# CS 4424
# (Sparse) linear systems

# Linear systems

## Example

$$\begin{aligned} x_1 + x_2 - 3x_3 &= 3 \\ -x_1 + 3x_2 - x_3 &= 0 \\ 10x_1 + 3x_2 - x_3 &= 5 \end{aligned}$$

## Matrix form

$$\begin{bmatrix} 1 & 1 & -3 \\ -1 & 3 & -1 \\ 10 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 5 \end{bmatrix}.$$

# Dense vs sparse

## Dense

- matrices where most entries are non-zero
- usually, not that big (a few hundreds or thousands)
- Gauss' algorithm, $O(n^3)$ (later)

## Sparse

- matrices where most entries are zero
- can be very big (millions of rows / columns)
- iterative algorithms, aiming at $O(n^2)$

# Reminder: matrix multiplication

Matrix-vector product. Starting from

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix},$$

we get

$$Ab = \begin{bmatrix} a_{1,1}b_1 + \cdots + a_{1,n}b_n \\ \vdots \\ a_{n,1}b_1 + \cdots + a_{n,n}b_n \end{bmatrix}.$$

# Reminder: matrix multiplication

Matrix-vector product. Starting from

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} b_1 & \cdots & b_n \end{bmatrix},$$

we get

$$bA = \begin{bmatrix} a_{1,1}b_1 + \cdots + a_{n,1}b_n & \cdots & a_{1,n}b_1 + \cdots + a_{n,n}b_n \end{bmatrix}.$$

# Reminder: matrix multiplication

Matrix-matrix product. Starting from

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & & \vdots \\ b_{n,1} & \cdots & b_{n,n} \end{bmatrix},$$

we get $AB$ by multiplying $A$ by all columns of $B$ (or all rows of $A$ by $B$).

Explicitly,

$$AB = \begin{bmatrix} \cdots & a_{1,1}b_{1,j} + \cdots + a_{1,n}b_{n,j} & \cdots \\ & \vdots & \\ \cdots & a_{n,1}b_{1,j} + \cdots + a_{n,n}b_{n,j} & \cdots \end{bmatrix}.$$

Particular case: matrix powers

$$A^0 = I, \quad A, \quad A^2, \quad \ldots$$

# Linear recurrence for matrices

Given a univariate polynomial

$$P = p_0 + p_1 x + \cdots + p_d x^d$$

and a matrix $A$, we can define

$$P(A) = p_0 I + p_1 A + \cdots + p_d A^d.$$

Prop-def

- For any matrix $A$ of size $n$, there exists one (or more) polynomial(s) $P$ of degree at most $n$ such that $P(A) = 0$.
- The monic polynomial $P$ of smallest degree such that $P(A) = 0$ is the minimal polynomial of $A$.
- Written $m_A$.

# Examples

Simplest case: diagonal matrices

$$A = \begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix} \implies m_A = x - c$$

Less easy. Take

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

We get

$$A^0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad A^1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad A^2 = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix},$$

and

$$m_A = x^2 - 5x - 2.$$

# From matrix to scalar

Given a matrix $A$ of size $n$, choose (at random) two vectors

$$u = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \quad \text{and} \quad v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}.$$

Then, define

$$a_0 = uA^0v, \quad a_1 = uA^1v, \quad \ldots, \quad a_i = uA^iv, \quad \ldots$$

Example With $A$ as before and

$$u = \begin{bmatrix} 1 & 2 \end{bmatrix}, \quad \text{and} \quad v = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

we get

$$a_0 = -3, \quad a_1 = -13, \quad a_2 = -71, \quad a_3 = -381, \quad a_4 = -2047, \quad a_5 = -10997, \quad \ldots$$

# Scalar linear recurrence

Starting from

$$p_0 I + p_1 A + \cdots + A^d = 0,$$

we can multiply on the left by $u$ and on the right by $v$, and we still get 0. This means that, for any choice of $u, v$, we have

$$p_0 a_0 + p_1 a_1 + \cdots + a_d = 0.$$

More generally: pre-multiply by $A^m$:

$$p_0 A^m + p_1 A^{m+1} + \cdots + A^{m+d} = 0,$$

multiply on the left by $u$, on the right by $v$, and get

$$p_0 a_m + p_1 a_{m+1} + \cdots + a_{m+d} = 0.$$

Conclusion: the sequence $(a_i)$ satisfies a linear recurrence with constant coefficients.

# Finding the minimal polynomial

## Algorithm

- compute $2n$ values $a_0, \ldots, a_{2n-1}$
- find the recurrence for $(a_i)$
- hope that it is the minimal polynomial of $A$.

## Prop

- For "most" choices of $u, v$, we find $m_A$.
- In unlucky situations, we may find a factor only.

## Example

- $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, $u = [1\ 0]$, any $v$.

# Solving systems

Let

$$m_A = p_0 + p_1 x \cdots + x^d,$$

and assume that $p_0 \neq 0$. Given a vector $c$, we can use $m_A$ to solve the system $Ab = c$

Prop.

- Define
$$b = -\frac{1}{p_0}(p_1 I + \cdots + A^{d-1})c.$$
- Then $Ab = c$.

# Complexity

1. Computing the $a_i$

   - remember: $a_i = uA^iv$
   - compute $v,\ Av,\ A^2v,\ \ldots$
   - deduce $uv,\ uAv,\ uA^2v,\ \ldots$

Total: $O(n)$ matrix-vector products by $A$, $O(n^2)$ other operations

2. Computing $m_A$: $O(n^2)$ by Euclid's algorithm

3. Computing $b$:

   - compute $c,\ Ac,\ A^2c,\ \ldots$
   - deduce $p_1c\ +\ p_2Ac\ +\ \cdots$

Total: $O(n)$ matrix-vector products by $A$, $O(n^2)$ other operations

# Summary

Total cost: $O(n)$ matrix-vector products by $A$, $O(n^2)$ other operations

Two cases:

- When $A$ is dense, a matrix-vector product by $A$ takes $O(n^2)$ operations, so the total is $O(n^3)$.

  Not really useful.

- Better when $A$ is sparse.

  Suppose that $A$ has $r$ non-zero entries. Then a matrix-vector product takes time $O(r)$; total: $O(rn + n^2)$.

# Example: factoring integers with Dixon's method

We want to factor an integer $N$.

Basic idea:

$$A^2 \text{ rem } N = B^2 \text{ rem } N \iff N \text{ divides } (A+B)(A-B)$$

$\implies$ we hope that $\gcd(N, A+B)$ and $\gcd(N, A-B)$ are not trivial.

Example: Take $N = 2183$; suppose we have "guessed" that

$$96002478^2 \text{ rem } N = 21^2 \text{ rem } N.$$

Then we get the gcd's

$$\gcd(96002478 + 21, N) = 59, \quad \gcd(96002478 - 21, N) = 37.$$

Notation: we say that $A \equiv B$ when $A \text{ rem } N = B \text{ rem } N$.

# Factor bases

We consider a few prime numbers $p_1, \ldots, p_S$, and we look for relations of the form

$$A^2 \equiv p_1^{\alpha_1} \cdots p_S^{\alpha_S}.$$

- If all $\alpha_i$ are even, $p_1^{\alpha_1} \cdots p_S^{\alpha_S}$ is of the form $B^2$.
- Else, we combine relations to get back to that case.

Example. $N = 2183$ and $p_1, \ldots, p_S = 3, 5, 7, 41$.

The relations modulo $N$

$$209^2 \equiv 3 \cdot 7, \quad 453^2 \equiv 7, \quad 1014^2 \equiv 3, \quad 1367^2 \equiv 41$$

give

$$(209 \cdot 453 \cdot 1014)^2 \equiv 21^2.$$

# A two-stage algorithm

1. Collect relations

   - try to decompose integers of the form $A^2$ rem $N$ on the factor basis
   - easy to distribute; low memory

2. Combine them

   - solve a large linear system with coefficients in $\{0, 1\}$.

Total cost: on average,

$$\exp\left(3\sqrt{2}\sqrt{\log(N)\log\log(N)}\right)$$

- naive algorithm: polynomial in $N$, which is

$$\exp\left(K\log(N)\right)$$

- input and output size: $\log(N)$.

# Combination: towards a linear system

We look for integers $e_1, e_2, e_3, e_4$ and $a_1, a_2, a_3, a_4$ such that

$$(209^{e_1} \cdot 453^{e_2} \cdot 1014^{e_3} \cdot 1367^{e_4})^2 \equiv (3^{a_1} \cdot 5^{a_2} \cdot 7^{a_3} \cdot 41^{a_4})^2 .$$

This gives:

$$209^{2e_1} \cdot 453^{2e_2} \cdot 1014^{2e_3} \cdot 1367^{2e_4} \equiv 3^{2a_1} \cdot 5^{2a_2} \cdot 7^{2a_3} \cdot 41^{2a_4}.$$

We know that

$$209^{2e_1} = 3^{e_1} \cdot 7^{e_1}, \quad 453^{2e_1} = 7^{e_1}, \ldots$$

So we get

$$3^{e_1+e_3} \cdot 5^0 \cdot 7^{e_1+e_2} \cdot 41^{e_4} \equiv 3^{2a_1} \cdot 5^{2a_2} \cdot 7^{2a_3} \cdot 41^{2a_4}.$$

In other words: $e_1 + e_3$ must be even, $e_1 + e_2$ must be even, $e_4$ must be even.

# Conclusion: linear system

We only look at the exponents modulo 2. This gives us the linear system over $\mathbb{Z}/2\mathbb{Z}$:

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \Longrightarrow e_1 = e_2 = e_3 = 1, e_4 = 0.$$

Summary

- look for a lot of small relations
- use them by solving a huge, sparse linear system.

In real life, one uses more complex algorithms; matrix size is a few millions.