Numerical Computation

So far, several algorithms via few basic techniques

- Eigenvalues of matrices (e.g. Cheeger)
- SDPs (e.g. max-cut)
- LPs (e.g. set cover)
- Linear equation solving (e.g. sat instances of UG)

⋮

How do we implement these primitives?

→ Several of the times we applied these methods, the inputs, the outputs or sometimes both were real numbers.

But our computers & our models for them are still "digital".

Every algorithm eventually should be a WORD-RAM m/c. (1) RAM memory
(2) w: word size, $w \sim \Theta(\log n)$.
(3) arithmetic operations on w bit integers can be done in $O(1)$ time.

What about real numbers?
How do we compute on real numbers?

<u>Answer</u>: We don't.

<u>Less glib answer</u>: Must "approximate" by rationals & hope for the best.

You might think such details just work out. But they can be anywhere from easy to non-trivial to impossible.

In fact, it is often assumed that basic operations on real #s can be done in $O(1)$ time. What are basic operations? One might hope that these can be implemented on TMs/digital computers with poly(n) slowdown. So no harm, no foul & great convenience.

What are basic operations?

Addition

Multiplication

Division?

Square roots?

rounding to integers? modulus?

Some subset appears to make the model unreasonable (solves PSPACE)

Can we just keep arithmetic ops and work to get reasonable results?

Also fraught with some issues.

SUM OF SQUARE ROOTS PROBLEM

Given 2 polygons in plane with integer coordinate vertices, decide which one has a larger perimeter?

$(x_1, y_1) \ldots (x_n, y_n) \to$ Polygon 1

$(u_1, v_1), \ldots, (u_n, v_n) \to$ Polygon 2

$$\sum_i \sqrt{(y_i - y_{i-1})^2 + (x_i - x_{i-1})^2} \overset{?}{<} \sum_i \sqrt{(u_i - u_{i-1})^2 + (v_i - v_{i-1})^2}$$

Similar issues in ...

Euclidean TSP, Euclidean Shortest Paths, ...

Can give a PSPACE algo for SOSR

(also known to be in $P^{PP^{PP^{PP}}}$
(4th level of counting hierarchy).).

Not known to be even in NP

O(n) time on "real RAM".

Blömer: poly time randomized algo to decide if two sums are equal. but cannot decide which is larger.

Moral: Got to be somewhat careful

of numerical issues. Can't always expect them to work themselves out.

<u>Today</u>: Solving Linear Equations

→ input numbers are rationals.

→ rationals described by a pair of integers.

<u>Obs^n</u>: given $\frac{p_1}{q_1}$, $\frac{p_2}{q_2}$, can

1) add 2) multiply 2) divide

in poly(input-size) time.

Why? 1) $\frac{p_1}{q_1} + \frac{p_2}{q_2} = \frac{p_1 \cdot q_2 + p_2 \cdot q_1}{q_1 q_2}$

If all p_i, q_i are m-bit long integers, then their sum is at most $2m+2$ bit integers in num/den.

2) division: $\frac{p_1 q_2}{q_1 \cdot p_2}$ ⤳ $2m$ bit long

3) multiplication: $\frac{p_1 \cdot p_2}{q_1 \cdot q_2}$ ⤳ $2m$ bit long.

Let's now consider the primitives we saw: . .

Linear Equation Solving

Given $A \in \mathbb{Q}^{n \times n}$, $b \in \mathbb{Q}^n$, find x s.t. $Ax = b$ if it exists.

Is there a poly-size representable solution?

Cramer's rule : $x_i = \dfrac{\det(A_i)}{\det(A)}$

where A_i : replace i^{th} col by b_i

Prop : If all entries of A are integers of $\leq b$ bits, $\det(A)$ is an integer of $\leq n \log n + n \cdot \log b$ bits

Proof: $\det(A) = \sum_{\substack{\sigma:[n]\to[n] \\ \text{perm.}}} \prod_{i\le n} A(i,\sigma(i))\cdot \operatorname{sgn}(\sigma)$

$$|\det(A)| \le n!\, \max_{\sigma} \left|\prod_{i\le n} A(i,\sigma(i))\right|$$

$$\le n!\, 2^{b\cdot n}. \qquad \square.$$

Thus, each $x_i = \dfrac{\det(A_i)}{\det(A)}$ can be written in $\le O(n\cdot\log n + n\cdot b)$ bits. $=$ poly(input-size) bits. if A is an integer-entry matrix

What if A's entries are not integers?

Can take "common denominator" of n^2 entries. If all entries are b bit long, the common denominator ~~ie~~ $\leq n^2 \cdot b$ bits long.

<u>So</u> making A's entries integers amounts to multiplying all entries by an integer of $\leq n^2 \cdot b$ bits. $\rightarrow$ at most poly blow up to the bit complexity.

(b-bit entries $\longrightarrow$ $\leq n^2 \cdot b$ bit
rationals (integer entries)

Eigenvalues of matrix $A \in \mathbb{Q}^{n \times n}$?

$A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$: eig $= \pm\sqrt{2}$.

even integer matrices can have irrational eigs.

So, must resort to approximation.

Note: can write down an ε error approx in $O(\log 1/\varepsilon)$ bits.

Linear programming

$$\max \; c^T x \qquad c, A \in \mathbb{Q}^{m \times n}$$

$$\text{s.t. } Ax \le b \qquad \in \mathbb{Q}^n$$

$$\mathbb{R}^n \ni 0 \le x \le B \leftarrow \text{poly}(n) \text{ bit integer.}$$

Fact: If LP is feasible then there is an optimal solution with poly(n, m, b) bits where b = bit complexity of entries of A, b

Can we always have LPs in the form above?

"$x_i \le B$" $\rightarrow$ "Bounding Box" Constraint

In our applications, B can often be 1.

We will now see an algorithm to solve linear equations over $\mathbb{Q}$ in polynomial time in the size of the input.

Input: $A \in \mathbb{Z}^{n \times n} = (a_{ij})$ of integers of at most m bits.

$b \in \mathbb{Z}^n$: with $\leq m$ bit integer entries.

Goal: find x s.t. $Ax = b$. if it exists.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & & a_{nn} \end{pmatrix} \in n \times n.$$

Obs^n: the following operations do not change solution set of the equations.

(1) non-zero Scaling equations.

$$\sum_i a_{1i} x_i = b_i$$

$$|||$$

$$\sum_i s \cdot a_{1i} x_i = s \cdot b_i$$

(2) Subtracting an equation from another

<u>Idea</u>: 1) Subtract appropriate scalings of the 1st equation from all others to make

1st col 0 for rows $2 \leq i \leq n$: $A = A^{(0)} \to A^{(1)}$.

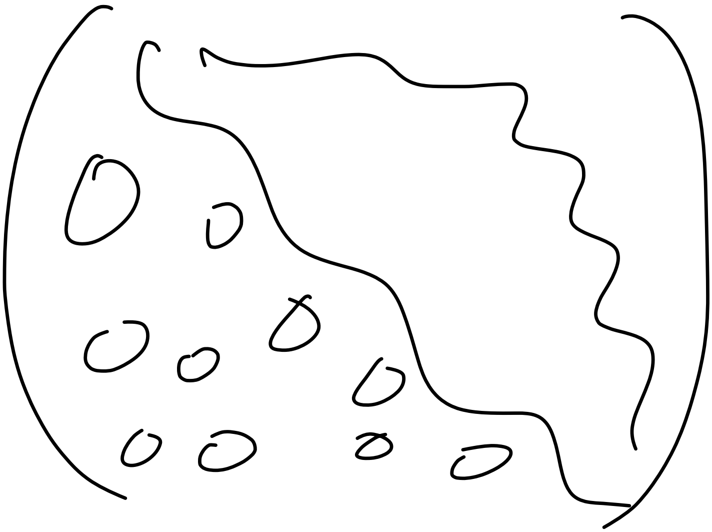
2) Subtract appropriate scalings of the 2nd equation to make 2nd column 0 for equations

$i = 3, \dots, n$ $\qquad A^{(1)} \to A^{(2)}$.

$\vdots \qquad \vdots$

$A^{(n)}$

At the end, we have a Upper triangular matrix $= A^{(n)}$

 $X = b$

easy to solve by "solve & substitute"

learn X_n from last equation.

substitute X_n in 2^{nd} last equation to learn X_{n-1}

and so on...

<u>Pivots</u>: Our discussion assumed that $a_{11} = 0$. And then for the matrix $A^{(1)}$ obtained after zeroing first column, that $a_{22}^{(1)} = 0$

$$\vdots$$

What if in $A^{(k-1)}$, $a_{kk}^{(k-1)} = 0$?

<u>Idea</u>: 1) If equation i for $k \le i \le n$ (= row i of $A^{(k-1)}$) has a non-zero entry in column k, can "swap" to make row i = row k of $A^{(k-1)}$)

2) What if all rows i for $k \le i \le n$ have zero in k-th column?

If some column $k \le j \le n$ has a non-zero entry in a row $k \le i \le n$ then swap j^{th} column with k^{th}.

row swap = re-ordering equations
col swap = renaming variables.

3) What if the whole block
rows $k \le i \le n$
cols $k \le j \le n$
is all zeros?

Then, done! We've identified the affine subspace of solutions!

What's the time complexity

Each elimination step involves n multiplications, n^2 subtractions and additions. So $O(n^2)$ arithmetic operations on b-bit numbers.

There are n total elimination steps. $\rightarrow$ So $O(n^3)$ arithmetic operations.

Substitute and solve takes

$$O(1) + O(2) + \dots + O(n)$$
$$= O(n^2).$$

So all in all, we take $O(n^3)$ arithmetic operations.

But is that the running time?

In the 1^{st} step, we are doing arithmetic on b bit #s.

So takes $\mathrm{poly}(b)$ time for each operation.

But what about subsequent steps?

Let's write down what happens to the entries of the matrix after the k-th elimination step.

Let $a_{ij}^{(k-1)}$: entries before k^{th} elimination step.

$a_{ij}^{(k)}$: after k^{th} elimination step.

Then, if $i \le k, j \le k-1$, $a_{ij}^{(k-1)} = a_{ij}^{(k)}$

Since we only modify rows k through n in the k^{th} elimination step.

If $i > k$, $j \le k-1$, then

$$a_{ij}^{(k-1)} = a_{ij}^{(k)} = 0.$$

We can assume WLOG that $a_{k,k}^{(k-1)}$ is the pivot (otherwise we can do some row/col exchanges)

For $i \geq k+1$, $j \geq k$,

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{k,j}^{(k-1)} \cdot \frac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}} \qquad -(1)$$

Note that this ensures that

$$a_{i,k}^{(k)} = 0 \quad \forall \; i \geq k+1$$

(1) is same as

$$a_{ij}^{(k)} = \frac{a_{ij}^{(k-1)} \cdot a_{k,k}^{(k-1)} - a_{kj}^{(k-1)} a_{ik}^{(k-1)}}{a_{k,k}^{(k-1)}}.$$

$$\begin{array}{cc} & \begin{array}{cc} k & \quad j \end{array} \\ \begin{array}{c} k \\ i \end{array} & \begin{pmatrix} a_{kk}^{(k-1)} & a_{kj}^{(k-1)} \\ a_{ik}^{(k-1)} & a_{ij}^{(k-1)} \end{pmatrix} \end{array}$$

$\hookrightarrow D^{(k-1)} \in \mathbb{Q}^{2\times 2}$

$$\boxed{a_{ij}^{(k)} = \frac{\mathrm{Det}(D^{(k-1)})}{a_{kk}^{(k-1)}}.}$$

Examining ①, observe that the

$$\frac{a_{kj}^{(k-1)} \cdot a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}$$ involves multiplying

2 entries of $A^{(k-1)}$. So bit

complexity becomes twice that

of the entries of $A^{(k-1)}$.

$$\text{bitcomplexity}(A^{(k)}) \leq 2\,\text{bit-comp}(A^{(k-1)}) + \ldots$$

best we can expect from this recursive

bound is

$$\therefore \text{bit-complexity}(A^{(n)}) \leq 2^n \cdot b$$

exponentially large numbers.

Thus, the naive upper bound seems to suggest a exponential blow up in the bit complexity of the intermediate rational numbers obtained.

It turns out that this ~~bound~~ is pessimistic. The key observation is the following claim of Edmonds (1967)

Lemma: For every k, every entry of $A^{(k)}$ is a ratio of determinants of two submatrices of A.

Notice that this lemma immediately implies that the bit complexity of all entries of $A^{(k)}$ is at most

$$\log(n!\,2^{bn}) \leq n\log_2 n + nb.$$

Proof of lemma: Very slick proof.

Key Fact: If you take any matrix A and produce A' by operations of the form

$$a_i' = a_i - \text{scaling} \cdot a_j$$

a_i': i^{th} row of A'

a_i: i^{th} row of A

scaling: any rational #.

"row operations"

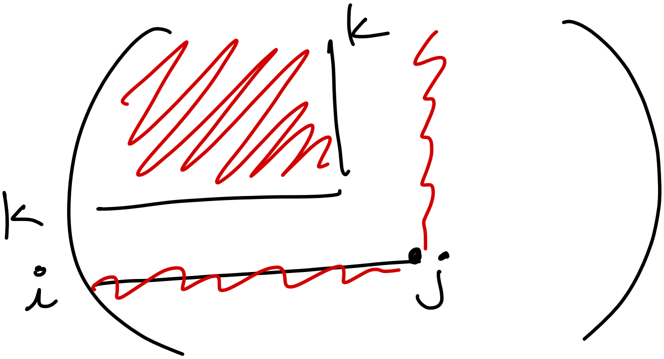
then $\det(A') = \det(A)$.

We will prove: Let $D^{(k)}$ be the matrix $(k \times k)$ with 1st k rows & cols of $A^{(k)}$.

Let $D_{ij}^{(k)}$ be the $k+1$ by $k+1$ matrix with 1^{st} k rows & i^{th} row of $A^{(k)}$ & 1^{st} k cols & j^{th} col of $A^{(k)}$. Then

$$a_{ij}^{(k)} = \frac{\det(D_{ij}^{(k)})}{\det(D^{(k)})}$$

Consider the matrix



Note: Can assume $i, j \geq k+1$.

If $i \leq k$, then $a_{ij}^{(k)} = a_{ij}^{(k-1)}$.

If $j \leq k$, then $a_{ij}^{(k)} = 0$

$D_{ij}^{(k)}$: first k rows & i^{th} row of $A^{(k)}$

first k cols & j^{th} col of $A^{(k)}$.

$D^{(k)}$: first k rows & cols of $A^{(k)}$

Then note that

$* \leftarrow \det(D^{(k)}) = a_{11}^{(k)} \cdot a_{22}^{(k)} \cdots a_{kk}^{(k)}$.

$\& \det(D_{ij}^{(k)}) = a_{11}^{(k)} \cdot a_{22}^{(k)} \cdots a_{kk}^{(k)} \cdot a_{ij}^{(k)}$

$**$

Notice that both $D^{(k)}$ & $D_{ij}^{(k)}$ are upper triangular. We are then using:

<u>Fact</u>: For any upper (or lower) Δ-ular M, $\det(M) =$ product of diagonal entries.

To complete the proof, note that $A^{(k)}$ is obtained by row operations (i.e. subtracting of scaled copies of rows).

<u>So</u>

$$\det(D_{ij}^{(k)}) = \det(D_{ij}^{(0)})$$

$$\det(D^{(k)}) = \det(D^{(0)})$$

determinants of submatrices of A.

$\square$