P_n \leq \left(\frac{p}{1-p}\right)^m \leq \frac{9}{10}^m$.

  - Probability of winning just 100€ (i.e. $m = 100$) is less than $\frac{1}{37648}$!
  - Note: P_n is *independent* of how much money, n, start with.

  Ex. For what range of p is $\lim_{m\to\infty} P_n = 0$?

(Google's disappointed face emoji)

Solving the recurrence

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

By quadratic formula, $x = \frac{1 \pm \sqrt{1-4p(1-p)}}{2p}$, i.e. have roots $x = \frac{1-p}{p}$ and $x = 1$.

Solving the recurrence

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

By quadratic formula, $x = \frac{1 \pm \sqrt{1-4p(1-p)}}{2p}$, i.e. have roots $x = \frac{1-p}{p}$ and $x = 1$.

- Case 2: $p = 1/2$, i.e. same root. By Fact 1,

$$P_n = an + b = \frac{n}{T} = \frac{n}{n+m}.$$

Ex. Use initial conditions $P_0 = 0$, $P_T = 1$ to figure out a and b. Then, prove red equality.

Solving the recurrence

In our setting, we have $pP_{n+1} - P_n + (1-p)P_{n-1} = 0$.

Need to solve for roots of characteristic equation $px^2 - x + (1-p) = 0$.

By quadratic formula, $x = \frac{1 \pm \sqrt{1-4p(1-p)}}{2p}$, i.e. have roots $x = \frac{1-p}{p}$ and $x = 1$.

- Case 2: $p = 1/2$, i.e. same root. By Fact 1,

$$P_n = an + b = \frac{n}{T} = \frac{n}{n+m}.$$

Ex. Use initial conditions $P_0 = 0, P_T = 1$ to figure out a and b. Then, prove red equality.

Conclusion: When the game is fair ($p = 1/2$), odds of winning are what you expect — the closer you start (n) to your goal ($T = n + m$), the more likely you are to win an additional m€!

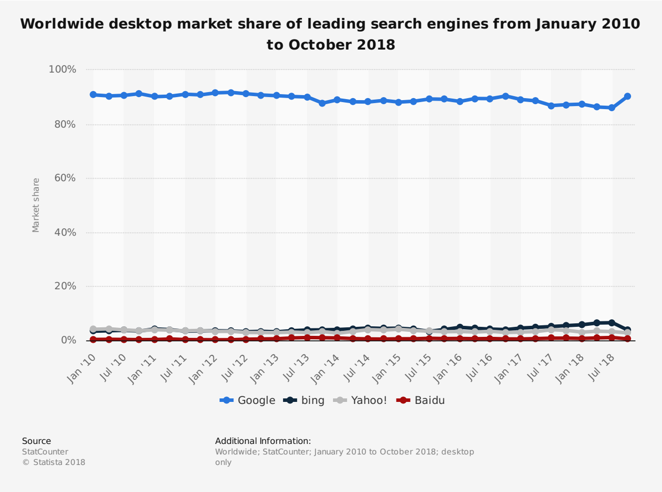
(Google's thinking face emoji)

Outline

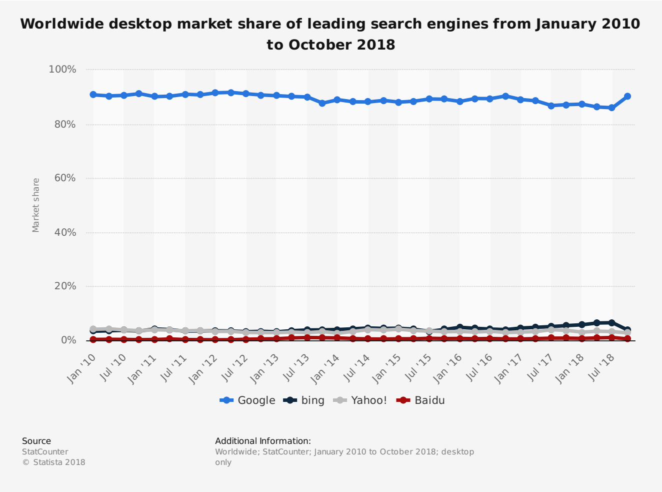
1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

Now let's take random walks beyond 1D and throw in matrices.

Google search



Google search



Conclusion: Google has strong impact on which information is accessed.

With great power comes great responsibility. . .

With great power comes great responsibility. . .

Q: How does Google decide which websites are more important than others?

PageRank algorithm

- Named after Larry Page (together with Sergey Brin, founded Google)
- Ranks webpages by importance
- Assumption: Pages with more links to them are "more important"
- L. Page, S. Brin, R. Motwani, T. Winograd. "The PageRank citation ranking: Bringing order to the Web", 1999.

Idea sketch (simplified)

Suppose internet consists of N webpages.

Imagine a random websurfer, who repeatedly does the following:

1. Pick a uniformly random link from current page.
2. Follow the link.

Idea sketch (simplified)

Suppose internet consists of N webpages.

Imagine a random websurfer, who repeatedly does the following:

1. Pick a uniformly random link from current page.
2. Follow the link.

Intuition: Pages with “many” incoming links get visited “often” by websurfer.

Idea sketch (simplified)

Suppose internet consists of N webpages.

Imagine a random websurfer, who repeatedly does the following:

1. Pick a uniformly random link from current page.
2. Follow the link.

Intuition: Pages with “many” incoming links get visited “often” by websurfer.

Punchline: After “sufficiently long time”, the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

Idea sketch (simplified)

Suppose internet consists of N webpages.

Imagine a random websurfer, who repeatedly does the following:

1. Pick a uniformly random link from current page.
2. Follow the link.

Intuition: Pages with “many” incoming links get visited “often” by websurfer.

Punchline: After “sufficiently long time”, the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

Observation: Websurfer is doing a random walk on the world wide web!

Encoding random walks into matrices

Visualize the world wide web as a directed graph $G(V, E)$:

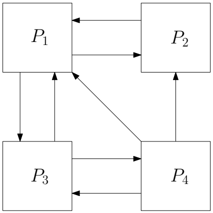
- Each vertex $v \in V$ represents a webpage. Recall $|V| = N$.
- $(u, v) \in E$ if there is a link from page u to page v.

Encoding random walks into matrices

Visualize the world wide web as a directed graph $G(V, E)$:

- Each vertex $v \in V$ represents a webpage. Recall $|V| = N$.
- $(u, v) \in E$ if there is a link from page u to page v.

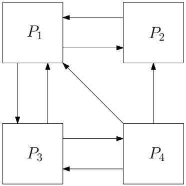
Ex. Consider directed graph $G = (V, E)$ with $V = \{A, B, C, D\}$:



The adjacency matrix A for G is

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

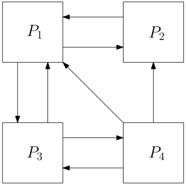
Encoding random walks into matrices



The adjacency matrix W for G is

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

Encoding random walks into matrices

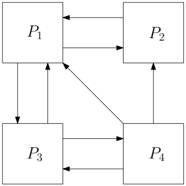


The adjacency matrix W for G is

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

- If websurfer starts at page P_1, its state encoded by vector $\mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$.

Encoding random walks into matrices

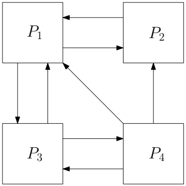


The adjacency matrix W for G is

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

- If websurfer starts at page P_1, its state encoded by vector $\mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$.
- After 1 step, moves to P_2 or P_3, with prob. $1/2$ each. New state $\mathbf{p_1} = \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$.

Encoding random walks into matrices



The adjacency matrix W for G is

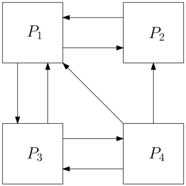
$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

- If websurfer starts at page P_1, its state encoded by vector $\mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$.
- After 1 step, moves to P_2 or P_3, with prob. $1/2$ each. New state $\mathbf{p_1} = \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$.

Observation

View $\mathbf{p_i}$ as a *distribution* encoding probability that surfer at particular page after step i.

Encoding random walks into matrices



The adjacency matrix W for G is

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

- If websurfer starts at page P_1, its state encoded by vector $\mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$.
- After 1 step, moves to P_2 or P_3, with prob. $1/2$ each. New state $\mathbf{p_1} = \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$.

Observation

View $\mathbf{p_i}$ as a *distribution* encoding probability that surfer at particular page after step i.

Q: Can we encode change in probabilities in each step by matrix multiplication?

Encoding random walks into matrices

Recall: $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad \mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{p_1} = \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$

- Could $A\mathbf{p}_0 = \mathbf{p}_1$? Ex. Work this out.

Encoding random walks into matrices

Recall: $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$ $\quad \mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ $\quad \mathbf{p_1} = \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$

- Could $A\mathbf{p}_0 = \mathbf{p}_1$? Ex. Work this out.
- Try taking transpose: $A^T\mathbf{p}_0 = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}^T$. Missing normalization...

Encoding random walks into matrices

Recall: $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$ $\quad \mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ $\quad \mathbf{p_1} = \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$

- Could $A\mathbf{p}_0 = \mathbf{p}_1$? Ex. Work this out.
- Try taking transpose: $A^T\mathbf{p}_0 = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}^T$. Missing normalization...
- Normalize each row of A by its out-degree (i.e. number of neighbors):

$$\widehat{A} = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 \\ 1 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 \\ 1/3 & 1/3 & 1/3 & 0 \end{pmatrix}, \quad M := \widehat{A}^T = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}$$

Now $\widehat{A}^T\mathbf{p}_0 = M\mathbf{p}_0 = \mathbf{p}_1$!

Multiplying by M updates surfer's current distribution via 1 step of random walk!

Encoding random walks into matrices

Recall: $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad \mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{p_1} = \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$

- Could $A\mathbf{p}_0 = \mathbf{p}_1$? Ex. Work this out.
- Try taking transpose: $A^T\mathbf{p}_0 = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}^T$. Missing normalization...
- Normalize each row of A by its out-degree (i.e. number of neighbors):

$$\widehat{A} = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 \\ 1 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 \\ 1/3 & 1/3 & 1/3 & 0 \end{pmatrix}, \quad M := \widehat{A}^T = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}$$

Now $\widehat{A}^T\mathbf{p}_0 = M\mathbf{p}_0 = \mathbf{p}_1$!

Multiplying by M updates surfer's current distribution via 1 step of random walk!

Thus, after k steps, surfer's distribution is $\mathbf{p_k} = M^k\mathbf{p_0}$.

Idea sketch (simplified)

Suppose internet consists of N webpages.

Imagine a random websurfer, who repeatedly does the following:

1. Pick a uniformly random link from current page.
2. Follow the link.

Intuition: Pages with "many" incoming links get visited "often" by websurfer.

Punchline: After "sufficiently long time", the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

Observation: Websurfer is doing a random walk on the world wide web!

Defining PageRank more formally

Punchline: After "sufficiently long time", the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

Defining PageRank more formally

Punchline: After "sufficiently long time", the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

PageRank:

- Recall if starting distribution is $\mathbf{p}_0$, after k steps have distribution $\mathbf{p}_k = M^k\mathbf{p}_0$.

Defining PageRank more formally

Punchline: After "sufficiently long time", the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

PageRank:

- Recall if starting distribution is $\mathbf{p}_0$, after k steps have distribution $\mathbf{p}_k = M^k\mathbf{p}_0$.
- A *steady state* would be a $\mathbf{p}_i$ such that $\mathbf{p}_{i+1} = M\mathbf{p}_i = \mathbf{p}_i$, i.e. probability to be in any particular webpage no longer changes.

Defining PageRank more formally

Punchline: After "sufficiently long time", the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

PageRank:

- Recall if starting distribution is $\mathbf{p}_0$, after k steps have distribution $\mathbf{p}_k = M^k\mathbf{p}_0$.
- A *steady state* would be a $\mathbf{p}_i$ such that $\mathbf{p}_{i+1} = M\mathbf{p}_i = \mathbf{p}_i$, i.e. probability to be in any particular webpage no longer changes.
- The wth entry of $\mathbf{p}_i$, corresponding to webpage w, is *PageRank* of w.

Defining PageRank more formally

Punchline: After "sufficiently long time", the probability $\Pr(w)$ that surfer is on any particular webpage w approaches a *steady state*, denoted $q(w)$.

The probability $q(w)$ is the *PageRank* for w.

PageRank:

- Recall if starting distribution is $\mathbf{p}_0$, after k steps have distribution $\mathbf{p}_k = M^k\mathbf{p}_0$.
- A *steady state* would be a $\mathbf{p}_i$ such that $\mathbf{p}_{i+1} = M\mathbf{p}_i = \mathbf{p}_i$, i.e. probability to be in any particular webpage no longer changes.
- The wth entry of $\mathbf{p}_i$, corresponding to webpage w, is *PageRank* of w.

Observation: Note that $M\mathbf{p}_i = \mathbf{p}_i$ is just an eigenvalue equation!

Aside: Eigenvalues and eigenvectors

The PageRank vector is a distribution $\mathbf{p}_i$ satisfying $M^k \mathbf{p}_i = \mathbf{p}_i$.

Thus, want to find eigenvector $\mathbf{p}_i$ of M with eigenvalue 1.

Aside: Eigenvalues and eigenvectors

The PageRank vector is a distribution $\mathbf{p}_i$ satisfying $M^k\mathbf{p}_i = \mathbf{p}_i$.

Thus, want to find eigenvector $\mathbf{p}_i$ of M with eigenvalue 1.

Eigenvalues and eigenvectors

For $A \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n$, say $\mathbf{v}$ is an *eigenvector* of A with *eigenvalue* $\lambda \in \mathbb{R}$ if

$$A\mathbf{v} = \lambda\mathbf{v}. \tag{1}$$

Aside: Eigenvalues and eigenvectors

The PageRank vector is a distribution $\mathbf{p}_i$ satisfying $M^k\mathbf{p}_i = \mathbf{p}_i$.

Thus, want to find eigenvector $\mathbf{p}_i$ of M with eigenvalue 1.

Eigenvalues and eigenvectors

For $A \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n$, say $\mathbf{v}$ is an *eigenvector* of A with *eigenvalue* $\lambda \in \mathbb{R}$ if

$$A\mathbf{v} = \lambda\mathbf{v}. \tag{1}$$

How to find eigenvectors and eigenvalues?

Aside: Eigenvalues and eigenvectors

The PageRank vector is a distribution $\mathbf{p}_i$ satisfying $M^k \mathbf{p}_i = \mathbf{p}_i$.

Thus, want to find eigenvector $\mathbf{p}_i$ of M with eigenvalue 1.

Eigenvalues and eigenvectors

For $A \in \mathbb{R}^{n \times n}$ and $\mathbf{v} \in \mathbb{R}^n$, say $\mathbf{v}$ is an *eigenvector* of A with *eigenvalue* $\lambda \in \mathbb{R}$ if

$$A\mathbf{v} = \lambda \mathbf{v}. \quad (1)$$

How to find eigenvectors and eigenvalues?

- "Traditional" method:
  - Solve for roots of characteristic equation $\det(A - \lambda I) = 0$ to obtain eigenvalues λ.
  - Substitute λ into Equation (1) to obtain a linear system of equations.
  - Solve the linear system to obtain $\mathbf{v}$.

Aside: Eigenvalues and eigenvectors

The PageRank vector is a distribution $\mathbf{p}_i$ satisfying $M^k\mathbf{p}_i = \mathbf{p}_i$.

Thus, want to find eigenvector $\mathbf{p}_i$ of M with eigenvalue 1.

Eigenvalues and eigenvectors

For $A \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n$, say $\mathbf{v}$ is an *eigenvector* of A with *eigenvalue* $\lambda \in \mathbb{R}$ if

$$A\mathbf{v} = \lambda\mathbf{v}. \tag{1}$$

How to find eigenvectors and eigenvalues?

- "Traditional" method:
  - Solve for roots of characteristic equation $\det(A - \lambda I) = 0$ to obtain eigenvalues λ.
  - Substitute λ into Equation (1) to obtain a linear system of equations.
  - Solve the linear system to obtain $\mathbf{v}$.
- Power method (Von Mises, 1929):
  - Start with some vector $\mathbf{v}_0$.
  - In iteration k, set $\mathbf{v}_{k+1} = \frac{A\mathbf{v}_k}{\|A\mathbf{v}_k\|}$.

Aside: Eigenvalues and eigenvectors

The PageRank vector is a distribution $\mathbf{p}_i$ satisfying $M^k\mathbf{p}_i = \mathbf{p}_i$.

Thus, want to find eigenvector $\mathbf{p}_i$ of M with eigenvalue 1.

Eigenvalues and eigenvectors

For $A \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n$, say $\mathbf{v}$ is an *eigenvector* of A with *eigenvalue* $\lambda \in \mathbb{R}$ if

$$A\mathbf{v} = \lambda\mathbf{v}. \quad (1)$$

How to find eigenvectors and eigenvalues?

- "Traditional" method:
  - Solve for roots of characteristic equation $\det(A - \lambda I) = 0$ to obtain eigenvalues λ.
  - Substitute λ into Equation (1) to obtain a linear system of equations.
  - Solve the linear system to obtain $\mathbf{v}$.
- Power method (Von Mises, 1929):
  - Start with some vector $\mathbf{v}_0$.
  - In iteration k, set $\mathbf{v}_{k+1} = \frac{A\mathbf{v}_k}{\|A\mathbf{v}_k\|}$.

PageRank implements Power method (with $\|\cdot\|$ the 1-norm/Taxicab norm (why?)).

Test case 1

Recall: $M = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}$ $\mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ $\mathbf{p}_k = M^k \mathbf{p}_0.$

Test case 1

Recall: $M = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix} \qquad \mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{p}_k = M^k \mathbf{p}_0.$

Results (via Mathematica):

$\mathbf{p}_0$	(1., 0., 0., 0.)
$\mathbf{p}_1$	(0., 0.5, 0.5, 0.)
$\mathbf{p}_2$	(0.75, 0., 0., 0.25)
$\mathbf{p}_3$	(0.0833333, 0.458333, 0.458333, 0.)
$\mathbf{p}_4$	(0.6875, 0.0416667, 0.0416667, 0.229167)
$\mathbf{p}_5$	(0.138889, 0.420139, 0.420139, 0.0208333)
$\mathbf{p}_6$	(0.637153, 0.0763889, 0.0763889, 0.210069)
$\mathbf{p}_7$	(0.184606, 0.3886, 0.3886, 0.0381944)
$\mathbf{p}_8$	(0.595631, 0.105035, 0.105035, 0.1943)
$\mathbf{p}_9$	(0.222319, 0.362582, 0.362582, 0.0525174)
$\mathbf{p}_{10}$	(0.561379, 0.128665, 0.128665, 0.181291)
$\mathbf{p}_{11}$	(0.253428, 0.34112, 0.34112, 0.0643326)
$\mathbf{p}_{12}$	(0.533124, 0.148158, 0.148158, 0.17056)

Test case 1

Recall: $M = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}$ $\mathbf{p_0} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ $\mathbf{p}_k = M^k \mathbf{p}_0.$

Results (via Mathematica):

$\mathbf{p}_0$	(1., 0., 0., 0.)
$\mathbf{p}_1$	(0., 0.5, 0.5, 0.)
$\mathbf{p}_2$	(0.75, 0., 0., 0.25)
$\mathbf{p}_3$	(0.0833333, 0.458333, 0.458333, 0.)
$\mathbf{p}_4$	(0.6875, 0.0416667, 0.0416667, 0.229167)
$\mathbf{p}_5$	(0.138889, 0.420139, 0.420139, 0.0208333)
$\mathbf{p}_6$	(0.637153, 0.0763889, 0.0763889, 0.210069)
$\mathbf{p}_7$	(0.184606, 0.3886, 0.3886, 0.0381944)
$\mathbf{p}_8$	(0.595631, 0.105035, 0.105035, 0.1943)
$\mathbf{p}_9$	(0.222319, 0.362582, 0.362582, 0.0525174)
$\mathbf{p}_{10}$	(0.561379, 0.128665, 0.128665, 0.181291)
$\mathbf{p}_{11}$	(0.253428, 0.34112, 0.34112, 0.0643326)
$\mathbf{p}_{12}$	(0.533124, 0.148158, 0.148158, 0.17056)

Seems to be converging, but slowly... No unique most important page yet...

Test case 2

Recall: $M = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}$ $\mathbf{p_0} = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$ $\mathbf{p}_k = M^k \mathbf{p}_0.$

Test case 2

Recall: $M = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix}$ $\quad \mathbf{p_0} = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$ $\quad \mathbf{p}_k = M^k \mathbf{p}_0.$

Results (via Mathematica):

$\mathbf{p}_0$	(0.25, 0.25, 0.25, 0.25)
$\mathbf{p}_1$	(0.458333, 0.208333, 0.208333, 0.125)
$\mathbf{p}_2$	(0.354167, 0.270833, 0.270833, 0.104167)
$\mathbf{p}_3$	(0.440972, 0.211806, 0.211806, 0.135417)
$\mathbf{p}_4$	(0.362847, 0.265625, 0.265625, 0.105903)
$\mathbf{p}_5$	(0.433738, 0.216725, 0.216725, 0.132813)
$\mathbf{p}_6$	(0.369358, 0.26114, 0.26114, 0.108362)
$\mathbf{p}_7$	(0.427831, 0.2208, 0.2208, 0.13057)
$\mathbf{p}_8$	(0.374723, 0.257439, 0.257439, 0.1104)
$\mathbf{p}_9$	(0.422958, 0.224161, 0.224161, 0.128719)
$\mathbf{p}_{10}$	(0.379148, 0.254385, 0.254385, 0.112081)
$\mathbf{p}_{11}$	(0.418938, 0.226934, 0.226934, 0.127193)
$\mathbf{p}_{12}$	(0.382799, 0.251867, 0.251867, 0.113467)

Test case 2

Recall: $$M = \begin{pmatrix} 0 & 1 & 1/2 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 1/2 & 0 & 0 & 1/3 \\ 0 & 0 & 1/2 & 0 \end{pmatrix} \qquad \mathbf{p_0} = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} \qquad \mathbf{p}_k = M^k \mathbf{p}_0.$$

Results (via Mathematica):

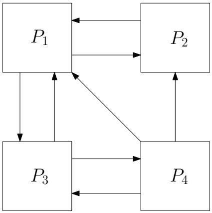
$\mathbf{p}_0$	(0.25, 0.25, 0.25, 0.25)
$\mathbf{p}_1$	(0.458333, 0.208333, 0.208333, 0.125)
$\mathbf{p}_2$	(0.354167, 0.270833, 0.270833, 0.104167)
$\mathbf{p}_3$	(0.440972, 0.211806, 0.211806, 0.135417)
$\mathbf{p}_4$	(0.362847, 0.265625, 0.265625, 0.105903)
$\mathbf{p}_5$	(0.433738, 0.216725, 0.216725, 0.132813)
$\mathbf{p}_6$	(0.369358, 0.26114, 0.26114, 0.108362)
$\mathbf{p}_7$	(0.427831, 0.2208, 0.2208, 0.13057)
$\mathbf{p}_8$	(0.374723, 0.257439, 0.257439, 0.1104)
$\mathbf{p}_9$	(0.422958, 0.224161, 0.224161, 0.128719)
$\mathbf{p}_{10}$	(0.379148, 0.254385, 0.254385, 0.112081)
$\mathbf{p}_{11}$	(0.418938, 0.226934, 0.226934, 0.127193)
$\mathbf{p}_{12}$	(0.382799, 0.251867, 0.251867, 0.113467)

Much better! Singled out P_1 as having largest PageRank.

$\mathbf{p}_{12} = (0.382799, 0.251867, 0.251867, 0.113467)$

Much better! Singled out *A* as having largest PageRank.

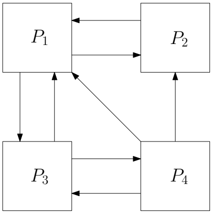
Indeed, P_1 had the largest in-degree:



$\mathbf{p}_{12} = (0.382799, 0.251867, 0.251867, 0.113467)$

Much better! Singled out A as having largest PageRank.

Indeed, P_1 had the largest in-degree:



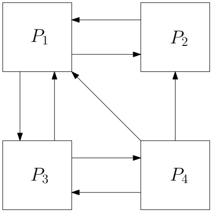
Rate of convergence:

- Seems to depend on starting vector, which is not really surprising.

$\mathbf{p}_{12} = (0.382799, 0.251867, 0.251867, 0.113467)$

Much better! Singled out A as having largest PageRank.

Indeed, P_1 had the largest in-degree:



Rate of convergence:

- Seems to depend on starting vector, which is not really surprising.
- Can we hope to prove rigorous upper bound on number of required iterations to get “close” to PageRank vector?
- Yes, but I’ve sort of been lying to you so far...

- Q: Does a “real” websurfer just follow links all day?

- Q: Does a “real” websurfer just follow links all day?
- A: No! Can enter address in browser’s address bar and jump straight there.
- Let’s try and include this “more realistic” behavior in our model. It will help us prove a convergence bound.

- Q: Does a “real” websurfer just follow links all day?
- A: No! Can enter address in browser’s address bar and jump straight there.
- Let’s try and include this “more realistic” behavior in our model. It will help us prove a convergence bound.

Steps:

1. Define more “realistic” model.
2. Define what we mean by being “close” to the target distribution.
3. “Show” that random walk algorithm converges exponentially quickly to PageRank vector.

More “realistic” model

Suppose internet consists of N webpages.

Fix $0 \leq s \leq 1$. Imagine random websurfer, who repeatedly does following:

1. Flip a biased coin which has probability s of landing HEADS.
2. If get HEADS, follow uniformly random link on current page, i.e. apply M.

More “realistic” model

Suppose internet consists of N webpages.

Fix $0 \leq s \leq 1$. Imagine random websurfer, who repeatedly does following:

1. Flip a biased coin which has probability s of landing HEADS.
2. If get HEADS, follow uniformly random link on current page, i.e. apply M.
3. If get TAILS, go to uniformly random page on internet, i.e. apply $\frac{1}{N}J$ for J the all-ones matrix.

More “realistic” model

Suppose internet consists of N webpages.

Fix $0 \leq s \leq 1$. Imagine random websurfer, who repeatedly does following:

1. Flip a biased coin which has probability s of landing HEADS.
2. If get HEADS, follow uniformly random link on current page, i.e. apply M.
3. If get TAILS, go to uniformly random page on internet, i.e. apply $\frac{1}{N}J$ for J the all-ones matrix.

Q: Why is the right transition matrix for TAILS $\frac{1}{N}J$?

More “realistic” model

Suppose internet consists of N webpages.

Fix $0 \leq s \leq 1$. Imagine random websurfer, who repeatedly does following:

1. Flip a biased coin which has probability s of landing HEADS.
2. If get HEADS, follow uniformly random link on current page, i.e. apply M.
3. If get TAILS, go to uniformly random page on internet, i.e. apply $\frac{1}{N}J$ for J the all-ones matrix.

Q: Why is the right transition matrix for TAILS $\frac{1}{N}J$?

So our new transition matrix is $M(s) = sM + \frac{1-s}{N}J$ (why?).

Quantifying "closeness" of distributions

Given $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ whose entries form probability distributions, how to quantify how "close" these distributions are?

Quantifying "closeness" of distributions

Given $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ whose entries form probability distributions, how to quantify how "close" these distributions are?

Total variation distance

The *total variation distance* between distributions $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ is

$$\|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{N} |p_i - q_i|.$$

Note this is just the Taxicab norm or 1-norm from earlier in slides.

Ex. What is the total variation distance between $\mathbf{p} = (1, 0, 0, 0)^T$ and $\mathbf{q} = (1/4, 1/4, 1/4, 1/4)$?

What does variational distance *mean*?

What does variational distance *mean*?

Suppose we play the following game on some sample space Ω.

1. I flip a fair coin.
2. If I get HEADS, I sample an element $t \in \Omega$ according to **p**.
3. Else, if I get TAILS, I sample an element $t \in \Omega$ according to **q**.
4. I send you t.
5. You try to guess whether I sampled from **p** or **q**.

What does variational distance *mean*?

Suppose we play the following game on some sample space Ω.

1. I flip a fair coin.
2. If I get HEADS, I sample an element $t \in \Omega$ according to $\mathbf{p}$.
3. Else, if I get TAILS, I sample an element $t \in \Omega$ according to $\mathbf{q}$.
4. I send you t.
5. You try to guess whether I sampled from $\mathbf{p}$ or $\mathbf{q}$.

It turns out that your optimal probability of guessing correctly is

$$\frac{1}{2} + \frac{1}{4} \|\mathbf{p} - \mathbf{q}\|_1 .$$

Ex. What is optimal probability of you winning the game for $\mathbf{p} = (1, 0, 0, 0)^T$ and $\mathbf{q} = (1/4, 1/4, 1/4, 1/4)$? Can you think of an optimal guessing strategy for achieving this?

Convergence bounds

Can now bound how quickly we converge to PageRank vector.

Suppose start with arbitrary distribution $\mathbf{p} \in \mathbb{R}^N$ over webpages.

Fact 1. $M(s) = sM + \frac{1-s}{N}J$ has unique PageRank vector, denoted $\mathbf{q}$.

Convergence bounds

Can now bound how quickly we converge to PageRank vector.

Suppose start with arbitrary distribution $\mathbf{p} \in \mathbb{R}^N$ over webpages.

Fact 1. $M(s) = sM + \frac{1-s}{N}J$ has unique PageRank vector, denoted $\mathbf{q}$.

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Convergence bounds

Can now bound how quickly we converge to PageRank vector.

Suppose start with arbitrary distribution $\mathbf{p} \in \mathbb{R}^N$ over webpages.

Fact 1. $M(s) = sM + \frac{1-s}{N}J$ has unique PageRank vector, denoted $\mathbf{q}$.

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Corollary. After $k \geq 1$ iterations, $\left\|M(s)^k\mathbf{p} - \mathbf{q}\right\|_1 \leq s^k\left\|\mathbf{p} - \mathbf{q}\right\|_1$. (why?)

Convergence bounds

Can now bound how quickly we converge to PageRank vector.

Suppose start with arbitrary distribution $\mathbf{p} \in \mathbb{R}^N$ over webpages.

Fact 1. $M(s) = sM + \frac{1-s}{N}J$ has unique PageRank vector, denoted $\mathbf{q}$.

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Corollary. After $k \geq 1$ iterations, $\left\|M(s)^k\mathbf{p} - \mathbf{q}\right\|_1 \leq s^k \left\|\mathbf{p} - \mathbf{q}\right\|_1$. (why?)

Notes:

- In original PageRank paper, $s = 0.85$ was used.

Convergence bounds

Can now bound how quickly we converge to PageRank vector.

Suppose start with arbitrary distribution $\mathbf{p} \in \mathbb{R}^N$ over webpages.

Fact 1. $M(s) = sM + \frac{1-s}{N}J$ has unique PageRank vector, denoted $\mathbf{q}$.

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Corollary. After $k \geq 1$ iterations, $\left\|M(s)^k\mathbf{p} - \mathbf{q}\right\|_1 \leq s^k \left\|\mathbf{p} - \mathbf{q}\right\|_1$. (why?)

Notes:

- In original PageRank paper, $s = 0.85$ was used.
- Since $\left\|\mathbf{p} - \mathbf{q}\right\|_1 \leq 2$ for any unit vectors $\mathbf{p}, \mathbf{q}$ (why?), conclude that we converge exponentially quickly (in k) to PageRank vector $\mathbf{q}$.

Convergence bounds

Can now bound how quickly we converge to PageRank vector.

Suppose start with arbitrary distribution $\mathbf{p} \in \mathbb{R}^N$ over webpages.

Fact 1. $M(s) = sM + \frac{1-s}{N}J$ has unique PageRank vector, denoted $\mathbf{q}$.

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Corollary. After $k \geq 1$ iterations, $\left\|M(s)^k\mathbf{p} - \mathbf{q}\right\|_1 \leq s^k\left\|\mathbf{p} - \mathbf{q}\right\|_1$. (why?)

Notes:

- In original PageRank paper, $s = 0.85$ was used.
- Since $\left\|\mathbf{p} - \mathbf{q}\right\|_1 \leq 2$ for any unit vectors $\mathbf{p}, \mathbf{q}$ (why?), conclude that we converge exponentially quickly (in k) to PageRank vector $\mathbf{q}$.
- Magically, this bound is independent of size of internet, N.

Convergence bounds

Can now bound how quickly we converge to PageRank vector.

Suppose start with arbitrary distribution $\mathbf{p} \in \mathbb{R}^N$ over webpages.

Fact 1. $M(s) = sM + \frac{1-s}{N}J$ has unique PageRank vector, denoted $\mathbf{q}$.

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Corollary. After $k \geq 1$ iterations, $\left\|M(s)^k\mathbf{p} - \mathbf{q}\right\|_1 \leq s^k \left\|\mathbf{p} - \mathbf{q}\right\|_1$.(why?)

Notes:

- In original PageRank paper, $s = 0.85$ was used.
- Since $\left\|\mathbf{p} - \mathbf{q}\right\|_1 \leq 2$ for any unit vectors $\mathbf{p}, \mathbf{q}$ (why?), conclude that we converge exponentially quickly (in k) to PageRank vector $\mathbf{q}$.
- Magically, this bound is independent of size of internet, N.

Ok, so remains to prove Claim 2.

A helpful lemma

Observation. By construction, each column of $M(s)$ is probability vector. Thus, $M(s)$ is a *(left) stochastic matrix*.

A helpful lemma

Observation. By construction, each column of $M(s)$ is probability vector. Thus, $M(s)$ is a *(left) stochastic matrix*.

Lemma (Contractivity of l_1 norm)

For stochastic $A \in \mathbb{R}^{n \times n}$ and $\mathbf{v} \in \mathbb{R}^n$, $\|A\mathbf{v}\|_1 \leq \|\mathbf{v}\|_1$.

A helpful lemma

Observation. By construction, each column of $M(s)$ is probability vector. Thus, $M(s)$ is a *(left) stochastic matrix*.

Lemma (Contractivity of l_1 norm)

For stochastic $A \in \mathbb{R}^{n \times n}$ and $\mathbf{v} \in \mathbb{R}^n$, $\|A\mathbf{v}\|_1 \leq \|\mathbf{v}\|_1$.

Proof.

$$\|A\mathbf{v}\|_1 = \sum_{j=1}^{n} \left| \sum_{k=1}^{n} A_{jk} v_k \right| \quad \text{(def. of } l_1 \text{ norm)}$$

A helpful lemma

Observation. By construction, each column of $M(s)$ is probability vector. Thus, $M(s)$ is a *(left) stochastic matrix*.

Lemma (Contractivity of l_1 norm)

For stochastic $A \in \mathbb{R}^{n \times n}$ and $\mathbf{v} \in \mathbb{R}^n$, $\|A\mathbf{v}\|_1 \leq \|\mathbf{v}\|_1$.

Proof.

$$\|A\mathbf{v}\|_1 = \sum_{j=1}^{n} \left| \sum_{k=1}^{n} A_{jk} v_k \right| \quad \text{(def. of } l_1 \text{ norm)}$$

$$\leq \sum_{k=1}^{n} \sum_{j=1}^{n} A_{jk} \left| v_k \right| \quad \text{(triangle inequality, multiplicativity, } |A_{jk}| = A_{jk}\text{)}$$

A helpful lemma

Observation. By construction, each column of $M(s)$ is probability vector. Thus, $M(s)$ is a *(left) stochastic matrix*.

Lemma (Contractivity of l_1 norm)

For stochastic $A \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n$, $\|A\mathbf{v}\|_1 \leq \|\mathbf{v}\|_1$.

Proof.

$$
\begin{aligned}
\|A\mathbf{v}\|_1 &= \sum_{j=1}^{n}\left|\sum_{k=1}^{n} A_{jk}v_k\right| && \text{(def. of } l_1 \text{ norm)}\\
&\leq \sum_{k=1}^{n}\sum_{j=1}^{n} A_{jk}\left|v_k\right| && \text{(triangle inequality, multiplicativity, } |A_{jk}| = A_{jk}\text{)}\\
&= \sum_{k=1}^{n}\left|v_k\right| && \text{(sum of column entries of } A \text{ is 1)}\\
&= \|\mathbf{v}\|_1 .
\end{aligned}
$$

Proof of Claim 2

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Proof of Claim 2

Claim 2. For all $j \geq 1$, $\left\| M(s)^j \mathbf{p} - \mathbf{q} \right\|_1 \leq s \left\| M(s)^{j-1} \mathbf{p} - \mathbf{q} \right\|_1$.

Proof.

$$\left\| M(s)^j \mathbf{p} - \mathbf{q} \right\|_1 = \left\| M(s)(M(s)^{j-1} \mathbf{p} - \mathbf{q}) \right\|_1 \quad (M(s)\mathbf{q} = \mathbf{q})$$

Proof of Claim 2

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Proof.

$$\begin{aligned}
\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 &= \left\|M(s)(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \qquad (M(s)\mathbf{q} = \mathbf{q}) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q}) + \frac{1-s}{N}J(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \\
&\qquad \left(M(s) = sM + \frac{1-s}{N}J\right)
\end{aligned}$$

Proof of Claim 2

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Proof.

$$\begin{aligned}
\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 &= \left\|M(s)(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad (M(s)\mathbf{q} = \mathbf{q}) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q}) + \frac{1-s}{N}J(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \\
&\qquad \left(M(s) = sM + \frac{1-s}{N}J\right) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad \left(J(M(s)^{j-1}\mathbf{p}) = J\mathbf{q}\right) \text{ (why?)}
\end{aligned}$$

Proof of Claim 2

Claim 2. For all $j \geq 1$, $\left\|M(s)^j \mathbf{p} - \mathbf{q}\right\|_1 \leq s \left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Proof.

$$
\begin{aligned}
\left\|M(s)^j \mathbf{p} - \mathbf{q}\right\|_1 &= \left\|M(s)(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad (M(s)\mathbf{q} = \mathbf{q}) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q}) + \frac{1-s}{N} J(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \\
&\qquad \left(M(s) = sM + \frac{1-s}{N}J\right) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad \left(J(M(s)^{j-1}\mathbf{p}) = J\mathbf{q}\right) \text{ (why?)} \\
&= s\left\|M(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad (\text{absolute homogeneity, } |s| = s)
\end{aligned}
$$

Proof of Claim 2

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Proof.

$$
\begin{aligned}
\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 &= \left\|M(s)(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad (M(s)\mathbf{q} = \mathbf{q}) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q}) + \frac{1-s}{N}J(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \\
&\qquad \left(M(s) = sM + \frac{1-s}{N}J\right) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad \left(J(M(s)^{j-1}\mathbf{p}) = J\mathbf{q}\right) \text{ (why?)} \\
&= s\left\|M(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad \text{(absolute homogeneity, } |s| = s) \\
&\leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1 \quad \text{(contractivity of } l_1 \text{ norm, } M \text{ stochastic).}
\end{aligned}
$$

Proof of Claim 2

Claim 2. For all $j \geq 1$, $\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 \leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1$.

Proof.

$$\begin{aligned}
\left\|M(s)^j\mathbf{p} - \mathbf{q}\right\|_1 &= \left\|M(s)(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad (M(s)\mathbf{q} = \mathbf{q}) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q}) + \frac{1-s}{N}J(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \\
&\qquad \left(M(s) = sM + \frac{1-s}{N}J\right) \\
&= \left\|sM(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad \left(J(M(s)^{j-1}\mathbf{p}) = J\mathbf{q}\right) \text{ (why?)} \\
&= s\left\|M(M(s)^{j-1}\mathbf{p} - \mathbf{q})\right\|_1 \quad \text{(absolute homogeneity, } |s| = s\text{)} \\
&\leq s\left\|M(s)^{j-1}\mathbf{p} - \mathbf{q}\right\|_1 \quad \text{(contractivity of } l_1 \text{ norm, } M \text{ stochastic).}
\end{aligned}$$

Done! We conclude PageRank converges exponentially quickly (in number of iterations, k), to its stationary distribution ($\mathbf{q}$), irrespective of size of the internet (N).

(Google's happy face emoji)

Outline

Goals of section

- Practice working with complex numbers
- Practice working with polynomials
- Introduce Fourier transform and its applications

Outline

Complex numbers

Question: What are the roots of $x^2 + 1 = 0$?

$$i := \sqrt{-1} \text{ and } -i$$

Complex numbers

Question: What are the roots of $x^2 + 1 = 0$?

$$i := \sqrt{-1} \text{ and } -i$$

Selected history

- (Cardano 1545) Considers square roots of negative numbers in solving for roots of cubic polynomials. Calls them "as subtle as [they] are useless".
- (Bombelli 1572) Derives rules for basic arithmetic operations with roots of negative numbers
- (Euler 1707-1783) Introduces symbol i, proves $e^{it} = \cos(t) + i\sin(t)$
- (Wessel 1745-1818, also Gauss 1777-1855) Introduces 2D complex plane
- (Hamilton 1805-1865) Representation of complex numbers as 2-tuples from $\mathbb{R} \times \mathbb{R}$.

Complex numbers

Question: What are the roots of $x^2 + 1 = 0$?

$$i := \sqrt{-1} \text{ and } -i$$

Selected history

- (Cardano 1545) Considers square roots of negative numbers in solving for roots of cubic polynomials. Calls them "as subtle as [they] are useless".
- (Bombelli 1572) Derives rules for basic arithmetic operations with roots of negative numbers
- (Euler 1707-1783) Introduces symbol i, proves $e^{it} = \cos(t) + i\sin(t)$
- (Wessel 1745-1818, also Gauss 1777-1855) Introduces 2D complex plane
- (Hamilton 1805-1865) Representation of complex numbers as 2-tuples from $\mathbb{R} \times \mathbb{R}$.

"The shortest path between two truths in the real domain passes through the complex domain." – Hadamard

Why complex numbers?

- Many, many applications, ranging from control theory to geometry to quantum mechanics.

Why complex numbers?

- Many, many applications, ranging from control theory to geometry to quantum mechanics.
- Fundamental theorem of algebra (Argand 1806):

Why complex numbers?

- Many, many applications, ranging from control theory to geometry to quantum mechanics.
- Fundamental theorem of algebra (Argand 1806):

Every non-constant, univariate polynomial with complex coefficients has at least one root in $\mathbb{C}$.

or, equivalently,

Every non-zero, degree n univariate polynomial with complex coefficients has precisely n complex roots.

e.g. $x^2 + 1$ is degree 2, and has two complex roots: i, and $-i$.

Why complex numbers?

- Many, many applications, ranging from control theory to geometry to quantum mechanics.
- Fundamental theorem of algebra (Argand 1806):

  Every non-constant, univariate polynomial with complex coefficients has at least one root in $\mathbb{C}$.

  or, equivalently,

  Every non-zero, degree n univariate polynomial with complex coefficients has precisely n complex roots.

  e.g. $x^2 + 1$ is degree 2, and has two complex roots: i, and $-i$.
- Moral: You should care about complex numbers!

Two views

Any complex number $z \in \mathbb{C}$ can be viewed in two equivalent ways:

- $z = x + yi$, for $x, y \in \mathbb{R}$, $i = \sqrt{-1}$.
  - Q: Why does this mean $\mathbb{C}$ can be viewed equivalently as $\mathbb{R} \times \mathbb{R}$?

Two views

Any complex number $z \in \mathbb{C}$ can be viewed in two equivalent ways:

- $z = x + yi$, for $x, y \in \mathbb{R}$, $i = \sqrt{-1}$.
  - Q: Why does this mean $\mathbb{C}$ can be viewed equivalently as $\mathbb{R} \times \mathbb{R}$?
  - $\overline{z} = x - iy$ is *complex conjugate* of z. (Sometimes denoted z^*.)

Two views

Any complex number $z \in \mathbb{C}$ can be viewed in two equivalent ways:

- $z = x + yi$, for $x, y \in \mathbb{R}$, $i = \sqrt{-1}$.
  - Q: Why does this mean $\mathbb{C}$ can be viewed equivalently as $\mathbb{R} \times \mathbb{R}$?
  - $\overline{z} = x - iy$ is *complex conjugate* of z. (Sometimes denoted z^*.)
- (Polar form) $z = re^{i\phi}$ for $r, \phi \in \mathbb{R}$. Here,
  - r is the "magnitude" of z, i.e. $r = |z| = \sqrt{x^2 + y^2}$.

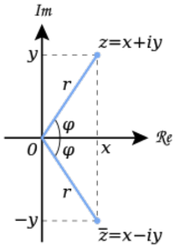
    Q: What norm does the formula for magnitude remind you of?

Two views

Any complex number $z \in \mathbb{C}$ can be viewed in two equivalent ways:

- $z = x + yi$, for $x, y \in \mathbb{R}$, $i = \sqrt{-1}$.
  - Q: Why does this mean $\mathbb{C}$ can be viewed equivalently as $\mathbb{R} \times \mathbb{R}$?
  - $\overline{z} = x - iy$ is *complex conjugate* of z. (Sometimes denoted z^*.)
- (Polar form) $z = re^{i\phi}$ for $r, \phi \in \mathbb{R}$. Here,
  - r is the "magnitude" of z, i.e. $r = |z| = \sqrt{x^2 + y^2}$.

    Q: What norm does the formula for magnitude remind you of?
  - $\phi \in [\pi, -\pi)$ is the angle of z (in radians):



Exercises with complex numbers

1. Is $\mathbb{R} \subseteq \mathbb{C}$?
2. Compute sum $(a + bi) + (c + di)$.
3. Compute product $(a + bi)(c + di)$.
4. Recall for $z = x + iy$ that $|z| = \sqrt{x^2 + y^2}$. Observe that this reduces to the usual absolute value when $z \in \mathbb{R}$.
5. Show that for any $z \in \mathbb{C}$, $z + z^* \in \mathbb{R}$.
6. Rewrite the formula $|z| = \sqrt{x^2 + y^2}$ in terms of the product of zz^*.
7. What are $\pm 1, \pm i$ in polar form?
8. Using the 2D complex plane, derive the formula $|z| = \sqrt{x^2 + y^2}$.
9. If we allow angles $\phi \in \mathbb{R}$, is the representation of a given $z \in \mathbb{C}$ unique?
10. Use the 2D complex plane to derive the two square roots of 1. (Q: Why are we guaranteed that 1 has precisely 2 square roots?)

With $\mathbb{C}$ in hand, can now define polynomials with coefficients from $\mathbb{C}$.

Later, we will use $\mathbb{C}$ for the Fourier transform as well.

Outline

Polynomials (brief review)

Univariate polynomial

A univariate polynomial is a function $f : \mathbb{C} \mapsto \mathbb{C}$ of form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{j=0}^{n} a_j x^j,$$

for all $a_j \in \mathbb{C}$. *Degree* of f is $\deg(f) = n$ (i.e. index of largest non-zero coefficient a_n). The set of univariate polynomials over $\mathbb{C}$ is denoted $\mathbb{C}[x]$.

Polynomials (brief review)

Univariate polynomial

A univariate polynomial is a function $f : \mathbb{C} \mapsto \mathbb{C}$ of form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{j=0}^{n} a_j x^j,$$

for all $a_j \in \mathbb{C}$. *Degree* of f is $\deg(f) = n$ (i.e. index of largest non-zero coefficient a_n). The set of univariate polynomials over $\mathbb{C}$ is denoted $\mathbb{C}[x]$.

Sum and product of polynomials

For $f, g \in \mathbb{C}[x]$ with $f(x) = \sum_{j=0}^{n} a_j x^j$ and $g(x) = \sum_{j=0}^{n} b_j x^j$,

$$f(x) + g(x) = \sum_{j=0}^{n} (a_j + b_j) x^j, \quad \text{and} \quad f(x)g(x) = \sum_{j=0}^{2n} \left(\sum_{k=0}^{j} a_k b_{j-k} \right) x^j.$$

Ex. Prove the multiplication formula for $f(x)g(x)$ holds.

Exercises with polynomials

1. What is the degree of $f(x) = -7x^3 + 4x + \sqrt{2}$? $f(x) = 4$?
2. Are non-positive-integer exponents on x allowed in our definition of polynomials?
3. Compute the sum of $f(x) = 3x^2 - 4x - 9$ and $g(x) = x^3 + 4$.
4. For $f, g \in \mathbb{C}[x]$ of degree n_f and n_g, resp., what is $\deg(f(x) + g(x))$?
5. Compute the product of $f(x) = 3x^2 - 4x - 9$ and $g(x) = x^3 + 4$.
6. For $f, g \in \mathbb{C}[x]$ of degree n_f and n_g, resp., what is $\deg(f(x)g(x))$?
7. Recall the Fundamental Theorem of Algebra says that any $f \in \mathbb{C}[x]$ with $\deg(f) = n$ has precisely n roots over $\mathbb{C}$. What are the roots of $3x^2 - 1$? $x^3 - 1$? $x^4 - 1$? More generally, $x^n - 1$?
8. Is there a real-numbered analogue of the Fundamental Theorem of Algebra? i.e. true that any $f \in \mathbb{R}[x]$ with $\deg(f) = n$ has n roots over $\mathbb{R}$?

Cost of polynomial multiplication

How many field operations over $\mathbb{C}$ does the naive algorithm take to multiply two degree-n polynomials $f, g \in \mathbb{C}[x]$?

Cost of polynomial multiplication

How many field operations over $\mathbb{C}$ does the naive algorithm take to multiply two degree-n polynomials $f, g \in \mathbb{C}[x]$?

A: $\Theta(n^2)$ time.

Cost of polynomial multiplication

How many field operations over $\mathbb{C}$ does the naive algorithm take to multiply two degree-n polynomials $f, g \in \mathbb{C}[x]$?

A: $\Theta(n^2)$ time.

Q: Can we do it in subquadratic time?

Cost of polynomial multiplication

How many field operations over $\mathbb{C}$ does the naive algorithm take to multiply two degree-n polynomials $f, g \in \mathbb{C}[x]$?

A: $\Theta(n^2)$ time.

Q: Can we do it in subquadratic time?

A: Surprisingly, yes! Can do it in $\Theta(n \log n)$ time.

Cost of polynomial multiplication

How many field operations over $\mathbb{C}$ does the naive algorithm take to multiply two degree-n polynomials $f, g \in \mathbb{C}[x]$?

A: $\Theta(n^2)$ time.

Q: Can we do it in subquadratic time?

A: Surprisingly, yes! Can do it in $\Theta(n \log n)$ time.

Battle plan:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.

Cost of polynomial multiplication

How many field operations over $\mathbb{C}$ does the naive algorithm take to multiply two degree-n polynomials $f, g \in \mathbb{C}[x]$?

A: $\Theta(n^2)$ time.

Q: Can we do it in subquadratic time?

A: Surprisingly, yes! Can do it in $\Theta(n \log n)$ time.

Battle plan:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.

Cost of polynomial multiplication

How many field operations over $\mathbb{C}$ does the naive algorithm take to multiply two degree-n polynomials $f, g \in \mathbb{C}[x]$?

A: $\Theta(n^2)$ time.

Q: Can we do it in subquadratic time?

A: Surprisingly, yes! Can do it in $\Theta(n \log n)$ time.

Battle plan:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.

Two representations of polynomials

Coefficient representation

Polynomial $f \in \mathbb{C}[x]$ of degree n written as $f(x) = \sum_{j=0}^{n} a_j x^n$, or in vector form:

$$\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \end{pmatrix}^T \in \mathbb{C}^n.$$

Two representations of polynomials

Coefficient representation

Polynomial $f \in \mathbb{C}[x]$ of degree n written as $f(x) = \sum_{j=0}^{n} a_j x^n$, or in vector form:

$$\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \end{pmatrix}^T \in \mathbb{C}^n.$$

Observation: Given $f \in \mathbb{C}[x]$ in coefficient form, can evaluate f at any point $x \in \mathbb{C}$ in $\Theta(n)$ time using *Horner's rule*:

$$f(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x(a_n))\cdots)).$$

Ex. Use Horner's rule to evaluate $f(x) = 5x^3 - 2x^2 - x + 1$ at $x = e^{i\pi}$.

Two representations of polynomials

Coefficient representation

Polynomial $f \in \mathbb{C}[x]$ of degree n written as $f(x) = \sum_{j=0}^{n} a_j x^n$, or in vector form:

$$\mathbf{a} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \end{pmatrix}^T \in \mathbb{C}^n.$$

Observation: Given $f \in \mathbb{C}[x]$ in coefficient form, can evaluate f at any point $x \in \mathbb{C}$ in $\Theta(n)$ time using *Horner's rule*:

$$f(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x(a_n)) \cdots)).$$

Ex. Use Horner's rule to evaluate $f(x) = 5x^3 - 2x^2 - x + 1$ at $x = e^{i\pi}$.

(Aside: What is $e^{i\pi}$?)

Two representations of polynomials

Point-value representation

Point-value rep. of $f \in \mathbb{C}[x]$ of degree n is a set of $n+1$ *point-value pairs* $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, such that:

- $f(x_i) = y_i$ for all x_i, and
- all x_i are distinct.

Two representations of polynomials

Point-value representation

Point-value rep. of $f \in \mathbb{C}[x]$ of degree n is a set of $n+1$ *point-value pairs* $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, such that:

- $f(x_i) = y_i$ for all x_i, and
- all x_i are distinct.

Ex. Give two distinct point-value representations for $f(x) = 3x^2 + 1$.

Two representations of polynomials

Point-value representation

Point-value rep. of $f \in \mathbb{C}[x]$ of degree n is a set of $n+1$ *point-value pairs* $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, such that:

- $f(x_i) = y_i$ for all x_i, and
- all x_i are distinct.

Ex. Give two distinct point-value representations for $f(x) = 3x^2 + 1$.

Obs: Not clear *a priori* that point-value representation captures f...

Two representations of polynomials

Point-value representation

Point-value rep. of $f \in \mathbb{C}[x]$ of degree n is a set of $n+1$ *point-value pairs* $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, such that:

- $f(x_i) = y_i$ for all x_i, and
- all x_i are distinct.

Ex. Give two distinct point-value representations for $f(x) = 3x^2 + 1$.

Obs: Not clear *a priori* that point-value representation captures f...

Interpolation Theorem

Any set of $n+1$ point-value pairs $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ with distinct x_i defines a unique polynomial f such that:

- $\deg(f) \leq n$,
- $f(x_j) = y_j$ for $j \in \{0, \ldots, n\}$.

Interpolation Theorem

Any set of $n+1$ point-value pairs $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ with distinct x_i defines a unique polynomial f such that:

- $\deg(f) \leq n$,
- $f(x_j) = y_j$ for $j \in \{0, \ldots, n\}$.

Interpolation Theorem

Any set of $n+1$ point-value pairs $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ with distinct x_i defines a unique polynomial f such that:

- $\deg(f) \leq n$,
- $f(x_j) = y_j$ for $j \in \{0, \ldots, n\}$.

Proof. Looking for $f(x) = \sum_{j=0}^{n} a_j x^n$ s.t. $f(x_i) = y_i$. Encode as matrix mult.:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Interpolation Theorem

Any set of $n+1$ point-value pairs $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ with distinct x_i defines a unique polynomial f such that:

- $\deg(f) \leq n$,
- $f(x_j) = y_j$ for $j \in \{0, \ldots, n\}$.

Proof. Looking for $f(x) = \sum_{j=0}^{n} a_j x^n$ s.t. $f(x_i) = y_i$. Encode as matrix mult.:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Write as $V(x_0, \ldots, x_n)\mathbf{a} = \mathbf{y}$, for $V \in \mathbb{C}^{n+1 \times n+1}$ a *Vandermonde matrix*.

Interpolation Theorem

Any set of $n+1$ point-value pairs $\{(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)\}$ with distinct x_i defines a unique polynomial f such that:

- $\deg(f) \leq n$,
- $f(x_j) = y_j$ for $j \in \{0,\ldots,n\}$.

Proof. Looking for $f(x) = \sum_{j=0}^{n} a_j x^n$ s.t. $f(x_i) = y_i$. Encode as matrix mult.:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Write as $V(x_0,\ldots,x_n)\mathbf{a} = \mathbf{y}$, for $V \in \mathbb{C}^{n+1\times n+1}$ a *Vandermonde matrix*.

Fact: Any Vandermonde matrix has an inverse if all x_j are distinct.

Interpolation Theorem

Any set of $n+1$ point-value pairs $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ with distinct x_i defines a unique polynomial f such that:

- $\deg(f) \leq n$,
- $f(x_j) = y_j$ for $j \in \{0, \ldots, n\}$.

Proof. Looking for $f(x) = \sum_{j=0}^{n} a_j x^n$ s.t. $f(x_i) = y_i$. Encode as matrix mult.:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Write as $V(x_0, \ldots, x_n)\mathbf{a} = \mathbf{y}$, for $V \in \mathbb{C}^{n+1 \times n+1}$ a *Vandermonde matrix*.

Fact: Any Vandermonde matrix has an inverse if all x_j are distinct.

Conclusion: Unique solution for $\mathbf{a}$ given by $\mathbf{a} = V(x_0, \ldots, x_n)^{-1}\mathbf{y}$.

The big fuss about the point-value representation

The big fuss about the point-value representation

Q: Given degree-n polynomials in point-value form,

$$f(x) \quad \text{``}=\text{''} \quad \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\},$$
$$g(x) \quad \text{``}=\text{''} \quad \{(x_0, y_0'), (x_1, y_1'), \ldots, (x_n, y_n')\},$$

what is cost of multiplying $f(x)$ and $g(x)$? (Note: Shared x_j values above!)

The big fuss about the point-value representation

Q: Given degree-n polynomials in point-value form,

$$\begin{aligned} f(x) \quad &\text{“}=\text{”} \quad \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}, \\ g(x) \quad &\text{“}=\text{”} \quad \{(x_0, y_0'), (x_1, y_1'), \ldots, (x_n, y_n')\}, \end{aligned}$$

what is cost of multiplying $f(x)$ and $g(x)$? (Note: Shared x_j values above!)

A: $\Theta(n)$ time! The point-value representation for $f(x)g(x)$ is

$$\{(x_0, y_0y_0'), (x_1, y_1y_1'), \ldots, (x_n, y_ny_n')\}, \tag{2}$$

i.e. suffices to point-wise multiply. (Ex. Convince yourself of this claim.)

The big fuss about the point-value representation

Q: Given degree-n polynomials in point-value form,

$$\begin{aligned} f(x) \quad &\text{``} = \text{''} \quad \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}, \\ g(x) \quad &\text{``} = \text{''} \quad \{(x_0, y'_0), (x_1, y'_1), \ldots, (x_n, y'_n)\}, \end{aligned}$$

what is cost of multiplying $f(x)$ and $g(x)$? (Note: Shared x_j values above!)

A: $\Theta(n)$ time! The point-value representation for $f(x)g(x)$ is

$$\{(x_0, y_0y'_0), (x_1, y_1y'_1), \ldots, (x_n, y_ny'_n)\}, \qquad (2)$$

i.e. suffices to point-wise multiply. (Ex. Convince yourself of this claim.)

Q: Do you see a problem with the statement above?

The big fuss about the point-value representation

Q: Given degree-n polynomials in point-value form,

$$f(x) \quad \text{“} = \text{”} \quad \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\},$$
$$g(x) \quad \text{“} = \text{”} \quad \{(x_0, y_0'), (x_1, y_1'), \ldots, (x_n, y_n')\},$$

what is cost of multiplying $f(x)$ and $g(x)$? (Note: Shared x_j values above!)

A: $\Theta(n)$ time! The point-value representation for $f(x)g(x)$ is

$$\{(x_0, y_0y_0'), (x_1, y_1y_1'), \ldots, (x_n, y_ny_n')\}, \tag{2}$$

i.e. suffices to point-wise multiply. (Ex. Convince yourself of this claim.)

Q: Do you see a problem with the statement above?

A: Eq. (2) has $n+1$ datapoints, but $f(x)g(x)$ is degree $\leq 2n$ (i.e. not enough data to uniquely identify $f(x)g(x)$ via Interpolation Theorem).

The big fuss about the point-value representation

Q: Given degree-n polynomials in point-value form,

$$\begin{aligned} f(x) \quad &\text{“}=\text{”} \quad \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}, \\ g(x) \quad &\text{“}=\text{”} \quad \{(x_0, y_0'), (x_1, y_1'), \ldots, (x_n, y_n')\}, \end{aligned}$$

what is cost of multiplying $f(x)$ and $g(x)$? (Note: Shared x_j values above!)

A: $\Theta(n)$ time! The point-value representation for $f(x)g(x)$ is

$$\{(x_0, y_0y_0'), (x_1, y_1y_1'), \ldots, (x_n, y_ny_n')\}, \tag{2}$$

i.e. suffices to point-wise multiply. (Ex. Convince yourself of this claim.)

Q: Do you see a problem with the statement above?

A: Eq. (2) has $n+1$ datapoints, but $f(x)g(x)$ is degree $\leq 2n$ (i.e. not enough data to uniquely identify $f(x)g(x)$ via Interpolation Theorem).

Solution: Start with point-value representations for f and g which have $2n+1$ points (i.e. before multiplying).

Recap so far

- Multiplying $f, g \in \mathbb{C}[x]$ of degree n naively takes $O(n^2)$ time in coefficient form.

Recap so far

- Multiplying $f, g \in \mathbb{C}[x]$ of degree n naively takes $O(n^2)$ time in coefficient form.
- Hope: If could convert to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.

Recap so far

- Multiplying $f, g \in \mathbb{C}[x]$ of degree n naively takes $O(n^2)$ time in coefficient form.
- Hope: If could convert to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Cost of going from coefficient to point-value form with $2n + 1$ point-value pairs?

Recap so far

- Multiplying $f, g \in \mathbb{C}[x]$ of degree n naively takes $O(n^2)$ time in coefficient form.
- Hope: If could convert to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Cost of going from coefficient to point-value form with $2n+1$ point-value pairs?
  - Via Horner's rule, $O(n)$-time per evaluation of f at a point (x, y).
  - Repeating for $2n+1$ points x_j yields $O(n^2)$ time total.

Recap so far

- Multiplying $f, g \in \mathbb{C}[x]$ of degree n naively takes $O(n^2)$ time in coefficient form.
- Hope: If could convert to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Cost of going from coefficient to point-value form with $2n + 1$ point-value pairs?
  - Via Horner's rule, $O(n)$-time per evaluation of f at a point (x, y).
  - Repeating for $2n + 1$ points x_j yields $O(n^2)$ time total.
- Fact: Can also convert back from point-value to coefficient form in $O(n^2)$ time using *Lagrange's formula*.

Recap so far

- Multiplying $f, g \in \mathbb{C}[x]$ of degree n naively takes $O(n^2)$ time in coefficient form.
- Hope: If could convert to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Cost of going from coefficient to point-value form with $2n+1$ point-value pairs?
  - Via Horner's rule, $O(n)$-time per evaluation of f at a point (x, y).
  - Repeating for $2n+1$ points x_j yields $O(n^2)$ time total.
- Fact: Can also convert back from point-value to coefficient form in $O(n^2)$ time using *Lagrange's formula*.
- But this is stupid...Just converting between representations takes as much time as naive multiplication algorithm...

Recap so far

- Multiplying $f, g \in \mathbb{C}[x]$ of degree n naively takes $O(n^2)$ time in coefficient form.
- Hope: If could convert to and from point-value form “quickly”, could instead do point-wise multiplication, which takes $O(n)$ time.
- Cost of going from coefficient to point-value form with $2n+1$ point-value pairs?
  - Via Horner’s rule, $O(n)$-time per evaluation of f at a point (x, y).
  - Repeating for $2n+1$ points x_j yields $O(n^2)$ time total.
- Fact: Can also convert back from point-value to coefficient form in $O(n^2)$ time using *Lagrange’s formula*.
- But this is stupid. . . Just converting between representations takes as much time as naive multiplication algorithm. . .

(Facebook worried emoji)

Recap so far

- Hope: If could convert to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.

Recap so far

- Hope: If could convert to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!

Recap so far

- Hope: If could convert to and from point-value form “quickly”, could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!
- Then, our total cost for multiplying polynomials would be (why?)

$$O(n \log n) + O(n) + O(n \log n) \in O(n \log n),$$

improving on $O(n^2)$ of naive algorithm.

Recap so far

- Hope: If could convert to and from point-value form “quickly”, could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!
- Then, our total cost for multiplying polynomials would be (why?)

$$O(n \log n) + O(n) + O(n \log n) \in O(n \log n),$$

  improving on $O(n^2)$ of naive algorithm.
- Trick: Use Fourier transform.

Recap so far

- Hope: If could convert to and from point-value form “quickly”, could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!
- Then, our total cost for multiplying polynomials would be (why?)

$$O(n \log n) + O(n) + O(n \log n) \in O(n \log n),$$

  improving on $O(n^2)$ of naive algorithm.
- Trick: Use Fourier transform.

(Facebook relieved emoji)

Outline

1. Introduction to matrices (review)
2. Matrix multiplication algorithms
   - Strassen's algorithm (1967)
   - Drineas-Kannan-Mahoney randomized algorithm (2006)
3. Random walks
   - Gambler's ruin
   - Google's PageRank algorithm (1999)
4. Polynomial multiplication
   - Complex numbers
   - Polynomials
   - $O(N \log N)$-time polynomial multiplication via Fourier Transform

Roots of unity

The Discrete Fourier Transform (DFT) is a matrix, whose definition requires special complex numbers known as *roots of unity*.

Roots of unity

The Discrete Fourier Transform (DFT) is a matrix, whose definition requires special complex numbers known as *roots of unity*.

Recall: What are the roots of $f(x) = x^n - 1$?

Roots of unity

The Discrete Fourier Transform (DFT) is a matrix, whose definition requires special complex numbers known as *roots of unity*.

Recall: What are the roots of $f(x) = x^n - 1$?

*N*th roots of unity

The *n*th roots of unity are the roots of $f(x) = x^n - 1$, namely

$$1, e^{2\pi i/n}, e^{2\cdot 2\pi i/n}, e^{3\cdot 2\pi i/n}, \ldots, e^{(n-1)\cdot 2\pi i/n}.$$ (Why?)

Roots of unity

The Discrete Fourier Transform (DFT) is a matrix, whose definition requires special complex numbers known as *roots of unity*.

Recall: What are the roots of $f(x) = x^n - 1$?

*N*th roots of unity

The *n*th roots of unity are the roots of $f(x) = x^n - 1$, namely

$$1, e^{2\pi i/n}, e^{2\cdot 2\pi i/n}, e^{3\cdot 2\pi i/n}, \ldots, e^{(n-1)\cdot 2\pi i/n}.$$ (Why?)

More concisely, define principal *n*th root of unity as $\omega_n = e^{2\pi i/n}$.

Then, *n*th roots of unity are: $\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}$.

Roots of unity

The Discrete Fourier Transform (DFT) is a matrix, whose definition requires special complex numbers known as *roots of unity*.

Recall: What are the roots of $f(x) = x^n - 1$?

Nth roots of unity

The nth roots of unity are the roots of $f(x) = x^n - 1$, namely

$$1, e^{2\pi i/n}, e^{2\cdot 2\pi i/n}, e^{3\cdot 2\pi i/n}, \ldots, e^{(n-1)\cdot 2\pi i/n}.$$ (Why?)

More concisely, define principal nth root of unity as $\omega_n = e^{2\pi i/n}$.

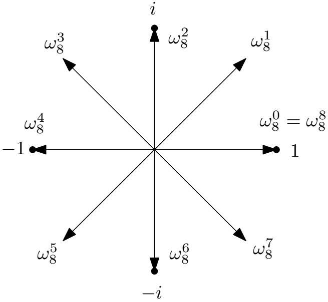
Then, nth roots of unity are: $\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}$.

Ex. What is the magnitude of any root of unity, i.e. $\left|e^{2j\pi i/n}\right|$ for $j \in \mathbb{Z}$?

Ex. What are the 4th roots of unity?

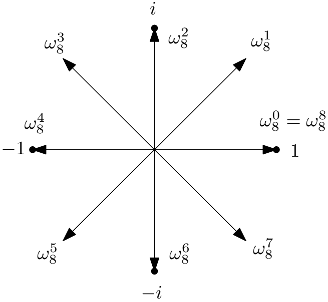
Properties

Recall: $\omega_n = e^{2\pi i/n}$. Below are the 8th roots of unity:



Properties

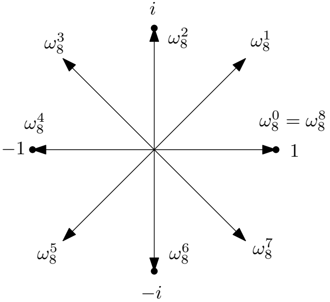
Recall: $\omega_n = e^{2\pi i/n}$. Below are the 8th roots of unity:



- **Cancellation Lemma:** For any integers $n, k \geq 0$, and $d > 0$, $\omega_{dn}^{dk} = \omega_n^k$.

Properties

Recall: $\omega_n = e^{2\pi i/n}$. Below are the 8th roots of unity:



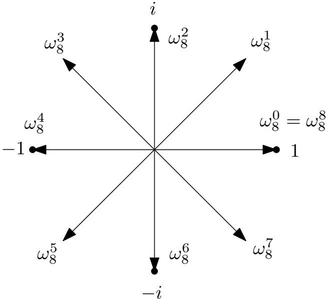
- Cancellation Lemma: For any integers $n, k \geq 0$, and $d > 0$, $\omega_{dn}^{dk} = \omega_n^k$.
- Summation Lemma: For any integer $n \geq 1$ and integer $k \neq 0$ not divisible by n,

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

Q: What if $k = 0$?

Properties

Recall: $\omega_n = e^{2\pi i/n}$. Below are the 8th roots of unity:



- Cancellation Lemma: For any integers $n, k \geq 0$, and $d > 0$, $\omega_{dn}^{dk} = \omega_n^k$.
- Summation Lemma: For any integer $n \geq 1$ and integer $k \neq 0$ not divisible by n,

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

Q: What if $k = 0$?

Ex. Prove both lemmas above (Hint: Use closed form for geometric series). Can you visualize proofs on 2D plane?

Let $f = \sum_{j=0}^{n} a_j x^j \in \mathbb{C}[x]$, and recall $\omega_n = e^{2\pi i/n}$.

Let $f = \sum_{j=0}^{n} a_j x^j \in \mathbb{C}[x]$, and recall $\omega_n = e^{2\pi i/n}$.

Recall:

- Hope: If could convert f to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!

Let $f = \sum_{j=0}^{n} a_j x^j \in \mathbb{C}[x]$, and recall $\omega_n = e^{2\pi i/n}$.

Recall:

- Hope: If could convert f to and from point-value form “quickly”, could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!

Moving forward:

- The *Discrete Fourier Transform (DFT)* evaluates f at n "carefully" chosen points x_j. Can you guess which ones?

Let $f = \sum_{j=0}^{n} a_j x^j \in \mathbb{C}[x]$, and recall $\omega_n = e^{2\pi i/n}$.

Recall:

- Hope: If could convert f to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!

Moving forward:

- The *Discrete Fourier Transform (DFT)* evaluates f at n "carefully" chosen points x_j. Can you guess which ones?
- A: The nth roots of unity, ω_n^k!

Let $f = \sum_{j=0}^{n} a_j x^j \in \mathbb{C}[x]$, and recall $\omega_n = e^{2\pi i/n}$.

Recall:

- Hope: If could convert f to and from point-value form "quickly", could instead do point-wise multiplication, which takes $O(n)$ time.
- Key observation: Evaluating f at *arbitrary* points $x_0, \ldots, x_{2n}$ takes $O(n^2)$, but if we choose the x_j "carefully", can do it in $O(n \log n)$ time!

Moving forward:

- The *Discrete Fourier Transform (DFT)* evaluates f at n "carefully" chosen points x_j. Can you guess which ones?
- A: The nth roots of unity, ω_n^k!
- For succinctness, let $N := n + 1$. Then, DFT maps coefficient vector $\mathbf{a}$ to:

$$\mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} \quad \mapsto \quad \mathbf{y} = \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix},$$

i.e. $y_k = f(\omega_N^k) = \sum_{j=0} a_j \left(\omega_N^k\right)^j$.

Discrete Fourier Transform (DFT)

Recall: The Discrete Fourier Transform (DFT) is a matrix, whose definition requires special complex numbers known as *roots of unity*.

Discrete Fourier Transform (DFT)

Recall: The Discrete Fourier Transform (DFT) is a matrix, whose definition requires special complex numbers known as *roots of unity*.

Q: Can you guess the matrix now? (Hint: Vandermonde matrix.)

Interpolation Theorem

Any set of $n+1$ point-value pairs $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ with distinct x_i defines a unique polynomial f such that:

- $\deg(f) \leq n$,
- $f(x_j) = y_j$ for $j \in \{0, \ldots, n\}$.

Proof. Looking for $f(x) = \sum_{j=0}^{n} a_j x^n$ s.t. $f(x_i) = y_i$. Encode as matrix mult.:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Write as $V(x_0, \ldots, x_n)\mathbf{a} = \mathbf{y}$, for $V \in \mathbb{C}^{n+1 \times n+1}$ a *Vandermonde matrix*.

Fact: Any Vandermonde matrix has an inverse if all x_j are distinct.
Conclusion: Unique solution for $\mathbf{a}$ given by $\mathbf{a} = V(x_0, \ldots, x_n)^{-1}\mathbf{y}$.

The DFT matrix

Moral: Evaluating a polynomial at a set of points is matrix multiplication.

Want to evaluate $f(x) = \sum_{j=0}^{n} a_j x^n$ at inputs $\omega_N^0, \omega_N^1, \ldots, \omega_N^{N-1}$:

$$\begin{pmatrix} 1 & ? & ? & \cdots & ? \\ 1 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & ? & ? & \cdots & ? \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix}.$$

The DFT matrix

Moral: Evaluating a polynomial at a set of points is matrix multiplication.

Want to evaluate $f(x) = \sum_{j=0}^{n} a_j x^n$ at inputs $\omega_N^0, \omega_N^1, \ldots, \omega_N^{N-1}$:

$$\begin{pmatrix} 1 & ? & ? & \cdots & ? \\ 1 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & ? & ? & \cdots & ? \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix}.$$

$$\begin{pmatrix} 1 & \left(\omega_N^0\right)^1 & \left(\omega_N^0\right)^2 & \cdots & \left(\omega_N^0\right)^{(N-1)} \\ 1 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & ? & ? & \cdots & ? \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix}.$$

The DFT matrix

Moral: Evaluating a polynomial at a set of points is matrix multiplication.

Want to evaluate $f(x) = \sum_{j=0}^{n} a_j x^n$ at inputs $\omega_N^0, \omega_N^1, \ldots, \omega_N^{N-1}$:

$$\begin{pmatrix} 1 & ? & ? & \cdots & ? \\ 1 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & ? & ? & \cdots & ? \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix}.$$

$$\begin{pmatrix} 1 & (\omega_N^0)^1 & (\omega_N^0)^2 & \cdots & (\omega_N^0)^{(N-1)} \\ 1 & ? & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & ? & ? & \cdots & ? \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix}.$$

$$\begin{pmatrix} 1 & (\omega_N^0)^1 & (\omega_N^0)^2 & \cdots & (\omega_N^0)^{(N-1)} \\ 1 & (\omega_N^1)^1 & (\omega_N^1)^2 & \cdots & (\omega_N^1)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & ? & ? & \cdots & ? \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(\omega_N^0) \\ f(\omega_N^1) \\ \vdots \\ f(\omega_N^{N-1}) \end{pmatrix}.$$

The DFT matrix

The $N \times N$ complex matrix encoding the DFT of order N is thus:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & \left(\omega_N^0\right)^1 & \left(\omega_N^0\right)^2 & \cdots & \left(\omega_N^0\right)^{(N-1)} \\ 1 & \left(\omega_N^1\right)^1 & \left(\omega_N^1\right)^2 & \cdots & \left(\omega_N^1\right)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\omega_N^{N-1}\right)^1 & \left(\omega_N^{N-1}\right)^2 & \cdots & \left(\omega_N^{N-1}\right)^{(N-1)} \end{pmatrix}.$$

The DFT matrix

The $N \times N$ complex matrix encoding the DFT of order N is thus:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & \left(\omega_N^0\right)^1 & \left(\omega_N^0\right)^2 & \cdots & \left(\omega_N^0\right)^{(N-1)} \\ 1 & \left(\omega_N^1\right)^1 & \left(\omega_N^1\right)^2 & \cdots & \left(\omega_N^1\right)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\omega_N^{N-1}\right)^1 & \left(\omega_N^{N-1}\right)^2 & \cdots & \left(\omega_N^{N-1}\right)^{(N-1)} \end{pmatrix}.$$

Recap:

- Wanted to beat $O(n^2)$ time polynomial multiplication.

The DFT matrix

The $N \times N$ complex matrix encoding the DFT of order N is thus:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & (\omega_N^0)^1 & (\omega_N^0)^2 & \cdots & (\omega_N^0)^{(N-1)} \\ 1 & (\omega_N^1)^1 & (\omega_N^1)^2 & \cdots & (\omega_N^1)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\omega_N^{N-1}\right)^1 & \left(\omega_N^{N-1}\right)^2 & \cdots & \left(\omega_N^{N-1}\right)^{(N-1)} \end{pmatrix}.$$

Recap:

- Wanted to beat $O(n^2)$ time polynomial multiplication.
- To do so, wanted to map f from coefficient to point-value representation, and then do linear-time point-wise multiplication.

The DFT matrix

The $N \times N$ complex matrix encoding the DFT of order N is thus:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & \left(\omega_N^0\right)^1 & \left(\omega_N^0\right)^2 & \cdots & \left(\omega_N^0\right)^{(N-1)} \\ 1 & \left(\omega_N^1\right)^1 & \left(\omega_N^1\right)^2 & \cdots & \left(\omega_N^1\right)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\omega_N^{N-1}\right)^1 & \left(\omega_N^{N-1}\right)^2 & \cdots & \left(\omega_N^{N-1}\right)^{(N-1)} \end{pmatrix}.$$

Recap:

- Wanted to beat $O(n^2)$ time polynomial multiplication.
- To do so, wanted to map f from coefficient to point-value representation, and then do linear-time point-wise multiplication.
- To do this conversion, decided to evaluate f at Nth roots of unity, ω_N^k.

The DFT matrix

The $N \times N$ complex matrix encoding the DFT of order N is thus:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & \left(\omega_N^0\right)^1 & \left(\omega_N^0\right)^2 & \cdots & \left(\omega_N^0\right)^{(N-1)} \\ 1 & \left(\omega_N^1\right)^1 & \left(\omega_N^1\right)^2 & \cdots & \left(\omega_N^1\right)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\omega_N^{N-1}\right)^1 & \left(\omega_N^{N-1}\right)^2 & \cdots & \left(\omega_N^{N-1}\right)^{(N-1)} \end{pmatrix}.$$

Recap:

- Wanted to beat $O(n^2)$ time polynomial multiplication.
- To do so, wanted to map f from coefficient to point-value representation, and then do linear-time point-wise multiplication.
- To do this conversion, decided to evaluate f at Nth roots of unity, ω_N^k.
- This evaluation can be encoded as multiplication by the matrix DFT_N.

The DFT matrix

The $N \times N$ complex matrix encoding the DFT of order N is thus:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & \left(\omega_N^0\right)^1 & \left(\omega_N^0\right)^2 & \cdots & \left(\omega_N^0\right)^{(N-1)} \\ 1 & \left(\omega_N^1\right)^1 & \left(\omega_N^1\right)^2 & \cdots & \left(\omega_N^1\right)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\omega_N^{N-1}\right)^1 & \left(\omega_N^{N-1}\right)^2 & \cdots & \left(\omega_N^{N-1}\right)^{(N-1)} \end{pmatrix}.$$

Recap:

- Wanted to beat $O(n^2)$ time polynomial multiplication.
- To do so, wanted to map f from coefficient to point-value representation, and then do linear-time point-wise multiplication.
- To do this conversion, decided to evaluate f at Nth roots of unity, ω_N^k.
- This evaluation can be encoded as multiplication by the matrix DFT_N.
- Q: How long to naively compute $\mathrm{DFT}_N \mathbf{v}$ for arbitrary $\mathbf{v} \in \mathbb{C}^N$?

The DFT matrix

The $N \times N$ complex matrix encoding the DFT of order N is thus:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & \left(\omega_N^0\right)^1 & \left(\omega_N^0\right)^2 & \cdots & \left(\omega_N^0\right)^{(N-1)} \\ 1 & \left(\omega_N^1\right)^1 & \left(\omega_N^1\right)^2 & \cdots & \left(\omega_N^1\right)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \left(\omega_N^{N-1}\right)^1 & \left(\omega_N^{N-1}\right)^2 & \cdots & \left(\omega_N^{N-1}\right)^{(N-1)} \end{pmatrix}.$$

Recap:

- Wanted to beat $O(n^2)$ time polynomial multiplication.
- To do so, wanted to map f from coefficient to point-value representation, and then do linear-time point-wise multiplication.
- To do this conversion, decided to evaluate f at Nth roots of unity, ω_N^k.
- This evaluation can be encoded as multiplication by the matrix DFT_N.
- Q: How long to naively compute $\mathrm{DFT}_N \mathbf{v}$ for arbitrary $\mathbf{v} \in \mathbb{C}^N$?
- A: $O(N^2) \in O(n^2)\ldots$ (Recall $N = n + 1$.) #$&%&$%!

This makes me feel. . .


?

This makes me feel. . .

?

?

This makes me feel...

?

?

?

Breathe in, breathe out

We can't stop now…Let's remind ourselves why it really is important to find a clever implementation of the DFT.

`http://nautil.us/blog/`
`the-math-trick-behind-mp3s-jpegs-and-homer-simpsons-f`
Math Trick Behind MP3s, JPEGs, and Homer Simpson's Face

(Click on link in pdf to follow link)

Warning: Must read. I *will* test you on this on exam.

Fast Fourier Transform (FFT)

Can implement DFT_N in time $O(N \log N)$:

Fast Fourier Transform (FFT)

Can implement DFT_N in time $O(N \log N)$:

- Via divide-and-conquer

Fast Fourier Transform (FFT)

Can implement DFT_N in time $O(N \log N)$:

- Via divide-and-conquer
- Note: Only allows us to evaluate f at Nth roots of unity
- Q: Why can we evaluate f at Nth roots of unity quickly, whereas evaluating f at N arbitrary points would take $O(N^2)$?

Fast Fourier Transform (FFT)

Can implement DFT_N in time $O(N \log N)$:

- Via divide-and-conquer
- Note: Only allows us to evaluate f at Nth roots of unity
- Q: Why can we evaluate f at Nth roots of unity quickly, whereas evaluating f at N arbitrary points would take $O(N^2)$?

Halving Lemma

Suppose N is even. Then, for any $k \in \mathbb{Z}^+$, $(\omega_N^k)^2 = \omega_{N/2}^k$.

Proof.

Fast Fourier Transform (FFT)

Can implement DFT_N in time $O(N \log N)$:

- Via divide-and-conquer
- Note: Only allows us to evaluate f at Nth roots of unity
- Q: Why can we evaluate f at Nth roots of unity quickly, whereas evaluating f at N arbitrary points would take $O(N^2)$?

Halving Lemma

Suppose N is even. Then, for any $k \in \mathbb{Z}^+$, $(\omega_N^k)^2 = \omega_{N/2}^k$.

Proof. Use Cancellation Lemma, which said:

$$\text{For any integers } n, k \geq 0, \text{ and } d > 0, \ \omega_{dn}^{dk} = \omega_n^k.$$

Q: What value of d to choose to prove Halving Lemma?

Fast Fourier Transform (FFT)

Can implement DFT_N in time $O(N \log N)$:

- Via divide-and-conquer
- Note: Only allows us to evaluate f at Nth roots of unity
- Q: Why can we evaluate f at Nth roots of unity quickly, whereas evaluating f at N arbitrary points would take $O(N^2)$?

Halving Lemma

Suppose N is even. Then, for any $k \in \mathbb{Z}^+$, $(\omega_N^k)^2 = \omega_{N/2}^k$.

Proof. Use Cancellation Lemma, which said:

$$\text{For any integers } n, k \geq 0, \text{ and } d > 0, \ \omega_{dn}^{dk} = \omega_n^k.$$

Q: What value of d to choose to prove Halving Lemma?

Idea: Halving Lemma allows us to recurse by simulating order-N DFT by a pair of order-$(N/2)$ DFTs.

Recursive breakdown of polynomials

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

Recursive breakdown of polynomials

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\ &= \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}\right) + \quad \text{(even)} \\ &\quad \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots + a_n x^n\right) \quad \text{(odd)} \end{aligned}$$

Recursive breakdown of polynomials

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$
\begin{aligned}
f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\
&= \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}\right) + \quad \text{(even)} \\
&\quad \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots + a_n x^n\right) \quad \text{(odd)} \\
&= \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}\right) + \quad \text{(even)} \\
&\quad x\left(a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_n x^{n-1}\right) \quad \text{(odd)}
\end{aligned}
$$

Recursive breakdown of polynomials

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$\begin{aligned}
f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\
&= \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}\right) + \quad \text{(even)} \\
&\quad \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots + a_n x^n\right) \quad \text{(odd)} \\
&= \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}\right) + \quad \text{(even)} \\
&\quad x\left(a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_n x^{n-1}\right) \quad \text{(odd)} \\
&= \left(a_0 + a_2 (x^2)^1 + a_4 (x^2)^2 + \cdots + a_{n-1} (x^2)^{\frac{n-1}{2}}\right) + \quad \text{(even)} \\
&\quad x\left(a_1 + a_3 (x^2)^1 + a_5 (x^2)^2 + \cdots + a_n (x^2)^{\frac{n-1}{2}}\right) \quad \text{(odd)}
\end{aligned}$$

Recursive breakdown of polynomials

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$
\begin{aligned}
f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\
&= \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}\right) + \quad \text{(even)} \\
&\quad \left(a_1 x + a_3 x^3 + a_5 x^5 + \cdots + a_n x^n\right) \quad \text{(odd)} \\
&= \left(a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{n-1} x^{n-1}\right) + \quad \text{(even)} \\
&\quad x\left(a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_n x^{n-1}\right) \quad \text{(odd)} \\
&= \left(a_0 + a_2 (x^2)^1 + a_4 (x^2)^2 + \cdots + a_{n-1} (x^2)^{\frac{n-1}{2}}\right) + \quad \text{(even)} \\
&\quad x\left(a_1 + a_3 (x^2)^1 + a_5 (x^2)^2 + \cdots + a_n (x^2)^{\frac{n-1}{2}}\right) \quad \text{(odd)} \\
&=: f_0(x^2) + x f_1(x^2),
\end{aligned}
$$

for $f_0(x) := a_0 + a_2 x + \cdots a_{n-1} x^{(n-1)/2}$ and $f_1(x) := a_1 + a_3 x + \cdots a_n x^{(n-1)/2}$.

Recursive breakdown of polynomials

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$\begin{aligned}
f(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1} + a_nx^n \\
&= \left(a_0 + a_2x^2 + a_4x^4 + \cdots + a_{n-1}x^{n-1}\right) + \quad \text{(even)} \\
&\quad \left(a_1x + a_3x^3 + a_5x^5 + \cdots + a_nx^n\right) \quad \text{(odd)} \\
&= \left(a_0 + a_2x^2 + a_4x^4 + \cdots + a_{n-1}x^{n-1}\right) + \quad \text{(even)} \\
&\quad x\left(a_1 + a_3x^2 + a_5x^4 + \cdots + a_nx^{n-1}\right) \quad \text{(odd)} \\
&= \left(a_0 + a_2(x^2)^1 + a_4(x^2)^2 + \cdots + a_{n-1}(x^2)^{\frac{n-1}{2}}\right) + \quad \text{(even)} \\
&\quad x\left(a_1 + a_3(x^2)^1 + a_5(x^2)^2 + \cdots + a_n(x^2)^{\frac{n-1}{2}}\right) \quad \text{(odd)} \\
&=: f_0(x^2) + xf_1(x^2),
\end{aligned}$$

for $f_0(x) := a_0 + a_2x + \cdots a_{n-1}x^{(n-1)/2}$ and $f_1(x) := a_1 + a_3x + \cdots a_nx^{(n-1)/2}$.

Observe:

- f_0 and f_1 have degree $(n-1)/2$!
- "Feels like" we've cut our problem into a pair of smaller problems of half the size.

The key step

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\ &= f_0(x^2) + x f_1(x^2), \end{aligned} \tag{3}$$

for $f_0(x) := a_0 + a_2 x + \cdots a_{n-1} x^{(n-1)/2}$ and $f_1(x) := a_1 + a_3 x + \cdots a_n x^{(n-1)/2}$.

The key step

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\ &= f_0(x^2) + x f_1(x^2), \end{aligned} \tag{3}$$

for $f_0(x) := a_0 + a_2 x + \cdots a_{n-1} x^{(n-1)/2}$ and $f_1(x) := a_1 + a_3 x + \cdots a_n x^{(n-1)/2}$.

- DFT_N evaluates degree-$(N-1)$ polynomial at Nth roots of unity.

The key step

Assume (WLOG) that $N = n + 1$ is a power of 2.

$$\begin{aligned} f(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \\ &= f_0(x^2) + x f_1(x^2), \end{aligned} \tag{3}$$

for $f_0(x) := a_0 + a_2 x + \cdots a_{n-1} x^{(n-1)/2}$ and $f_1(x) := a_1 + a_3 x + \cdots a_n x^{(n-1)/2}$.

- DFT_N evaluates degree-$(N-1)$ polynomial at Nth roots of unity.
- By Halving Lemma: Letting $x = \omega_N^k$ in Eqn. (3),

$$f_0(x^2) = f_0((\omega_N^k)^2) = f_0(\omega_{N/2}^k).$$

- Thus, roots of unity are very special — allow us to recursively simulate order-N DFT via order-$(N/2)$ DFTs.

FFT Algorithm

Preconditions: $N = n + 1$ is a power of 2.

Input: Coefficient vector $\mathbf{a} = (a_0, a_1, \ldots, a_n)^T$ representing polynomial $f \in \mathbb{C}[x]$.
Output: $\mathrm{DFT}_N\,\mathbf{a} = (f(\omega_N^0), f(\omega_N^1), \ldots, f(\omega_N^{N-1}))^T$.

FFT($\mathbf{a}$,N):

1. (Base case) if $N = 1$, return $\mathbf{a}$ (why?)
2. Set $\omega = 1$
3. Set $\mathbf{y}^{[0]} = \mathrm{FFT}((a_0, a_2, \ldots, a_{n-1}), N/2)$
4. Set $\mathbf{y}^{[1]} = \mathrm{FFT}((a_1, a_3, \ldots, a_n), N/2)$
5. for k from 0 to $N/2 - 1$ do
    1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$
    2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$
    3. Set $\omega = \omega\omega_N$
6. Return $\mathbf{y}$

Analysis

Let $f^{[j]}$ denote polynomial with coefficients $\mathbf{y}^{[j]}$ below.

Input: Coefficient vector $\mathbf{a} = (a_0, a_1, \ldots, a_n)^T$ representing polynomial $f \in \mathbb{C}[x]$.
Output: $\text{DFT}_N\, \mathbf{a} = (f(\omega_N^0), f(\omega_N^1), \ldots, f(\omega_N^{N-1}))^T$ for $N = n + 1$.

FFT($\mathbf{a}$,N):

1. (Base case) if $N = 1$, return $\mathbf{a}$
2. Set $\omega = 1$
3. Set $\mathbf{y}^{[0]} = \text{FFT}((a_0, a_2, \ldots, a_{n-1}), N/2)$
4. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

Analysis

Let $f^{[j]}$ denote polynomial with coefficients $\mathbf{y}^{[j]}$ below.

Input: Coefficient vector $\mathbf{a} = (a_0, a_1, \ldots, a_n)^T$ representing polynomial $f \in \mathbb{C}[x]$.
Output: $\text{DFT}_N\, \mathbf{a} = (f(\omega_N^0), f(\omega_N^1), \ldots, f(\omega_N^{N-1}))^T$ for $N = n + 1$.

FFT($\mathbf{a}$,N):

1. (Base case) if $N = 1$, return $\mathbf{a}$
2. Set $\omega = 1$
3. Set $\mathbf{y}^{[0]} = \text{FFT}((a_0, a_2, \ldots, a_{n-1}), N/2)$
4. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$ (why?)

Analysis

Let $f^{[j]}$ denote polynomial with coefficients $\mathbf{y}^{[j]}$ below.

Input: Coefficient vector $\mathbf{a} = (a_0, a_1, \ldots, a_n)^T$ representing polynomial $f \in \mathbb{C}[x]$.
Output: $\mathrm{DFT}_N\, \mathbf{a} = (f(\omega_N^0), f(\omega_N^1), \ldots, f(\omega_N^{N-1}))^T$ for $N = n + 1$.

FFT($\mathbf{a}$,N):

1. (Base case) if $N = 1$, return $\mathbf{a}$
2. Set $\omega = 1$
3. Set $\mathbf{y}^{[0]} = \mathrm{FFT}((a_0, a_2, \ldots, a_{n-1}), N/2)$
4. Set $\mathbf{y}^{[1]} = \mathrm{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$ (why?)
5. for k from 0 to $N/2 - 1$ do
   1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$

      $y_k = f^{[0]}(\omega_N^{2k}) + \omega_N^k f^{[1]}(\omega_N^{2k}) = f(\omega_N^k)$ by recursive decomposition
   2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$
   3. Set $\omega = \omega\omega_N$
6. Return $\mathbf{y}$

Analysis

1. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$

2. for k from 0 to $N/2 - 1$ do

   1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$

      $y_k = f^{[0]}(\omega_N^{2k}) + \omega_N^k f^{[1]}(\omega_N^{2k}) = f(\omega_N^k)$ by recursive decomposition

   2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$

      Q: Why treat indices in range $N/2, \ldots, N-1$ differently?

Analysis

1. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$
2. for k from 0 to $N/2 - 1$ do
   1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$

      $y_k = f^{[0]}(\omega_N^{2k}) + \omega_N^k f^{[1]}(\omega_N^{2k}) = f(\omega_N^k)$ by recursive decomposition
   2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$

      Q: Why treat indices in range $N/2, \ldots, N-1$ differently?

$$y_{k+(N/2)} = y_k^{[0]} - \omega_N^k y_k^{[1]} = y_k^{[0]} + \omega_N^{k+(N/2)} y_k^{[1]} \quad (\omega_N^{N/2} = -1 \text{ for even } N)$$

Analysis

1. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$

2. for k from 0 to $N/2 - 1$ do

   1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$

      $y_k = f^{[0]}(\omega_N^{2k}) + \omega_N^k f^{[1]}(\omega_N^{2k}) = f(\omega_N^k)$ by recursive decomposition

   2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$

      Q: Why treat indices in range $N/2, \ldots, N-1$ differently?

$$\begin{aligned} y_{k+(N/2)} &= y_k^{[0]} - \omega_N^k y_k^{[1]} = y_k^{[0]} + \omega_N^{k+(N/2)} y_k^{[1]} \quad (\omega_N^{N/2} = -1 \text{ for even } N) \\ &= f^{[0]}(\omega_N^{2k}) + \omega_N^{k+(N/2)} f^{[1]}(\omega_N^{2k}) \end{aligned}$$

Analysis

1. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$

2. for k from 0 to $N/2 - 1$ do

   1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$

      $y_k = f^{[0]}(\omega_N^{2k}) + \omega_N^k f^{[1]}(\omega_N^{2k}) = f(\omega_N^k)$ by recursive decomposition

   2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$

      Q: Why treat indices in range $N/2, \ldots, N-1$ differently?

$$\begin{aligned} y_{k+(N/2)} &= y_k^{[0]} - \omega_N^k y_k^{[1]} = y_k^{[0]} + \omega_N^{k+(N/2)} y_k^{[1]} \quad (\omega_N^{N/2} = -1 \text{ for even } N) \\ &= f^{[0]}(\omega_N^{2k}) + \omega_N^{k+(N/2)} f^{[1]}(\omega_N^{2k}) \\ &= f^{[0]}(\omega_N^{2k+N}) + \omega_N^{k+(N/2)} f^{[1]}(\omega_N^{2k+N}) \quad (\omega_N^N = 1 \text{ by definition}) \end{aligned}$$

Analysis

1. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$

2. for k from 0 to $N/2 - 1$ do

   1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$

      $y_k = f^{[0]}(\omega_N^{2k}) + \omega_N^k f^{[1]}(\omega_N^{2k}) = f(\omega_N^k)$ by recursive decomposition

   2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$

      Q: Why treat indices in range $N/2, \ldots, N-1$ differently?

$$\begin{aligned} y_{k+(N/2)} &= y_k^{[0]} - \omega_N^k y_k^{[1]} = y_k^{[0]} + \omega_N^{k+(N/2)} y_k^{[1]} \quad (\omega_N^{N/2} = -1 \text{ for even } N) \\ &= f^{[0]}(\omega_N^{2k}) + \omega_N^{k+(N/2)} f^{[1]}(\omega_N^{2k}) \\ &= f^{[0]}(\omega_N^{2k+N}) + \omega_N^{k+(N/2)} f^{[1]}(\omega_N^{2k+N}) \quad (\omega_N^N = 1 \text{ by definition}) \\ &= f(\omega_N^{k+(N/2)}) \quad \text{(by recursive decomposition)} \end{aligned}$$

Analysis

1. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$

   $y_k^{[0]} = f^{[0]}(\omega_{N/2}^k) = f^{[0]}(\omega_N^{2k})$ and $y_k^{[1]} = f^{[1]}(\omega_{N/2}^k) = f^{[1]}(\omega_N^{2k})$

2. for k from 0 to $N/2 - 1$ do

   1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$

      $y_k = f^{[0]}(\omega_N^{2k}) + \omega_N^k f^{[1]}(\omega_N^{2k}) = f(\omega_N^k)$ by recursive decomposition

   2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$

      Q: Why treat indices in range $N/2, \ldots, N-1$ differently?

      $$\begin{aligned} y_{k+(N/2)} &= y_k^{[0]} - \omega_N^k y_k^{[1]} = y_k^{[0]} + \omega_N^{k+(N/2)} y_k^{[1]} \quad (\omega_N^{N/2} = -1 \text{ for even } N) \\ &= f^{[0]}(\omega_N^{2k}) + \omega_N^{k+(N/2)} f^{[1]}(\omega_N^{2k}) \\ &= f^{[0]}(\omega_N^{2k+N}) + \omega_N^{k+(N/2)} f^{[1]}(\omega_N^{2k+N}) \quad (\omega_N^N = 1 \text{ by definition}) \\ &= f(\omega_N^{k+(N/2)}) \quad \text{(by recursive decomposition)} \end{aligned}$$

   3. Set $\omega = \omega\omega_N$

3. Return $\mathbf{y}$

Input: Coefficient vector $\mathbf{a} = (a_0, a_1, \ldots, a_n)^T$ representing polynomial $f \in \mathbb{C}[x]$.
Output: $\mathrm{DFT}_N\,\mathbf{a} = (f(\omega_N^0), f(\omega_N^1), \ldots, f(\omega_N^{N-1}))^T$.

FFT($\mathbf{a}$,N):

1. (Base case) if $N = 1$, return $\mathbf{a}$
2. Set $\omega = 1$
3. Set $\mathbf{y}^{[0]} = \mathrm{FFT}((a_0, a_2, \ldots, a_{n-1}), N/2)$
4. Set $\mathbf{y}^{[1]} = \mathrm{FFT}((a_1, a_3, \ldots, a_n), N/2)$
5. for k from 0 to $N/2 - 1$ do
    1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$
    2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$
    3. Set $\omega = \omega\omega_N$
6. Return $\mathbf{y}$

Input: Coefficient vector $\mathbf{a} = (a_0, a_1, \ldots, a_n)^T$ representing polynomial $f \in \mathbb{C}[x]$.
Output: $\text{DFT}_N\,\mathbf{a} = (f(\omega_N^0), f(\omega_N^1), \ldots, f(\omega_N^{N-1}))^T$.

FFT($\mathbf{a}$,N):

1. (Base case) if $N = 1$, return $\mathbf{a}$
2. Set $\omega = 1$
3. Set $\mathbf{y}^{[0]} = \text{FFT}((a_0, a_2, \ldots, a_{n-1}), N/2)$
4. Set $\mathbf{y}^{[1]} = \text{FFT}((a_1, a_3, \ldots, a_n), N/2)$
5. for k from 0 to $N/2 - 1$ do
    1. Set $y_k = y_k^{[0]} + \omega y_k^{[1]}$
    2. Set $y_{k+(N/2)} = y_k^{[0]} - \omega y_k^{[1]}$
    3. Set $\omega = \omega\omega_N$
6. Return $\mathbf{y}$

Runtime

- Each call to FFT takes $O(N)$ time, and makes two recursive calls of size $N/2$.
- Thus, runtime $T(N) = 2T(N/2) + \Theta(N) \in \Theta(N \log N)$.
- Conclusion: Evaluate degree-$(N-1)$ polynomial at Nth roots of unity in subquadratic time.

The big picture

Recall our battle plan for polynomial multiplication:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.

The big picture

Recall our battle plan for polynomial multiplication:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.

Can now fill in details:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at Nth roots of unity.
   - Takes $O(N \log N)$ time via FFT.

The big picture

Recall our battle plan for polynomial multiplication:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.

Can now fill in details:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at Nth roots of unity.
   - Takes $O(N \log N)$ time via FFT.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.

The big picture

Recall our battle plan for polynomial multiplication:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.

Can now fill in details:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at Nth roots of unity.
   - Takes $O(N \log N)$ time via FFT.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.
   - Claim. Interpolation corresponds to *inverse* DFT. Takes $O(N \log N)$ time.

The big picture

Recall our battle plan for polynomial multiplication:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at cleverly chosen points.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.

Can now fill in details:

1. Convert $f, g \in \mathbb{C}[x]$ from *coefficient representation* to *point-value representation* by evaluating f, g at Nth roots of unity.
   - Takes $O(N \log N)$ time via FFT.
2. Multiplication in point-value representation takes only $\Theta(n)$ time.
3. Convert back from point-value representation to coefficient representation (i.e. *interpolate*) to recover final answer.
   - Claim. Interpolation corresponds to *inverse* DFT. Takes $O(N \log N)$ time.

Conclusion: Polynomial multiplication takes $O(N \log N)$ time.

Final exercises

1. Why does interpolation correspond to inverse DFT? (Hint: Recall proof of Interpolation Theorem.)
2. What is the matrix representation of the inverse of DFT_N?
3. We cheated slightly in our algorithm — where did we bend the rules? (Hint: How many data points did we need to evaluate a polynomial at in order to recover a unique inverse via interpolation?)

Looking to the future

- You didn't think applications of Fourier transform end with *today's* technology?

[4] The best known classical factoring algorithms take superpolynomial time.

Looking to the future

- You didn't think applications of Fourier transform end with *today's* technology?
- Using a *quantum* implementation of DFT_N, one can exponentially outperform[4] classical computers at a crucial problem: The Integer Factorization Problem:

[4] The best known classical factoring algorithms take superpolynomial time.

Looking to the future

- You didn't think applications of Fourier transform end with *today's* technology?
- Using a *quantum* implementation of DFT_N, one can exponentially outperform[4] classical computers at a crucial problem: The Integer Factorization Problem:
  - Given an integer M, what are the prime factors of M?

[4] The best known classical factoring algorithms take superpolynomial time.

Looking to the future

- You didn't think applications of Fourier transform end with *today's* technology?
- Using a *quantum* implementation of DFT_N, one can exponentially outperform[4] classical computers at a crucial problem: The Integer Factorization Problem:
  - Given an integer M, what are the prime factors of M?
  - The assumption that this problem is classically hard underlies security of common cryptosystems like RSA.

[4] The best known classical factoring algorithms take superpolynomial time.

Looking to the future

- You didn't think applications of Fourier transform end with *today's* technology?
- Using a *quantum* implementation of DFT_N, one can exponentially outperform[4] classical computers at a crucial problem: The Integer Factorization Problem:
  - Given an integer M, what are the prime factors of M?
  - The assumption that this problem is classically hard underlies security of common cryptosystems like RSA.
  - Thus, a large-scale quantum computer would break today's cryptosystems completely…

[4] The best known classical factoring algorithms take superpolynomial time.

Looking to the future

- You didn't think applications of Fourier transform end with *today's* technology?
- Using a *quantum* implementation of DFT_N, one can exponentially outperform[4] classical computers at a crucial problem: The Integer Factorization Problem:
  - Given an integer M, what are the prime factors of M?
  - The assumption that this problem is classically hard underlies security of common cryptosystems like RSA.
  - Thus, a large-scale quantum computer would break today's cryptosystems completely...
- Shameless advertisements:
  - See upcoming Masters lecture on Introduction to Quantum Computation (SS2020)
  - Interested in undergraduate research in quantum computation? Come talk to me! Required background is Linear Algebra.

[4] The best known classical factoring algorithms take superpolynomial time.