

# **ORF307 – Optimization**

## **2. Solving linear systems in practice**

# Today's lecture

## Solving linear systems in practice

- Matrices: definition, operations, special cases
- Linear systems solutions
- Solving linear systems

# Matrices

# Matrices

**matrix** of size  $m \times n$

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix}$$

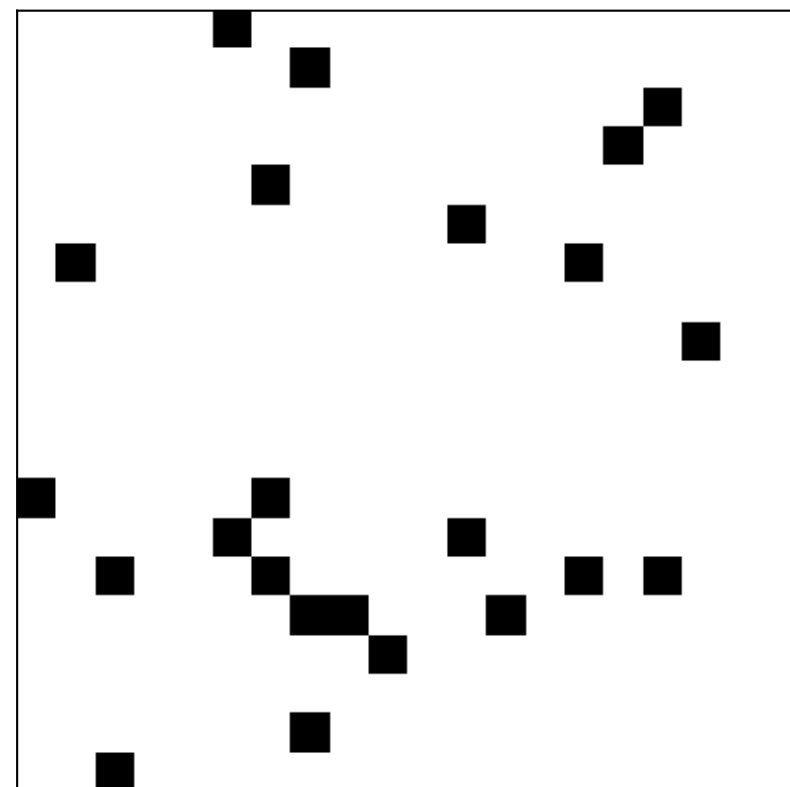
- $A_{ij}$  is the  $i, j$  *element*  
(also called *entry* or *coefficient*)
- $i$  is the row index,  $j$  the column index
- indices start at 1  
(when you code, at 0)
- vectors are like matrices with 1 column

# Special matrices

## Special matrices

- $A = 0$  (zero matrix):  $A_{ij} = 0$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, n$
- $A = I$  (identity matrix):  $m = n$  with  $A_{ii} = 1$  and  $A_{ij} = 0$  for  $i \neq j$

## Sparse matrices (most entries are 0)



- Examples:  $0$  and  $I$
- Can be stored and manipulated efficiently
- $\text{nnz}(A)$  is the number of nonzero entries

# Diagonal and triangular matrices

## diagonal matrix

- A square matrix  $n \times n$  with  $A_{ij} = 0$  when  $i \neq j$
- $\text{diag}(a_1, \dots, a_n)$  is the diagonal matrix with  $A_{ii} = a_i$  for  $i = 1, \dots, n$

$$\text{diag}(0.2, -3) = \begin{bmatrix} 0.2 & 0 \\ 0 & -3 \end{bmatrix}$$

## lower triangular matrix

$$A_{ij} = 0 \text{ for } i < j$$

$$\begin{bmatrix} -0.6 & 0 \\ 1.6 & -2 \end{bmatrix}$$

## upper triangular matrix

$$A_{ij} = 0 \text{ for } i > j$$

$$\begin{bmatrix} -0.2 & 0.3 \\ 0 & -1 \end{bmatrix}$$

# Block matrices

Matrices whose entries are matrices

$$n \times m \text{ matrix } A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$$

where  $B, C, D, E$  are  
*submatrices of blocks of  $A$*

**column representation**

$$A = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}$$

$(a_i \text{ are } m\text{-vectors})$

**row representation**

$$A = \begin{bmatrix} b_1^T \\ b_2^T \\ \vdots \\ b_n^T \end{bmatrix}$$

$(b_i \text{ are } n\text{-row-vectors})$

# Matrix operations

## transpose

A *transpose* of a matrix  $A$  is denoted as  $A^T$  where

$$(A^T)_{ij} = A_{ji}, \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

## addition (just like vectors)

$$(A + B)_{ij} = A_{ij} + B_{ij}, \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

## scalar multiplication (just like vectors)

$$(\alpha A)_{ij} = \alpha A_{ij}, \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

## Many properties

$$(A^T)^T = A, \quad A + B = B + A, \quad \alpha(A + B) = \alpha A + \alpha B$$



# Matrix-vector multiplication

## dot product

A matrix-vector product of an  $m \times n$  matrix  $A$  and a  $n$ -vector  $x$  is denoted as

$$y = Ax, \quad \text{where} \quad y_i = A_{i1}x_1 + \cdots + A_{in}x_n, \quad i = 1, \dots, m$$

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

## row interpretation

$$y_i = b_i^T x$$

where  $b_1^T, \dots, b_m^T$  are rows of  $A$

**example**  $A1$

## column interpretation

$$y = x_1 a_1 + \cdots x_n a_n$$

where  $a_1, \dots, a_n$  are the columns of  $A$

**example**  $Ae_j = a_j$

# Return matrix — portfolio vector

$R$  is the  $T \times n$  matrix of **asset returns**

|       | AAPL    | GOOG     | MMM      | AMZN     |             |
|-------|---------|----------|----------|----------|-------------|
| $R =$ | 0.00219 | 0.0006   | -0.00113 | 0.00202  | Mar 1, 2016 |
|       | 0.00744 | -0.00894 | -0.00019 | -0.00468 | Mar 2, 2016 |
|       | 0.01488 | -0.00215 | 0.00433  | -0.00407 | Mar 3, 2016 |

constant investment  $w$



$$R_{ti} = \frac{p_{ti}^{\text{final}} - p_{ti}^{\text{initial}}}{p_{ti}^{\text{initial}}}$$

$Rw$  is the vector of portfolio  
returns over periods  $1, \dots, T$

**example**  $w = (0.4, 0.3, -0.2, 0.5)$

$Rw = (0.00213, -0.00201, 0.00241)$  10

# Symmetric positive semidefinite matrices

**symmetric matrix**

$$A^T = A$$

**positive semidefinite matrix**

$$x^T A x \geq 0 \text{ for any } x \in \mathbf{R}^n$$

**positive definite matrix**

$$x^T A x > 0 \text{ for any } x \neq 0$$

**example**

$$A = \begin{bmatrix} 2 & 6 \\ 6 & 20 \end{bmatrix}$$



$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 6 \\ 6 & 20 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2x_1 + 6x_2 \\ 6x_1 + 20x_2 \end{bmatrix}$$

$$= 2x_1^2 + 12x_1x_2 + 20x_2^2$$

$$= 2(x_1 + 3x_2)^2 + 2x_2^2$$

**sum of squares**

# Matrix multiplication

## matrix product

A matrix product of an  $m \times p$  matrix  $A$  and a  $p \times n$  matrix  $B$  is

$$C = AB, \quad \text{where} \quad C_{ij} = A_{i1}B_{1j} + \cdots + A_{ip}B_{pj}, \quad i = 1, \dots, m, j = 1, \dots, n$$

(move along  $i$ th row of  $A$  and  $j$ th column of  $B$ )

$$\begin{bmatrix} -1.5 & 3 & 2 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 \\ 0 & -2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 3.5 & -4.5 \\ -1 & 1 \end{bmatrix}$$

# Complexity

Given  $m \times n$  matrix  $A$

- Matrix addition, scalar-matrix multiplication:  $mn$  flops
- Matrix-vector multiplication:  $m(2n - 1) \approx 2mn$  flops
- Matrix-matrix multiplication ( $A \in \mathbf{R}^{m \times p}$ ,  $B \in \mathbf{R}^{p \times n}$ ):  $(mn)(2p - 1) \approx 2mnp$  flops (inner product of  $p$  vectors)

## Questions

- How many flops does it take to multiply two  $1000 \times 1000$  matrices?
- How long does it take on a computer?

# Linear systems solutions

# Linear systems of equations

Given an  $m \times n$  matrix  $A$  and a  $m$ -vector  $b$ , find a  $n$ -vectors  $x$  such that

$$Ax = b$$

## typical scenarios

underdetermined  
(wide)

$$m < n$$

$$\begin{bmatrix} * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} * \\ * \end{bmatrix}$$

infinite  
solutions

square

$$m = n$$

$$\begin{bmatrix} * & * \\ * & * \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} * \\ * \end{bmatrix}$$

unique  
solution

**most common**

overdetermined  
(tall)

$$m > n$$

$$\begin{bmatrix} * & * \\ * & * \\ * & * \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} * \\ * \\ * \end{bmatrix}$$

no  
solution

# Solving square linear systems

Given an  $n \times n$  matrix  $A$  and a  $n$ -vector  $b$ , find a  $n$ -vector  $x$  such that

$$Ax = b \longrightarrow A^{-1}Ax = A^{-1}b \longrightarrow x = A^{-1}b$$

**When does it work?**

↓

$A$  must be invertible

↑  
inverse

- Columns of  $A$  are linearly independent
- Rows of  $A$  are linearly independent
- Columns/rows form a **basis** of  $\mathbf{R}^n$



# Solving linear systems

# How do we solve linear systems in practice?

## Idea

$$Ax = b$$

- compute  $A^{-1}$
- multiply  $A^{-1}b$

## Example

5000  $\times$  5000 matrix  $A$  and a 5000-vector  $b$

- Solve by computing  $A^{-1}$
- Solve with `numpy.linalg.solve`

**What's happening inside?**

# Easy linear systems

## Diagonal matrix

$$\begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \longrightarrow \begin{array}{l} A_{11}x_1 = b_1 \\ A_{22}x_2 = b_2 \\ \vdots \\ A_{nn}x_n = b_n \end{array}$$

## Solution

$$x = A^{-1}b = (b_1/A_{11}, \dots, b_n/A_{nn})$$

## Complexity

$n$  flops

# Easy linear systems

## Lower triangular matrix

$$\begin{bmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \vdots & & \ddots & \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \longrightarrow \begin{array}{l} A_{11}x_1 = b_1 \\ A_{21}x_1 + A_{22}x_2 = b_2 \\ \vdots \\ A_{n1}x_1 + A_{n2}x_2 + \dots + A_{nn}x_n = b_n \end{array}$$

### Solution: “forward substitution”

- First equation:  $x_1 = b_1/A_{11}$
- Second equation:  $x_2 = (b_2 - A_{21}x_1)/A_{22}$
- Repeat to get  $x_3, \dots, x_n$

### Complexity

- First equation: 1 flop (division)
- Second equation: 3 flops
- $i$ th step needs  $2i - 1$  flops

$$1 + 3 + \dots + (2n - 1) = n^2 \text{ flops}$$

# Easy linear systems

## Upper triangular matrix

$$\begin{bmatrix} A_{11} & \dots & A_{n-1,n} & A_{1n} \\ & \ddots & & \vdots \\ & & A_{n-1,n-1} & A_{n-1,n} \\ & & & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \longrightarrow \begin{array}{l} A_{11}x_1 + \dots + A_{1,n-1}x_{n-1} + A_{1n}x_n = b_1 \\ \vdots \\ A_{n-1,n-1}x_{n-1} + A_{n-1,n}x_n = b_{n-1} \\ A_{nn}x_n = b_n \end{array}$$

### Solution: “backward substitution”

- Last equation:  $x_n = b_n / A_{nn}$
- Second to last equation:  
 $x_{n-1} = (b_{n-1} - A_{n-1,n}x_n) / A_{n-1,n-1}$
- Repeat to get  $x_{n-2}, \dots, x_1$

### Complexity

- Last equation: 1 flop (division)
- Second to last equation: 3 flops
- $i$ th step needs  $2i - 1$  flops

$$1 + 3 + \dots + (2n - 1) = n^2 \text{ flops}_{21}$$

# Easy linear systems

## Permutation matrices

$\pi = (\pi_1, \dots, \pi_n)$  is a permutation of  $(1, 2, \dots, n)$

A  $n \times n$  permutation matrix  $P$ ,  
permutes the vector  $x$

$$Px = (x_{\pi_1}, \dots, x_{\pi_n})$$

## Properties

- $P_{ij} = \begin{cases} 1 & j = \pi_i \\ 0 & \text{otherwise} \end{cases}$
- $P^{-1} = P^T$  (inverse permutation)

## Complexity

Solve  $Px = b$ : 0 flops (no operations)

## example

$$\pi = (2, 3, 1)$$

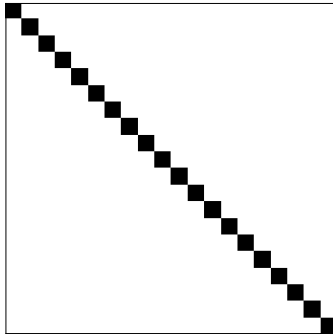
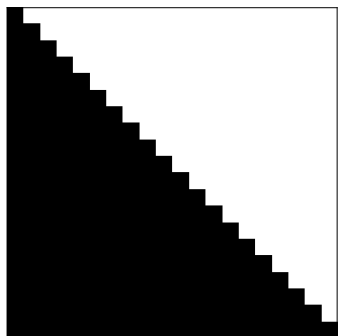
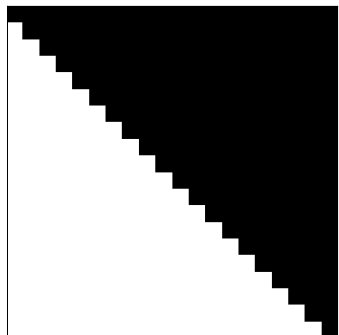


$$P \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ x_1 \end{bmatrix}$$



$$P^{-1} \begin{bmatrix} x_2 \\ x_3 \\ x_1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

# Summary of easy linear systems

|  |  | method                   | flops |
|--|--|--------------------------|-------|
|    | <b>diagonal</b><br>$A = \text{diag}(a_1, \dots, a_n)$    | $x_i = b_i / a_i$        | $n$   |
|   | <b>lower triangular</b><br>$A_{ij} = 0$ for $i < j$      | forward<br>substitution  | $n^2$ |
|  | <b>upper triangular</b><br>$A_{ij} = 0$ for $i > j$      | backward<br>substitution | $n^2$ |
|  | <b>permutation</b><br>$P_{ij} = 1$ if $j = \pi_i$ else 0 | inverse<br>permutation   | 0     |

# How do we solve linear systems in practice?

$$Ax = b$$

Any idea?

We know how to solve special ones

Let's use that!



# The factor-solve method for solving $Ax = b$

1. **Factor**  $A$  as a product of simple matrices:

$$A = A_1 A_2 \cdots A_k, \quad \longrightarrow \quad A_1 A_2 \cdots A_k x = b$$

( $A_i$  diagonal, upper/lower triangular, permutation, etc)

2. **Compute**  $x = A^{-1}b = A_k^{-1} \cdots A_1^{-1}b$   
by solving  $k$  “easy” systems



$$A_1 x_1 = b$$

$$A_2 x_2 = x_1$$

$$\vdots$$

$$A_k x = x_{k-1}$$

**Note:** step 2 is much cheaper than step 1

# Multiple right-hand sides

You now have factored  $A$  and you want to solve  $d$  linear systems with different right-hand side  $m$ -vectors  $b_i$

$$Ax = b_1 \quad Ax = b_2 \quad \dots \quad Ax = b_d$$

## Factorization-caching procedure

1. Factor  $A = A_1, \dots, A_k$  **only once** (expensive)
2. Solve all linear systems using **the same factorization** (cheap)

**Solve many “at the price of one”**

# *LU* Factorization

Every invertible matrix  $A$  can be factored as

$$A = PLU \quad \longrightarrow \quad P^T A = LU$$

$P$  permutation,  $L$  lower triangular,  $U$  upper triangular

## Procedure

- Similar to Gaussian elimination (but we can reuse  $P$ ,  $L$ , and  $U$ !)
- Permutation  $P$  avoids divisions by 0
- One of infinite possible combinations of  $P, L, U$

## Complexity

- $(2/3)n^3$  flops
- Less if  $A$  has special structure (sparse, diagonal, etc)

# *LU* Solution

$$Ax = b, \quad \Rightarrow \quad PLUx = b$$

## Iterations

1. *Permutation*: Solve  $Px_1 = b$  (0 flops)
2. *Forward substitution*: Solve  $Lx_2 = x_1$  ( $n^2$  flops)
3. *Backward substitution*: Solve  $Ux = x_2$  ( $n^2$  flops)

## Complexity

- Factor + solve:  $(2/3)n^3 + 2n^2 \approx (2/3)n^3$  (for large  $n$ )
- Just solve (prefactored):  $2n^2$

# $LL^T$ (Cholesky) Factorization

Every positive definite matrix  $A$  can be factored as

$$A = LL^T$$

$L$  lower triangular

## Procedure

- Works only on **symmetric with positive definite** matrices
- No need to permute as in  $LU$
- One of infinite possible choices of  $L$

## Complexity

- $(1/3)n^3$  flops (half of  $LU$  decomposition)
- Less if  $A$  has special structure (sparse, diagonal, etc)

# $LL^T$ (Cholesky) Solution

$$Ax = b, \quad \Rightarrow \quad LL^T x = b$$

## Iterations

1. *Forward substitution*: Solve  $Lx_1 = b$  ( $n^2$  flops)
2. *Backward substitution*: Solve  $L^T x = x_1$  ( $n^2$  flops)

## Complexity

- Factor + solve:  $(1/3)n^3 + 2n^2 \approx (1/3)n^3$  (for large  $n$ )
- Just solve (prefactored):  $2n^2$

# What complexity really means?

## Example

Large matrix  $n \times n$  matrix  $A$  with  $n = 10,000$

$$\begin{array}{l} \text{Factor + solve: } (2/3)n^3 \\ \text{Just solve: } 2n^2 \end{array} \longrightarrow \text{Gains: } \frac{(2/3)n^3}{2n^2} = (1/3)n \approx 3,333 \text{ times}$$

**3 thousand times!**

Something that takes 1 second  $\longrightarrow \approx 1$  hour

# Linear system example

## Polynomial interpolation

Given a cubic polynomial

$$p(x) = c_1 + c_2x + c_3x^2 + c_4x^3$$

Find the coefficients such  
that it passes by 4 points

$$p(-1.1) = b_1$$

$$p(-0.4) = b_2$$

$$p(0.1) = b_3$$

$$p(0.8) = b_4$$

Equivalent linear system

$$Ac = b$$

$$\begin{bmatrix} 1 & -1.1 & (-1.1)^2 & (-1.1)^3 \\ 1 & -0.4 & (-0.4)^2 & (-0.4)^3 \\ 1 & 0.1 & (0.1)^2 & (0.1)^3 \\ 1 & 0.8 & (0.8)^2 & (0.8)^3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

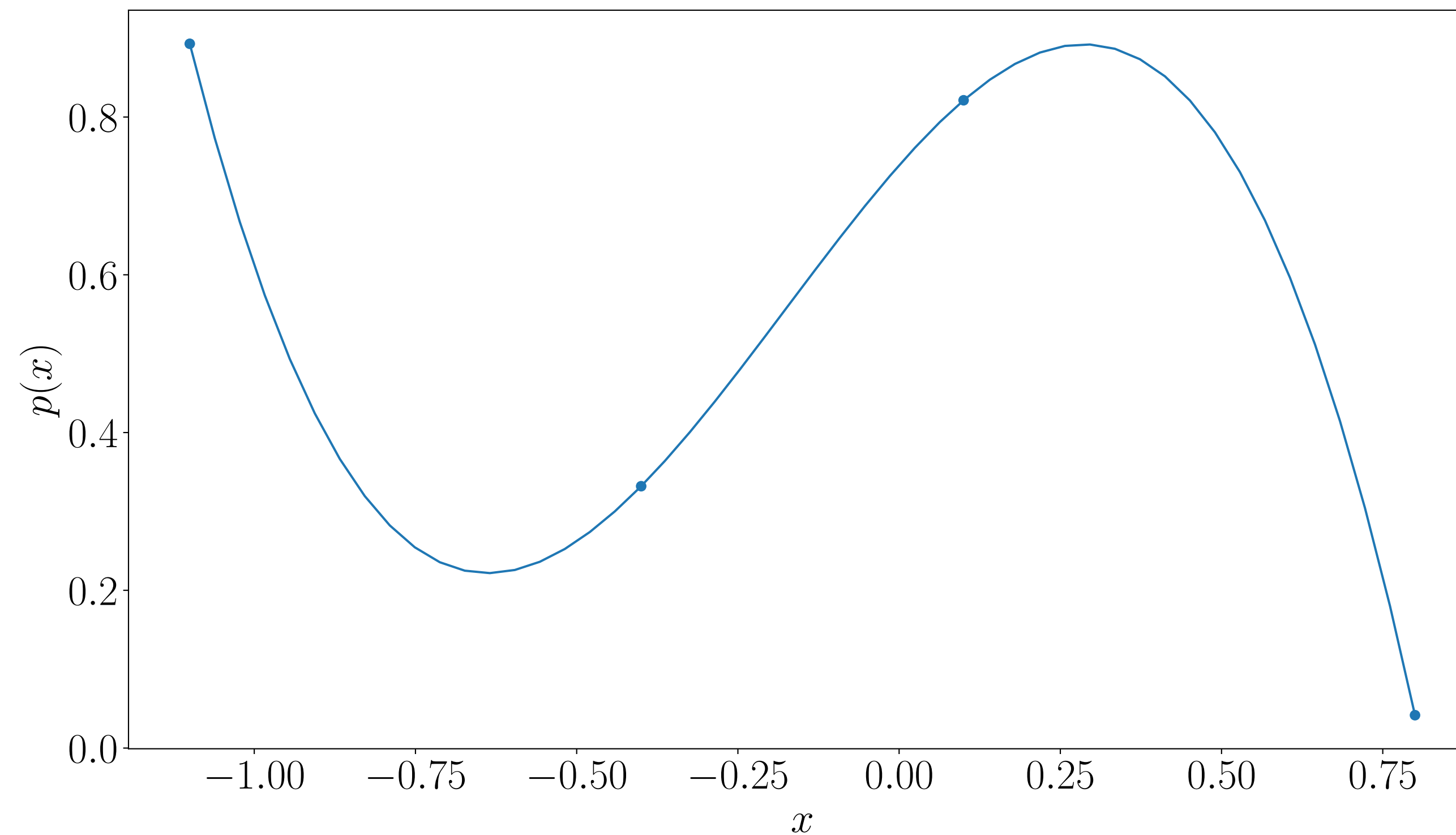


# Polynomial interpolation

## Plot

$$p(x) = c_1 + c_2x + c_3x^2 + c_4x^3$$

$$c = (0.74, 0.93, -0.89, -1.70)$$



# Solving linear systems in practice

Today, we learned to:

- **Avoid** computing inverses
- **Solve** linear systems using the factor-solve method
- **Understand** the complexity of solving linear systems  
(useful to build optimization algorithms!)

# References

- S. Boyd, L. Vandenberghe: Introduction to Applied Linear Algebra – Vectors, Matrices, and Least Squares
  - Chapter 6: matrices
  - Chapter 10: matrix multiplication
  - Chapter 11: matrix inverses/solving linear systems

# Next lecture

- Solve optimization problems: least squares