

# Introduction

- course topics
- basic course information

# Course topics

## Motivation

- modern computers are inexpensive, fast, have lots of memory
- it is easy to collect, store, transmit large amounts of data
- numerical software makes advanced algorithms simple to use

## Main topics

- numerical linear algebra (linear equations, least squares, . . . )
- nonlinear optimization, nonlinear least squares
- introduction to floating point numbers and rounding error
- examples from data analysis, statistics, image processing, control, . . .

# High-level languages for numerical computing

- MATLAB
- GNU Octave ([www.octave.org](http://www.octave.org))
- Python (via the libraries NumPy, SciPy, matplotlib, . . . )
- R ([www.r-project.org](http://www.r-project.org))
- Julia ([www.julialang.org](http://www.julialang.org))
- . . .

# Course information

## Course material

- textbook, lecture notes, homework assignments posted at  

ccle.ucla.edu
- homework solutions on CCLE course website

## Course requirements (see syllabus on CCLE website)

- weekly homework, most assignments include MATLAB exercises
- closed-book midterm exam  

Wednesday October 26, 2016 4PM
- closed-book final exam  

Wednesday December 7, 2016 11:30AM

# 1. Vectors

- notation
- examples
- vector operations
- linear functions
- complex vectors
- operation counts

# Vector

- a vector is an ordered, finite list of numbers
- we use two types of notation: vertical and horizontal arrays; for example

$$\begin{bmatrix} -1.1 \\ 0.0 \\ 3.6 \\ 7.2 \end{bmatrix} = (-1.1, 0.0, 3.6, 7.2)$$

- numbers in the list are the *elements* (*components, entries, coefficients*)
- number of elements is the *length* (*size, dimension*) of the vector
- a vector with length  $n$  is called an  $n$ -vector
- set of  $n$ -vectors with real elements is denoted  $\mathbf{R}^n$

# Conventions

- we *usually* denote vectors by lowercase letters

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = (a_1, a_2, \dots, a_n)$$

- $i$ th element of vector  $a$  is denoted  $a_i$
- $i$  is the *index* of the  $i$ th element  $a_i$

## Note

- several other conventions exist
- we'll make exceptions, *e.g.*,  $a_i$  can refer to  $i$ th vector in a list of vectors

## Block vectors, subvectors

vectors can be stacked (concatenated) to create larger vectors

### Example

- stacking vectors  $b, c, d$  of length  $m, n, p$  gives an  $(m + n + p)$ -vector

$$a = \begin{bmatrix} b \\ c \\ d \end{bmatrix} = (b_1, \dots, b_m, c_1, \dots, c_n, d_1, \dots, d_p)$$

- other notation:  $a = (b, c, d)$
- $b, c, d$  are *blocks* or *subvectors* of  $a$

# Special vectors

## Zero vector and vector of ones

$$\mathbf{0} = (0, 0, \dots, 0), \quad \mathbf{1} = (1, 1, \dots, 1)$$

length follows from context (if not, we add a subscript and write  $\mathbf{0}_n$ ,  $\mathbf{1}_n$ )

## Unit vectors

- there are  $n$  unit vectors of length  $n$ , written  $e_1, e_2, \dots, e_n$
- $i$ th unit vector is zero except its  $i$ th element which is 1; for  $n = 3$ ,

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

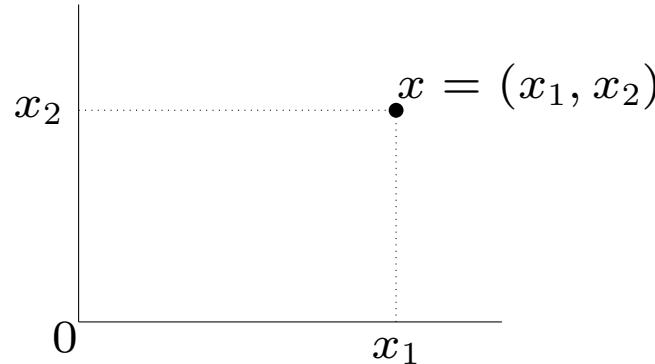
- length of  $e_i$  follows from context (or should be specified explicitly)

# Outline

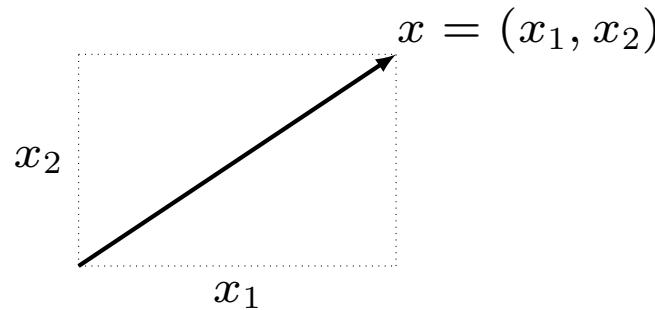
- notation
- **examples**
- vector operations
- linear functions
- complex vectors
- operation counts

# Geometry

**Position:** coordinates of a point in a plane or three-dimensional space



**Displacement:** shown as arrow in plane or 3-D space

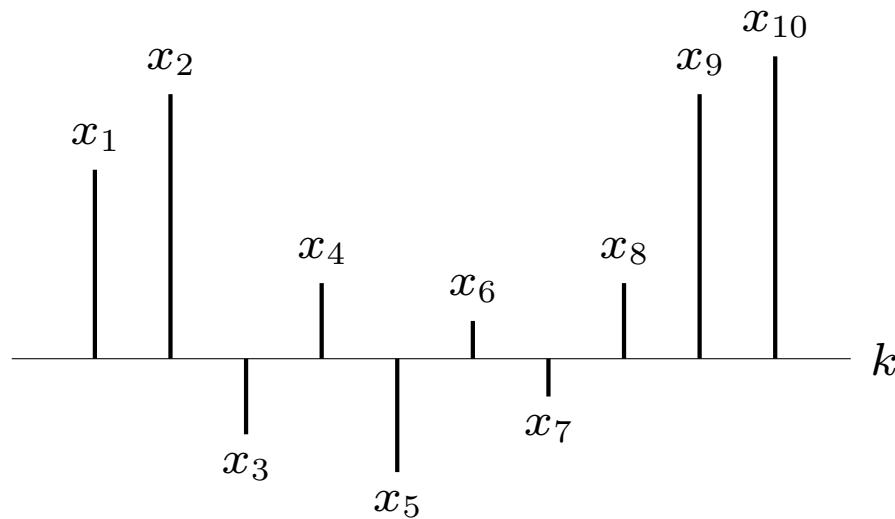


other quantities that have direction and magnitude, *e.g.*, force vector

# Signal or time series

values of some quantity at different (and finitely many) times

- $x_k$  is value at time  $k$ , or in period  $k$
- $n$ -vector  $x$  can be displayed by plotting  $x_k$  versus  $k$



- $x$  can represent a sampled function  $f(t)$  of a continuous time variable  $t$

$$x = (f(t_1), f(t_2), \dots, f(t_n))$$

# Images

## Monochrome (black-and-white) image

grayscale levels of  $M \times N$  pixels stored as  $MN$ -vector (*e.g.*, columnwise)

## Color image

- three  $MN$ -vectors with R, G, B intensities
- or concatenated as one vector of length  $3MN$

## Video sequence

- $K$  frames of size  $M \times N$  as  $K$  vectors or length  $MN$  (*if B&W*)
- or concatenated as one  $KMN$ -vector

# Feature vectors

contain values of variables or attributes that describe members of a set

## Examples

- age, weight, blood pressure, gender, . . . , of patients in a database
- square footage, #bedrooms, list price, . . . , of houses in an inventory

## Note

- vector elements represent very different quantities, in different units
- can contain categorical features (*e.g.*, 0/1 for male/female)
- ordering has no particular meaning

# Vectors of counts, histograms, occurrence vectors

## Word counts

- vector represents a document
- length of vector is number of words in a dictionary
- $i$ th element is number of times word  $i$  occurs in document

## Histogram

- vector represents a histogram or distribution
- $i$ th element is the frequency in bin  $i$

## Occurrence, set membership

- vector represents an object that can belong to  $n$  different sets
- $i$ th element is 1 if object is in set  $i$ ; zero otherwise

# Probability

- random event with  $n$  possible outcomes, numbered 1, 2, . . . ,  $n$
- probability vector  $p$  has length  $n$
- $i$ th element is probability of outcome  $i$
- elements of  $p$  are nonnegative and add up to one

# Economics and finance

## Portfolio

- vector represents portfolio of investments in  $n$  assets
- $i$ th element is amount invested in asset  $i$ , or #shares of asset  $i$  held

## Cash flow

- vector represents cash flow for  $n$  periods (e.g., quarters)
- $i$ th element represents payment to us (if positive), by us (if negative)

## Resource vector

- vector represents manufacturing process for a product
- process requires  $n$  resources (energy, labor, material, . . . )
- $i$ th element is amount of resource  $i$  used

# Polynomials and generalized polynomials

a polynomial of degree  $n - 1$  or less

$$f(t) = c_1 + c_2 t + c_3 t^2 + \cdots + c_n t^{n-1}$$

can be represented by an  $n$ -vector  $(c_1, c_2, \dots, c_n)$

**Extensions:** for example, a cosine polynomial

$$f(t) = c_1 + c_2 \cos t + c_3 \cos(2t) + \cdots + c_n \cos((n-1)t)$$

can be represented by an  $n$ -vector  $(c_1, c_2, \dots, c_n)$

# Summary

- vectors are used in a wide variety of applications
- can represent very different types of information
- usefulness of vector representation for a particular application depends on the relevance of vector *operations* for the application

# Outline

- notation
- examples
- **vector operations**
- linear functions
- complex vectors
- operation counts

# Addition and subtraction

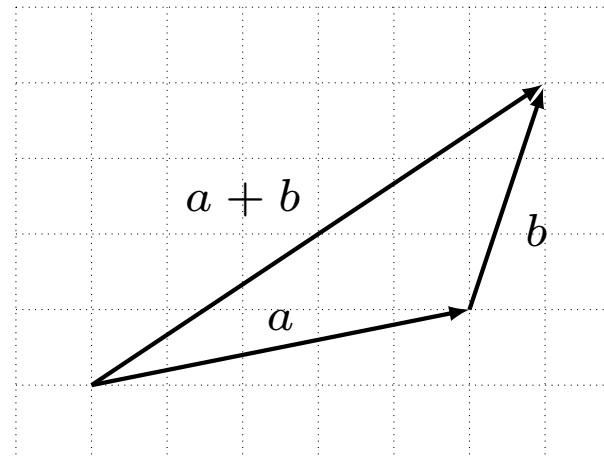
$$a + b = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_n + b_n \end{bmatrix}, \quad a - b = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \\ \vdots \\ a_n - b_n \end{bmatrix}$$

- commutative

$$a + b = b + a$$

- associative

$$a + (b + c) = (a + b) + c$$



## Scalar-vector and componentwise multiplication

**Scalar-vector multiplication:** for scalar  $\beta$  and  $n$ -vector  $a$ ,

$$\beta \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \beta a_1 \\ \beta a_2 \\ \vdots \\ \beta a_n \end{bmatrix}$$

**Componentwise multiplication:** for  $n$ -vectors  $a, b$

$$a \circ b = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{bmatrix}$$

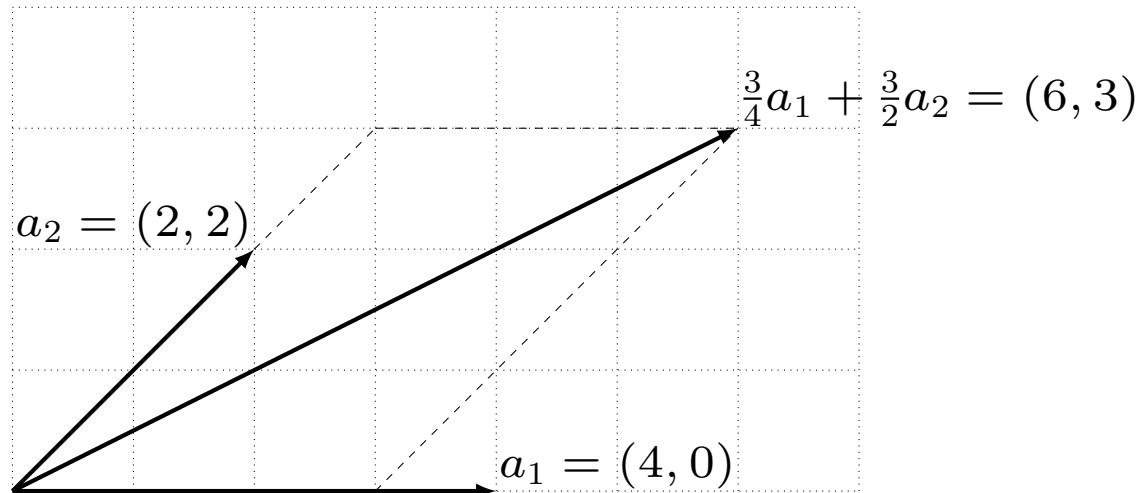
(in MATLAB: `a .* b`)

# Linear combination

a *linear combination* of vectors  $a_1, \dots, a_m$  is a sum of scalar products

$$\beta_1 a_1 + \beta_2 a_2 + \cdots + \beta_m a_m$$

the scalars  $\beta_1, \dots, \beta_m$  are the *coefficients* of the linear combination



## Inner product

the inner product of two  $n$ -vectors  $a, b$  is defined as

$$a^T b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

- a scalar
- meaning of superscript  $T$  will be explained when we discuss matrices
- other notation:  $\langle a, b \rangle$ ,  $(a | b)$ , . . .

(in MATLAB: `a' * b`)

# Properties

for vectors  $a, b, c$  of equal length, scalar  $\gamma$

- $a^T a = a_1^2 + a_2^2 + \cdots + a_n^2 \geq 0$

- $a^T a = 0$  only if  $a = 0$

- commutative:

$$a^T b = b^T a$$

- associative with scalar multiplication:

$$(\gamma a)^T b = \gamma(a^T b)$$

- distributive with vector addition:

$$(a + b)^T c = a^T c + b^T c$$

# Simple examples

**Inner product with unit vector**

$$e_i^T a = a_i$$

**Differencing**

$$(e_i - e_j)^T a = a_i - a_j$$

**Sum and average**

$$\mathbf{1}^T a = a_1 + a_2 + \cdots + a_n$$

$$\left(\frac{1}{n}\mathbf{1}\right)^T a = \frac{a_1 + a_2 + \cdots + a_n}{n}$$

# Applications

## Weighted sum

- $f$  is vector of features;  $w$  is vector of nonnegative weights
- $w^T f = w_1 f_1 + w_2 f_2 + \cdots + w_n f_n$  is total score

## Cost

- $p$  is vector of prices of  $n$  goods;  $q$  is vector of quantities purchased
- $p^T q = p_1 q_1 + p_2 q_2 + \cdots + p_n q_n$  is total cost

## Expected value

- $p$  is vector of probabilities of  $n$  outcomes
- $f_i$  is the value of a random variable in the event of outcome  $i$
- $p^T f$  is the expected value of the random variable

## Discounted value

- $c$  is a cash flow over  $n - 1$  periods;  $d$  is vector of discount factors

$$d = \left( 1, \frac{1}{1+r}, \frac{1}{(1+r)^2}, \dots, \frac{1}{(1+r)^{n-1}} \right)$$

$r > 0$  is interest rate

- $d^T c$  is net present value of cash flow

$$d^T c = c_1 + \frac{c_2}{1+r} + \frac{c_3}{(1+r)^2} + \dots + \frac{c_n}{(1+r)^{n-1}}$$

## Co-occurrence

- $a, b$  are 0-1 occurrence vectors; represent membership of A, B in  $n$  sets
- $a^T b$  is number of sets that contain both A and B

## Polynomial evaluation

- $c$  is vector of coefficients of  $f(t) = c_1 + c_2t + c_3t^2 + \cdots + c_nt^{n-1}$
- $x = (1, u, u^2, \dots, u^{n-1})$  is vector of powers of  $u$
- $c^T x$  is value of polynomial at  $u$

$$c^T x = c_1 + c_2u + c_3u^2 + \cdots + c_nu^{n-1} = f(u)$$

# Outline

- notation
- examples
- vector operations
- **linear functions**
- complex vectors
- operation counts

## Linear function

a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  is **linear** if superposition holds:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y) \quad (1)$$

for all  $n$ -vectors  $x, y$  and all scalars  $\alpha, \beta$

**Extension:** if  $f$  is linear, superposition holds for any linear combination:

$$f(\alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_m u_m) = \alpha_1 f(u_1) + \alpha_2 f(u_2) + \cdots + \alpha_m f(u_m)$$

for all scalars  $\alpha_1, \dots, \alpha_m$  and all  $n$ -vectors  $u_1, \dots, u_m$

(this follows by applying (1) repeatedly)

## Inner product function

for fixed  $a \in \mathbf{R}^n$ , define a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  as

$$f(x) = a^T x = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

- any function of this type is linear:

$$a^T(\alpha x + \beta y) = \alpha(a^T x) + \beta(a^T y)$$

for all scalars  $\alpha, \beta$  and all  $n$ -vectors  $x, y$

- every linear function can be written as an inner-product function:

$$\begin{aligned} f(x) &= f(x_1 e_1 + x_2 e_2 + \cdots + x_n e_n) \\ &= x_1 f(e_1) + x_2 f(e_2) + \cdots + x_n f(e_n) \end{aligned}$$

line 2 follows from superposition

## Examples in $\mathbf{R}^3$

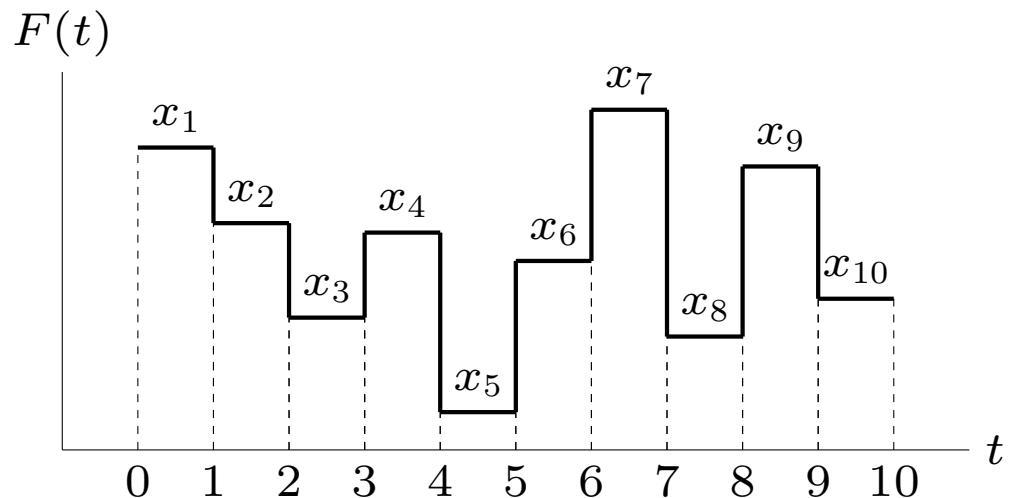
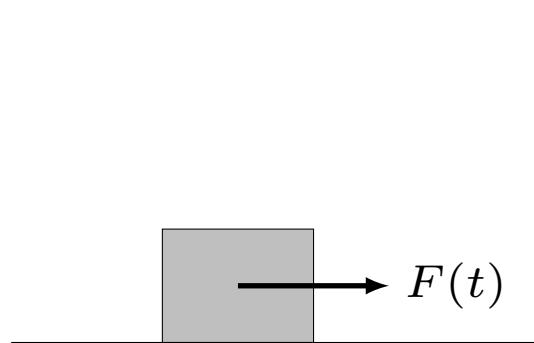
- $f(x) = \frac{1}{3}(x_1 + x_2 + x_3)$  is linear:  $f(x) = a^T x$  with  $a = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
- $f(x) = -x_1$  is linear:  $f(x) = a^T x$  with  $a = (-1, 0, 0)$
- $f(x) = \max\{x_1, x_2, x_3\}$  is not linear: superposition does not hold for

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad y = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \alpha = -1, \quad \beta = 1$$

we have  $f(x) = 1$ ,  $f(y) = 0$ ,

$$f(\alpha x + \beta y) = 0 \neq \alpha f(x) + \beta f(y) = -1$$

# Exercise



- unit mass with zero initial position and velocity
- subject to piecewise-constant force  $F(t)$  during interval  $[0, 10]$ :

$$F(t) = x_j \quad \text{for } t \in [j-1, j), \quad j = 1, \dots, 10$$

- define  $f(x)$  as position at  $t = 10$ ,  $g(x)$  as velocity at  $t = 10$

are  $f$  and  $g$  linear functions of  $x$ ?

## Solution

- from Newton's law  $s''(t) = F(t)$  where  $s(t)$  is the position at time  $t$
- integrate twice to get final velocity and position

$$\begin{aligned}s'(10) &= \int_0^{10} F(t) dt \\&= x_1 + x_2 + \cdots + x_{10} \\s(10) &= \int_0^{10} s'(t) dt \\&= \frac{19}{2}x_1 + \frac{17}{2}x_2 + \frac{15}{2}x_3 + \cdots + \frac{1}{2}x_{10}\end{aligned}$$

the two functions are linear:  $f(x) = a^T x$  and  $g(x) = b^T x$  with

$$a = \left(\frac{19}{2}, \frac{17}{2}, \dots, \frac{3}{2}, \frac{1}{2}\right), \quad b = (1, 1, \dots, 1)$$

## Affine function

a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  is **affine** if it satisfies

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

for all  $n$ -vectors  $x, y$  and all scalars  $\alpha, \beta$  with  $\alpha + \beta = 1$

**Extension:** if  $f$  is affine, then

$$f(\alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_m u_m) = \alpha_1 f(u_1) + \alpha_2 f(u_2) + \cdots + \alpha_m f(u_m)$$

for all  $n$ -vectors  $u_1, \dots, u_m$  and all scalars  $\alpha_1, \dots, \alpha_m$  with

$$\alpha_1 + \alpha_2 + \cdots + \alpha_m = 1$$

## Affine functions and inner products

for fixed  $a \in \mathbf{R}^n$ ,  $b \in \mathbf{R}$ , define a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  by

$$f(x) = a^T x + b = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + b$$

i.e., an inner-product function plus a constant (offset)

- any function of this type is affine: if  $\alpha + \beta = 1$  then

$$a^T(\alpha x + \beta y) + b = \alpha(a^T x + b) + \beta(a^T y + b)$$

- every affine function can be written as  $f(x) = a^T x + b$  with:

$$a = (f(e_1) - f(0), f(e_2) - f(0), \dots, f(e_n) - f(0)), \quad b = f(0)$$

## Affine approximation

first-order Taylor approximation of differentiable  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  around  $z$ :

$$\hat{f}(x) = f(z) + \frac{\partial f}{\partial x_1}(z)(x_1 - z_1) + \cdots + \frac{\partial f}{\partial x_n}(z)(x_n - z_n)$$

- generalizes first-order Taylor approximation of function of one variable

$$\hat{f}(x) = f(z) + f'(z)(x - z)$$

- $\hat{f}$  is a local affine approximation of  $f$  around  $z$
- in vector notation:  $\hat{f}(x) = f(z) + \nabla f(z)^T(x - z)$  where

$$\nabla f(z) = \left( \frac{\partial f}{\partial x_1}(z), \frac{\partial f}{\partial x_2}(z), \dots, \frac{\partial f}{\partial x_n}(z) \right)$$

the  $n$ -vector  $\nabla f(z)$  is called the *gradient* of  $f$  at  $z$

## Example

$$f(x_1, x_2) = x_1 - 3x_2 + e^{2x_1+x_2-1}$$

## Gradient

$$\nabla f(x) = \begin{bmatrix} 1 + 2e^{2x_1+x_2-1} \\ -3 + e^{2x_1+x_2-1} \end{bmatrix}$$

**First-order Taylor approximation** around  $z = 0$

$$\begin{aligned}\hat{f}(x) &= f(0) + \nabla f(0)^T(x - 0) \\ &= e^{-1} + (1 + 2e^{-1})x_1 + (-3 + e^{-1})x_2\end{aligned}$$

# Regression model

$$\begin{aligned}\hat{y} &= x^T \beta + \beta_0 \\ &= \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\end{aligned}$$

- $x$  is feature vector;  $x_i$ 's are *regressors*, *independent variables*, or *inputs*
- $\beta = (\beta_1, \dots, \beta_p)$  is vector of *weights*;  $\beta_0$  is *offset* or *intercept*
- $\hat{y}$  is (predicted) *outcome* or *dependent variable*
- regression model expresses  $\hat{y}$  as an affine function of  $x$
- model parameters are coefficients  $\beta_i$

## Example

- $\hat{y}$  is selling price of a house in some neighborhood
- regressors  $(x_1, x_2, x_3, x_4) = (\text{lot size}, \text{area}, \#\text{bedrooms}, \#\text{bathrooms})$

# Outline

- notation
- examples
- vector operations
- linear functions
- **complex vectors**
- operation counts

# Complex numbers

**Complex number:**  $x = \alpha + j\beta$  with  $\alpha, \beta$  real scalars

- $j = \sqrt{-1}$  (more common notation is  $i$  or  $j$ )
- $\alpha$  is the *real part* of  $x$ , denoted  $\operatorname{Re} x$
- $\beta$  is the *imaginary part*, denoted  $\operatorname{Im} x$

set of complex numbers is denoted **C**

## Modulus and conjugate

- modulus (absolute value, magnitude):  $|x| = \sqrt{(\operatorname{Re} x)^2 + (\operatorname{Im} x)^2}$
- conjugate:  $\bar{x} = \operatorname{Re} x - j \operatorname{Im} x$
- useful formulas:

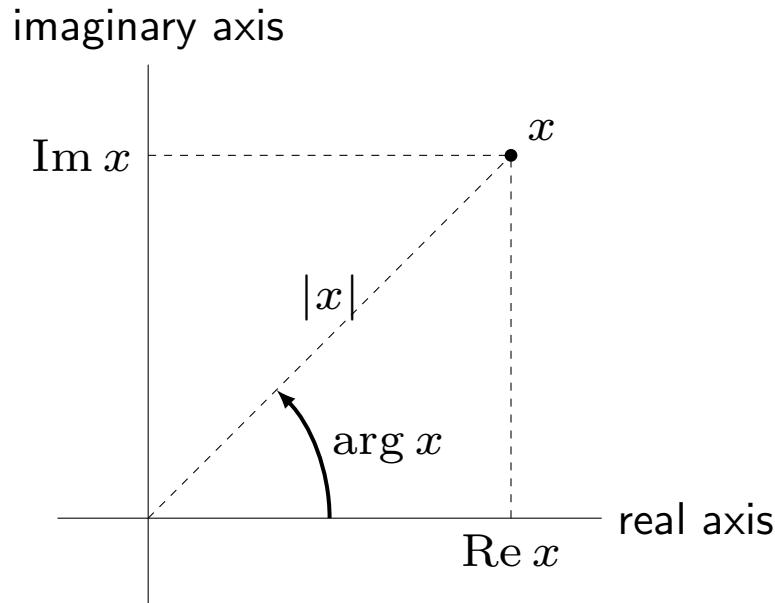
$$\operatorname{Re} x = \frac{x + \bar{x}}{2}, \quad \operatorname{Im} x = \frac{x - \bar{x}}{2j}, \quad |x|^2 = \bar{x}x$$

# Polar representation

nonzero complex  $x = \operatorname{Re} x + j \operatorname{Im} x$  can be written as

$$x = |x| (\cos \theta + j \sin \theta) = |x| e^{j\theta}$$

- $\theta \in [0, 2\pi)$  is the *argument (phase angle)* of  $x$  (notation:  $\arg x$ )
- $e^{j\theta}$  is complex exponential:  $e^{j\theta} = \cos \theta + j \sin \theta$



# Complex vector

- vector with complex elements:  $a = \alpha + j\beta$  with  $\alpha, \beta$  real vectors
- real and imaginary part, conjugate are defined componentwise:

$$\operatorname{Re} a = (\operatorname{Re} a_1, \operatorname{Re} a_2, \dots, \operatorname{Re} a_n)$$

$$\operatorname{Im} a = (\operatorname{Im} a_1, \operatorname{Im} a_2, \dots, \operatorname{Im} a_n)$$

$$\bar{a} = \operatorname{Re} a - j \operatorname{Im} a$$

- set of complex  $n$ -vectors is denoted  $\mathbf{C}^n$
- addition, scalar/componentwise multiplication defined as in  $\mathbf{R}^n$ :

$$a + b = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_n + b_n \end{bmatrix}, \quad \gamma a = \begin{bmatrix} \gamma a_1 \\ \gamma a_2 \\ \vdots \\ \gamma a_n \end{bmatrix}, \quad a \circ b = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{bmatrix}$$

## Complex inner product

the inner product of complex  $n$ -vectors  $a, b$  is defined as

$$b^H a = \bar{b}_1 a_1 + \bar{b}_2 a_2 + \cdots + \bar{b}_n a_n$$

- a complex scalar
- meaning of superscript  $H$  will be explained when we discuss matrices
- other notation:  $\langle a, b \rangle$ ,  $(a | b)$ , . . .
- for real vectors, reduces to real inner product  $b^T a$

(in MATLAB: `b' * a` )

# Properties

for complex  $n$ -vectors  $a, b, c$ , complex scalars  $\gamma$

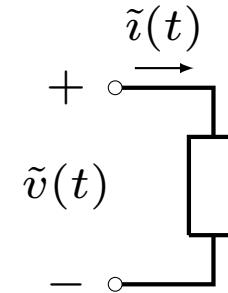
- $a^H a \geq 0$ : follows from

$$\begin{aligned} a^H a &= \bar{a}_1 a_1 + \bar{a}_2 a_2 + \cdots + \bar{a}_n a_n \\ &= |a_1|^2 + |a_2|^2 + \cdots + |a_n|^2 \end{aligned}$$

- $a^H a = 0$  only if  $a = 0$
- $b^H a = \overline{a^H b}$
- $b^H(\gamma a) = \gamma(b^H a)$
- $(\gamma b)^H a = \bar{\gamma}(b^H a)$
- $(b + c)^H a = b^H a + c^H a$
- $b^H(a + c) = b^H a + b^H c$

## Example: power in electric networks

- $\tilde{v}(t)$  is voltage across circuit element at time  $t$
- $\tilde{i}(t)$  is current through element



- $p(t) = \tilde{v}(t)\tilde{i}(t)$  is instantaneous power absorbed by element at time  $t$
- for  $n$  elements, with voltages  $\tilde{v}_k(t)$ ,  $\tilde{i}_k(t)$ ,  $k = 1, \dots, n$ , total power is

$$p(t) = \sum_{k=1}^n \tilde{v}_k(t)\tilde{i}_k(t),$$

the (real) inner product of two  $n$ -vectors of voltages and currents

# Sinusoidal voltage and current

- assume voltage and current are sinusoids with the same frequency

$$\tilde{v}(t) = V \cos(\omega t + \alpha), \quad \tilde{i}(t) = I \cos(\omega t + \beta) \quad (\text{with } V, I \geq 0)$$

- can be represented by complex numbers (phasors)

$$v = \frac{V}{\sqrt{2}} e^{j\alpha}, \quad i = \frac{I}{\sqrt{2}} e^{j\beta}$$

instantaneous power

$$\begin{aligned} p(t) &= VI \cos(\omega t + \alpha) \cos(\omega t + \beta) \\ &= \frac{VI}{2} (\cos(\alpha - \beta)(1 + \cos 2(\omega t + \alpha)) + \sin(\alpha - \beta) \sin 2(\omega t + \alpha)) \\ &= \operatorname{Re}(\bar{i}v) (1 + \cos 2(\omega t + \alpha)) + \operatorname{Im}(\bar{i}v) \sin 2(\omega t + \alpha) \end{aligned}$$

# Complex power

$$p(t) = \underbrace{\text{Re}(\bar{i}v) (1 + \cos 2(\omega t + \alpha))}_{\text{average } \text{Re}(\bar{i}v)} + \underbrace{\text{Im}(\bar{i}v) \sin 2(\omega t + \alpha)}_{\text{average zero}}$$

- $P = \text{Re}(\bar{i}v)$  is called *average* (or *real, active*) power
- $Q = \text{Im}(\bar{i}v)$  is *reactive power*
- $P + jQ = \bar{i}v$  is *complex power*

for  $n$  elements:  $n$ -vectors of phasors  $v = (v_1, \dots, v_n)$  and  $i = (i_1, \dots, i_n)$

$$i^H v = \text{Re}(i^H v) + j \text{Im}(i^H v)$$

- $\text{Re}(i^H v)$  is total average power
- $\text{Im}(i^H v)$  is sum of reactive powers
- $i^H v$  is sum of complex powers

# Outline

- notation
- examples
- vector operations
- linear functions
- complex vectors
- **operation counts**

# Floating-point operation

## Floating-point operation (flop)

- the unit of complexity when comparing vector and matrix algorithms
- 1 flop = one basic arithmetic operation (+, \*,  $\sqrt{\cdot}$ , . . . ) in **R** or **C**

**Comments:** this is a very simplified model of complexity of algorithms

- we don't distinguish between the different operations
- we don't distinguish between real and complex arithmetic
- we ignore integer operations (indexing, loop counters, . . . )
- we ignore cost of memory access

# Operation count

## Operation count (flop count)

- total number of operations in an algorithm
- in linear algebra, typically a polynomial of the dimensions in the problem
- a crude predictor of run time of the algorithm:

$$\text{run time} \approx \frac{\text{number of operations (flops)}}{\text{computer speed (flops per second)}}$$

**Dominant term:** the highest-order term in the flop count

$$\frac{1}{3}n^3 + 100n^2 + 10n + 5 \approx \frac{1}{3}n^3$$

**Order:** the power in the dominant term

$$\frac{1}{3}n^3 + 10n^2 + 100 = \text{order } n^3$$

## Examples

flop counts of vector operations in this lecture (for vectors of length  $n$ )

- addition, subtraction:  $n$  flops
- scalar multiplication:  $n$  flops
- componentwise multiplication:  $n$  flops
- inner product:  $2n - 1 \approx 2n$  flops

these operations are all order  $n$

## 2. Norm, distance, angle

- norm
- distance
- angle
- hyperplanes
- complex vectors

# Euclidean norm

(Euclidean) norm of vector  $a \in \mathbf{R}^n$ :

$$\begin{aligned}\|a\| &= \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} \\ &= \sqrt{a^T a}\end{aligned}$$

- if  $n = 1$ ,  $\|a\|$  reduces to absolute value  $|a|$
- measures the magnitude of  $a$
- sometimes written as  $\|a\|_2$  to distinguish from other norms, *e.g.*,

$$\|a\|_1 = |a_1| + |a_2| + \cdots + |a_n|$$

# Properties

## Nonnegative definiteness

$$\|a\| \geq 0 \quad \text{for all } a, \quad \|a\| = 0 \quad \text{only if } a = 0$$

## Homogeneity

$$\|\beta a\| = |\beta| \|a\| \quad \text{for all vectors } a \text{ and scalars } \beta$$

## Triangle inequality

$$\|a + b\| \leq \|a\| + \|b\| \quad \text{for all vectors } a \text{ and } b \text{ of equal length}$$

(proof on page 2-7)

## Cauchy-Schwarz inequality

$$|a^T b| \leq \|a\| \|b\| \quad \text{for all } a, b \in \mathbf{R}^n$$

moreover, equality  $|a^T b| = \|a\| \|b\|$  holds if:

- $a = 0$  or  $b = 0$ ; in this case  $a^T b = 0 = \|a\| \|b\|$
- $a \neq 0$  and  $b \neq 0$ , and  $b = \gamma a$  for some  $\gamma > 0$ ; in this case

$$0 < a^T b = \gamma \|a\|^2 = \|a\| \|b\|$$

- $a \neq 0$  and  $b \neq 0$ , and  $b = -\gamma a$  for some  $\gamma > 0$ ; in this case

$$0 > a^T b = -\gamma \|a\|^2 = -\|a\| \|b\|$$

# Proof of Cauchy-Schwarz inequality

1. trivial if  $a = 0$  or  $b = 0$
2. assume  $\|a\| = \|b\| = 1$ ; we show that  $-1 \leq a^T b \leq 1$

$$\begin{aligned} 0 &\leq \|a - b\|^2 & 0 &\leq \|a + b\|^2 \\ &= (a - b)^T(a - b) & &= (a + b)^T(a + b) \\ &= \|a\|^2 - 2a^T b + \|b\|^2 & &= \|a\|^2 + 2a^T b + \|b\|^2 \\ &= 2(1 - a^T b) & &= 2(1 + a^T b) \end{aligned}$$

with equality only if  $a = b$

with equality only if  $a = -b$

3. for general nonzero  $a, b$ , apply case 2 to the unit-norm vectors

$$\frac{1}{\|a\|}a, \quad \frac{1}{\|b\|}b$$

# RMS value

let  $a$  be a real  $n$ -vector

- the *average* of the elements of  $a$  is

$$\text{avg}(a) = \frac{a_1 + a_2 + \cdots + a_n}{n} = \frac{\mathbf{1}^T a}{n}$$

- the *root-mean-square* value is the root of the average squared entry

$$\text{rms}(a) = \sqrt{\frac{a_1^2 + a_2^2 + \cdots + a_n^2}{n}} = \frac{\|a\|}{\sqrt{n}}$$

## Exercise

- show that  $|\text{avg}(a)| \leq \text{rms}(a)$
- show that  $\text{avg}(b) \leq \text{rms}(a)$  where  $b = (|a_1|, |a_2|, \dots, |a_n|)$

## Triangle inequality from CS inequality

for vectors  $a, b$  of equal length

$$\begin{aligned}\|a + b\|^2 &= (a + b)^T(a + b) \\&= a^T a + b^T a + a^T b + b^T b \\&= \|a\|^2 + 2a^T b + \|b\|^2 \\&\leq \|a\|^2 + 2\|a\|\|b\| + \|b\|^2 \quad (\text{by Cauchy-Schwarz}) \\&= (\|a\| + \|b\|)^2\end{aligned}$$

- taking squareroots gives the triangle inequality
- triangle inequality is an equality if and only if  $a^T b = \|a\|\|b\|$  (see p. 2-4)
- also note from line 3 that  $\|a + b\|^2 = \|a\|^2 + \|b\|^2$  if  $a^T b = 0$

# Outline

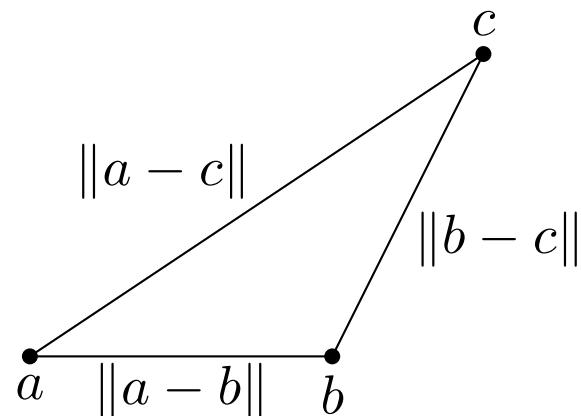
- norm
- **distance**
- angle
- hyperplanes
- complex vectors

# Distance

the (Euclidean) distance between vectors  $a$  and  $b$  is defined as  $\|a - b\|$

- $\|a - b\| \geq 0$  for all  $a, b$ ; distance is equal to zero only if  $a = b$
- triangle inequality

$$\|a - c\| \leq \|a - b\| + \|b - c\| \quad \text{for all } a, b, c$$



- RMS deviation between  $n$ -vectors  $a$  and  $b$  is  $\text{rms}(a - b) = \frac{\|a - b\|}{\sqrt{n}}$

# Standard deviation

let  $a$  be a real  $n$ -vector

- the *de-meaned* vector is the vector of deviations from the average

$$a - \text{avg}(a)\mathbf{1} = \begin{bmatrix} a_1 - \text{avg}(a) \\ a_2 - \text{avg}(a) \\ \vdots \\ a_n - \text{avg}(a) \end{bmatrix} = \begin{bmatrix} a_1 - (\mathbf{1}^T a)/n \\ a_2 - (\mathbf{1}^T a)/n \\ \vdots \\ a_n - (\mathbf{1}^T a)/n \end{bmatrix}$$

- the *standard deviation* is the RMS deviation from the average

$$\text{std}(a) = \text{rms}(a - \text{avg}(a)\mathbf{1}) = \frac{\|a - ((\mathbf{1}^T a)/n)\mathbf{1}\|}{\sqrt{n}}$$

- the de-meaned vector in *standard units* is

$$\frac{1}{\text{std}(a)}(a - \text{avg}(a)\mathbf{1})$$

# Exercise

show that

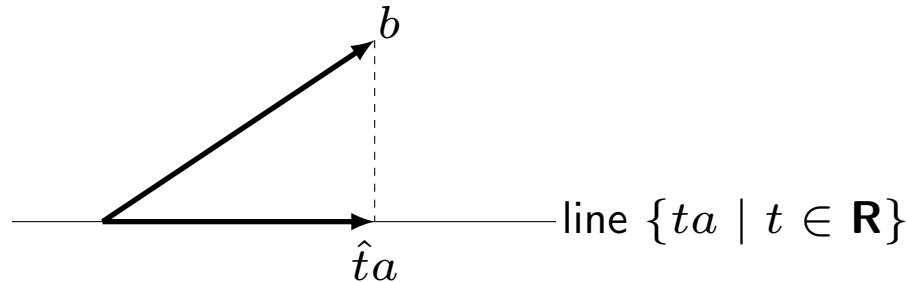
$$\text{avg}(a)^2 + \text{std}(a)^2 = \text{rms}(a)^2$$

## Solution

$$\begin{aligned}\text{std}(a)^2 &= \frac{\|a - \text{avg}(a)\mathbf{1}\|^2}{n} \\ &= \frac{1}{n} \left( a - \frac{\mathbf{1}^T a}{n} \mathbf{1} \right)^T \left( a - \frac{\mathbf{1}^T a}{n} \mathbf{1} \right) \\ &= \frac{1}{n} \left( a^T a - \frac{(\mathbf{1}^T a)^2}{n} - \frac{(\mathbf{1}^T a)^2}{n} + \left( \frac{\mathbf{1}^T a}{n} \right)^2 n \right) \\ &= \frac{1}{n} \left( a^T a - \frac{(\mathbf{1}^T a)^2}{n} \right) \\ &= \text{rms}(a)^2 - \text{avg}(a)^2\end{aligned}$$

## Exercise: nearest scalar multiple

given two vectors  $a, b \in \mathbf{R}^n$ , with  $a \neq 0$ , find scalar multiple  $ta$  closes to  $b$



### Solution

- squared distance between  $ta$  and  $b$  is

$$\|ta - b\|^2 = (ta - b)^T(ta - b) = t^2a^T a - 2ta^T b + b^T b$$

a quadratic function of  $t$  with positive leading coefficient  $a^T a$

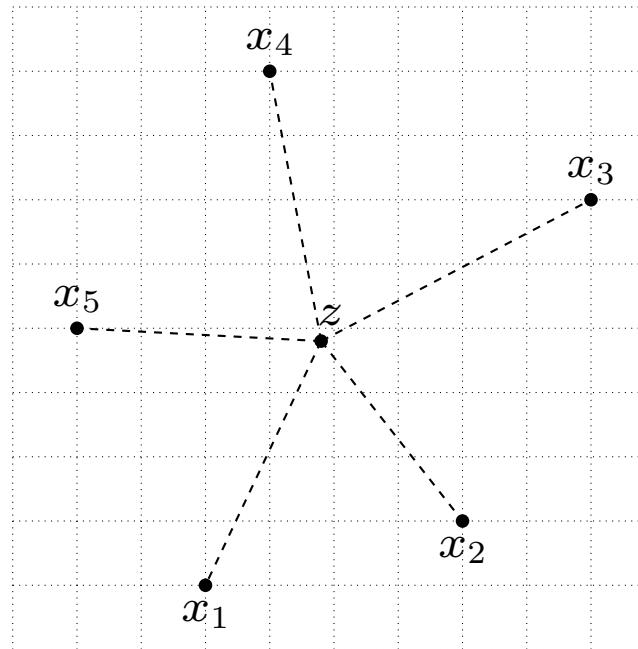
- derivative with respect to  $t$  is zero for

$$\hat{t} = \frac{a^T b}{a^T a} = \frac{a^T b}{\|a\|^2}$$

## Exercise: mean of set of points

given  $N$  vectors  $x_1, \dots, x_N \in \mathbf{R}^n$ , find the  $n$ -vector  $z$  that minimizes

$$\|z - x_1\|^2 + \|z - x_2\|^2 + \dots + \|z - x_N\|^2$$



$z$  is also known as the *centroid* of the points  $x_1, \dots, x_N$

**Solution:** sum of squared distances is

$$\begin{aligned} & \|z - x_1\|^2 + \|z - x_2\|^2 + \cdots + \|z - x_N\|^2 \\ &= \sum_{i=1}^n ((z_i - x_{1i})^2 + (z_i - x_{2i})^2 + \cdots + (z_i - x_{Ni})^2) \\ &= \sum_{i=1}^n (N z_i^2 - 2z_i(x_{1i} + x_{2i} + \cdots + x_{Ni}) + x_{1i}^2 + \cdots + x_{Ni}^2) \end{aligned}$$

(here  $x_{ji}$  is  $i$ th element of vector  $x_j$ )

- term  $i$  in the sum is minimized by

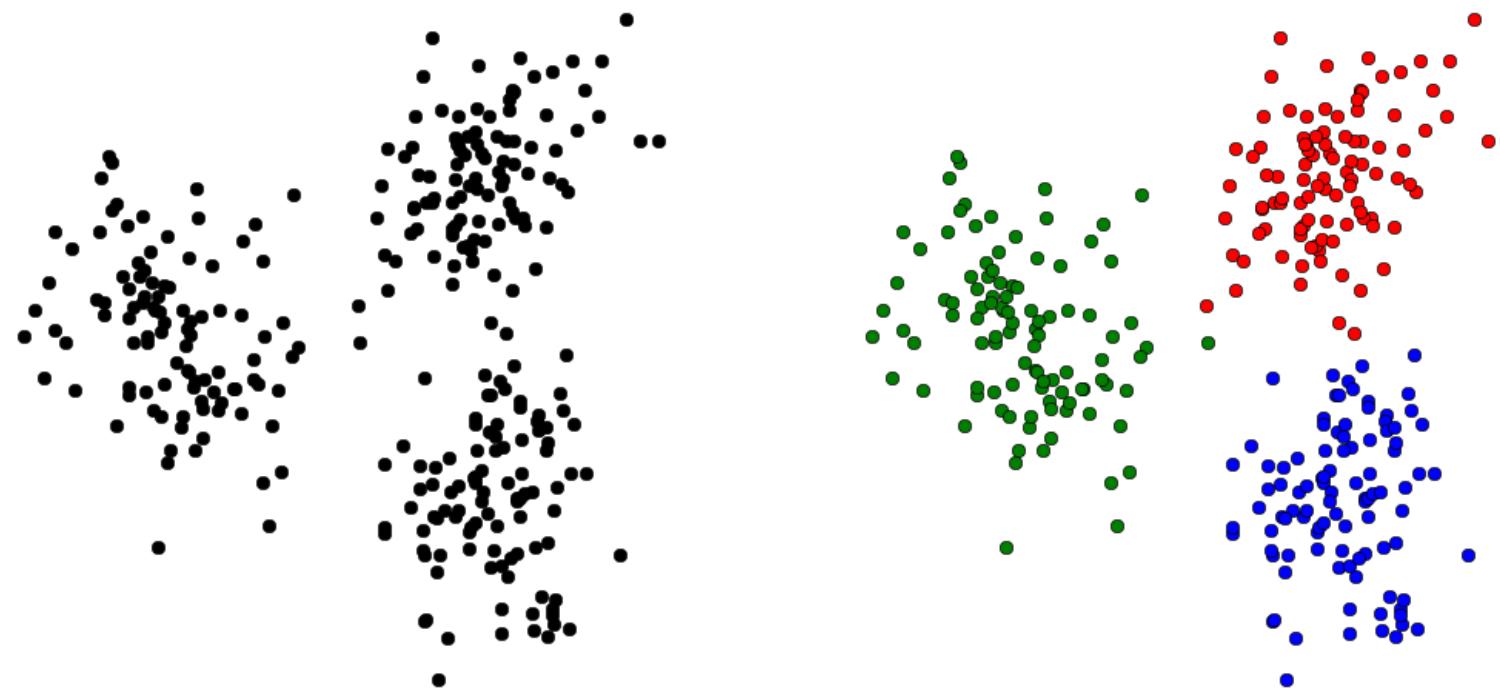
$$z_i = \frac{x_{1i} + x_{2i} + \cdots + x_{Ni}}{N}$$

- solution  $z$  is componentwise average of the points  $x_1, \dots, x_N$ :

$$z = \frac{1}{N} (x_1 + x_2 + \cdots + x_N)$$

## **$K$ -means clustering**

a very popular iterative algorithm for partitioning  $N$  vectors in  $K$  clusters

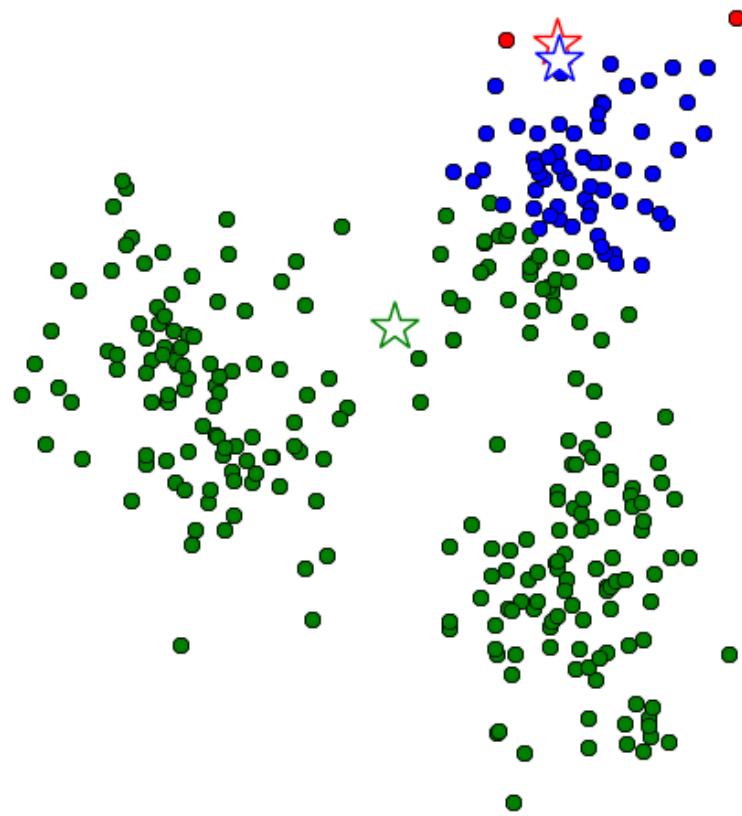


# Algorithm

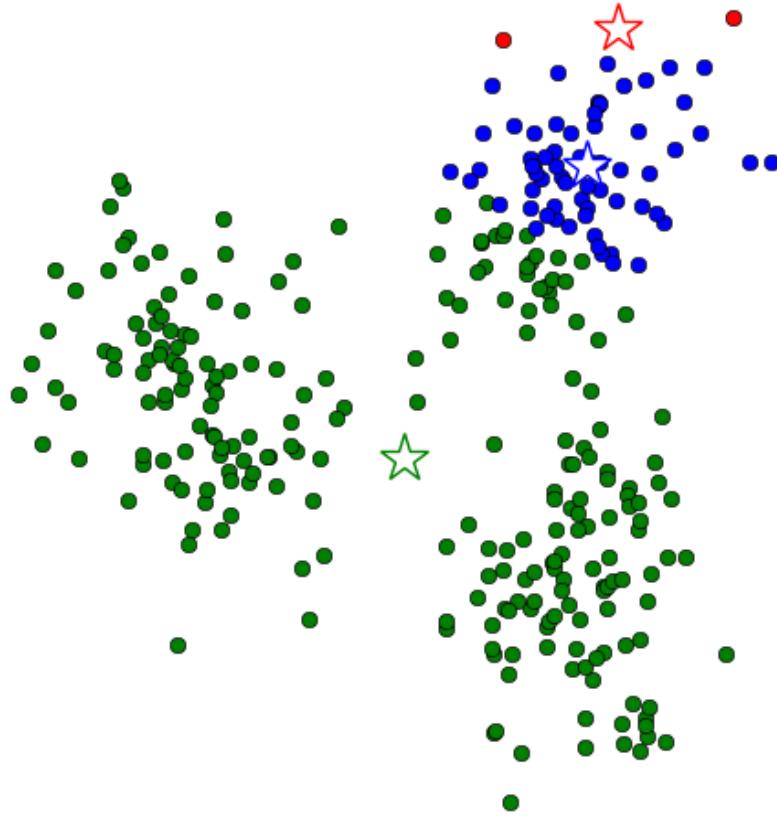
choose initial ‘representatives’  $z_1, \dots, z_K$  for the  $K$  clusters and repeat:

1. assign each vector  $x_i$  to the nearest representative  $z_j$
  2. replace each representative  $z_j$  by the mean of the vectors assigned to it
- 
- can be shown to converge in a finite number of iterations
  - initial representatives are often chosen randomly
  - as a variation, choose a random initial partition and start with step 2
  - solution depends on choice of initial representatives
  - in practice, often restarted a few times, with different starting points

## Example: first iteration

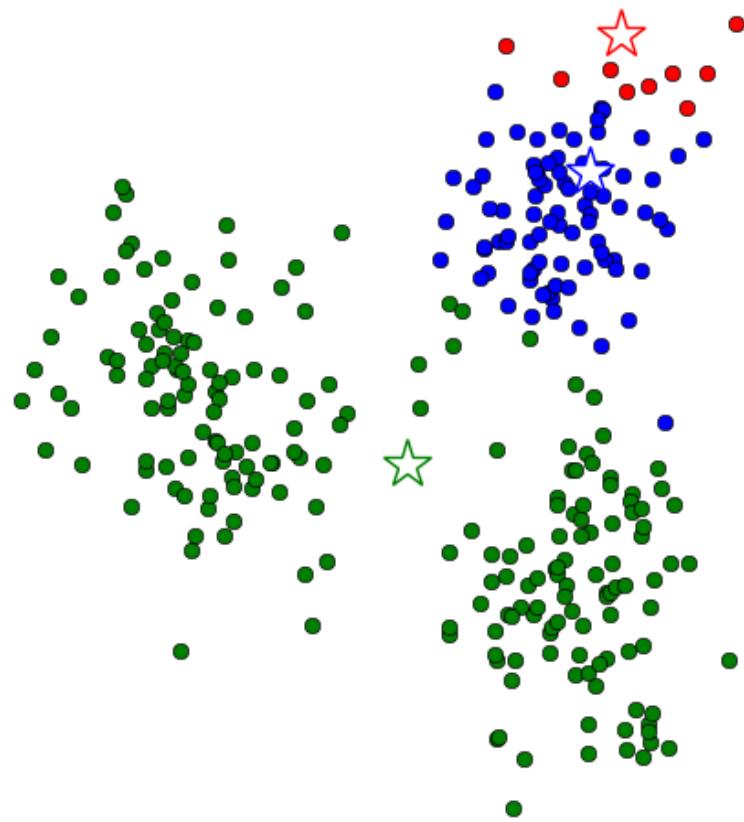


assignment to clusters

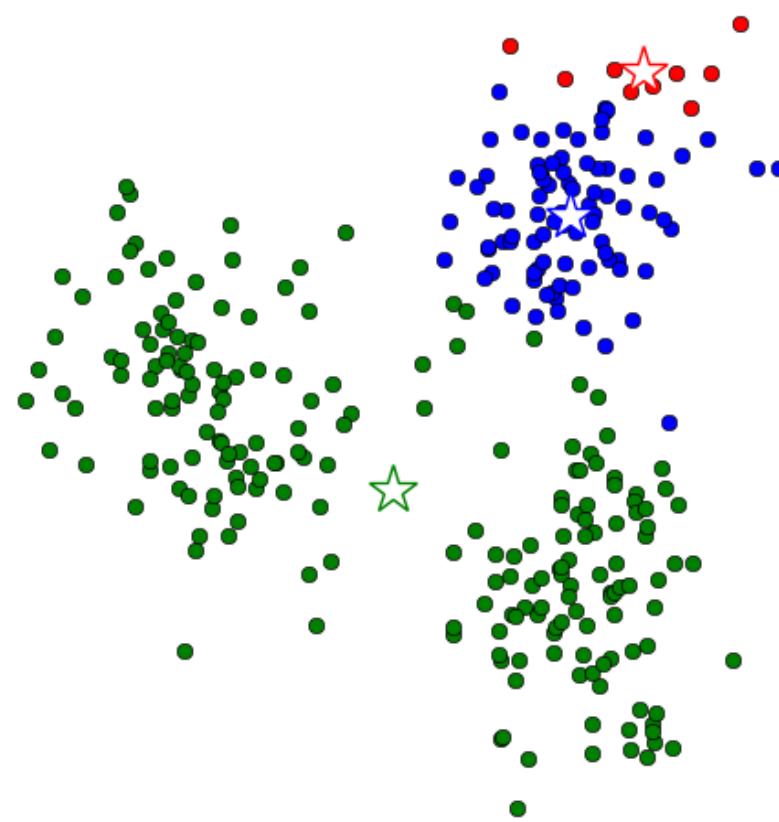


updated representatives

## Example: iteration 2

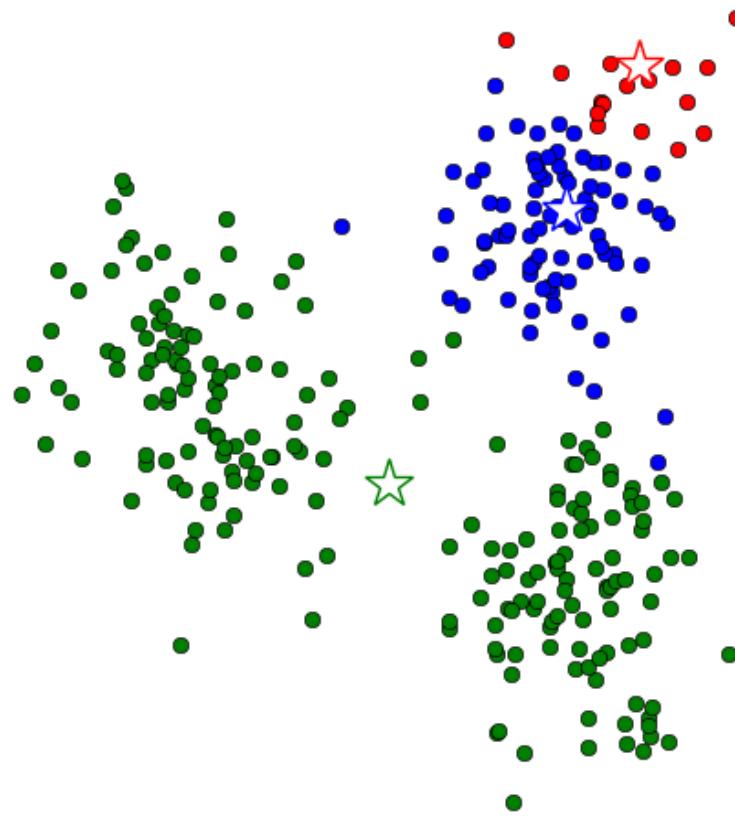


assignment to clusters

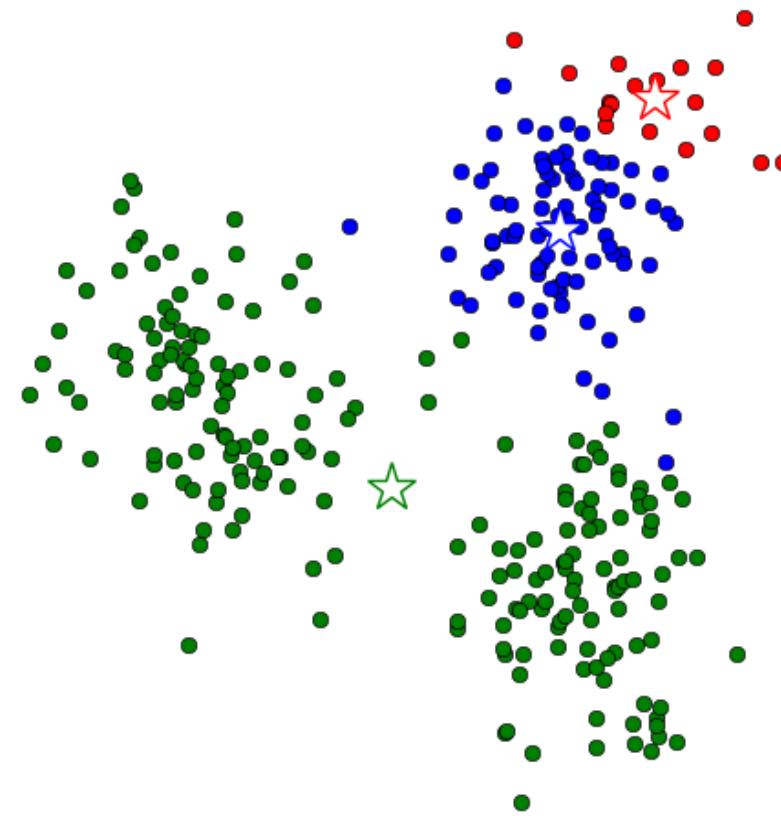


updated representatives

## Example: iteration 3

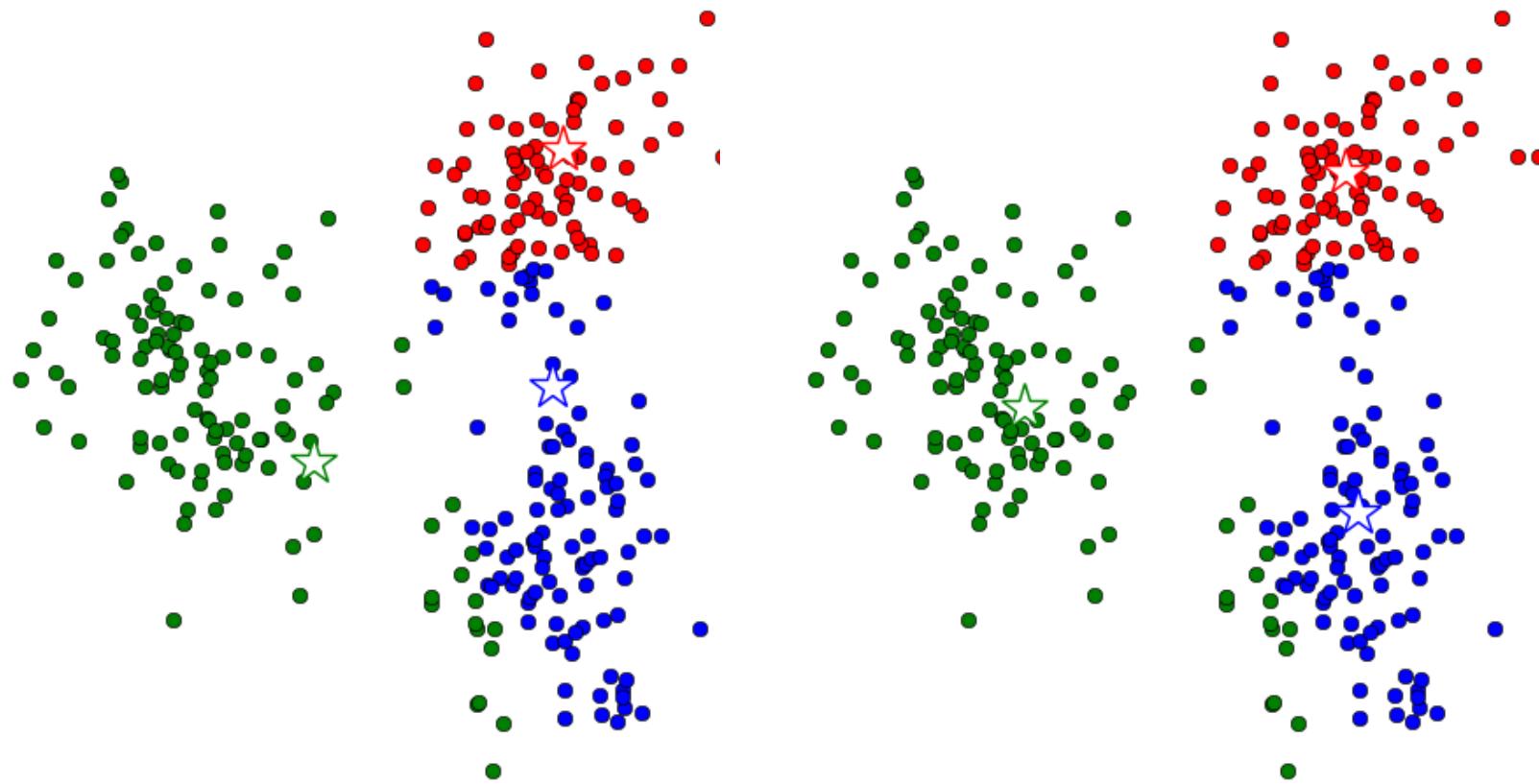


assignment to clusters



updated representatives

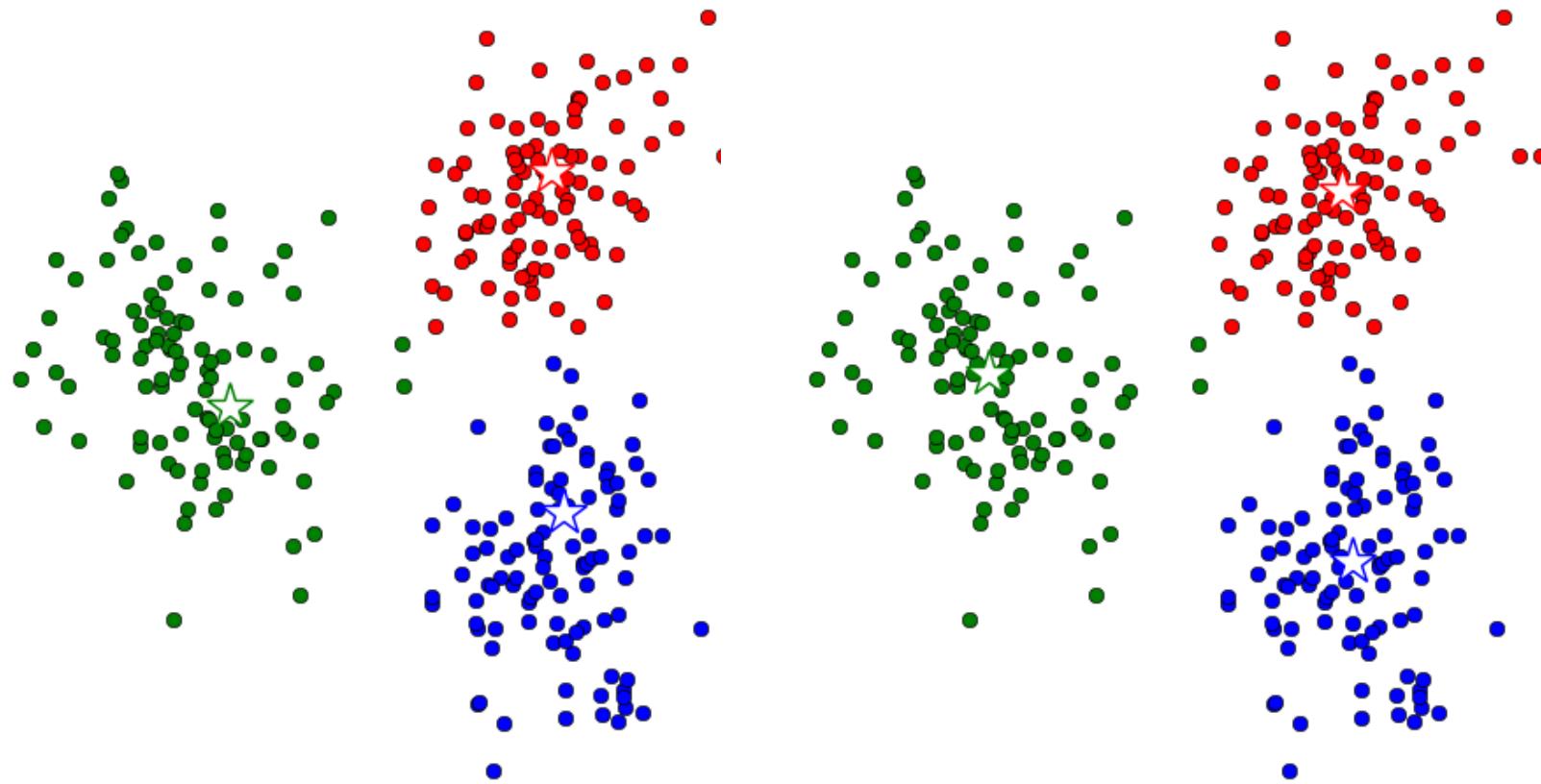
## Example: iteration 9



assignment to clusters

updated representatives

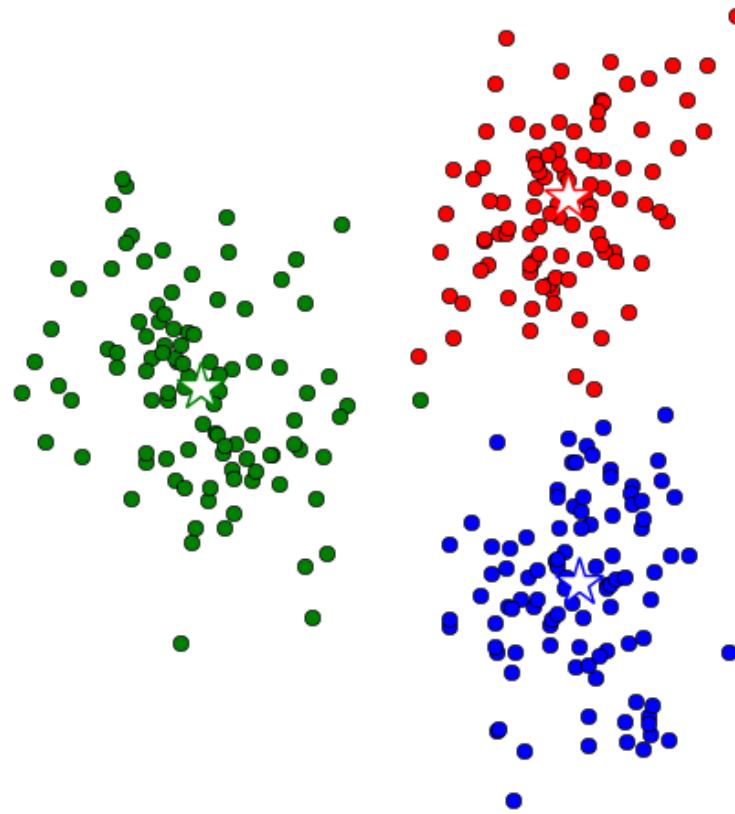
## Example: iteration 10



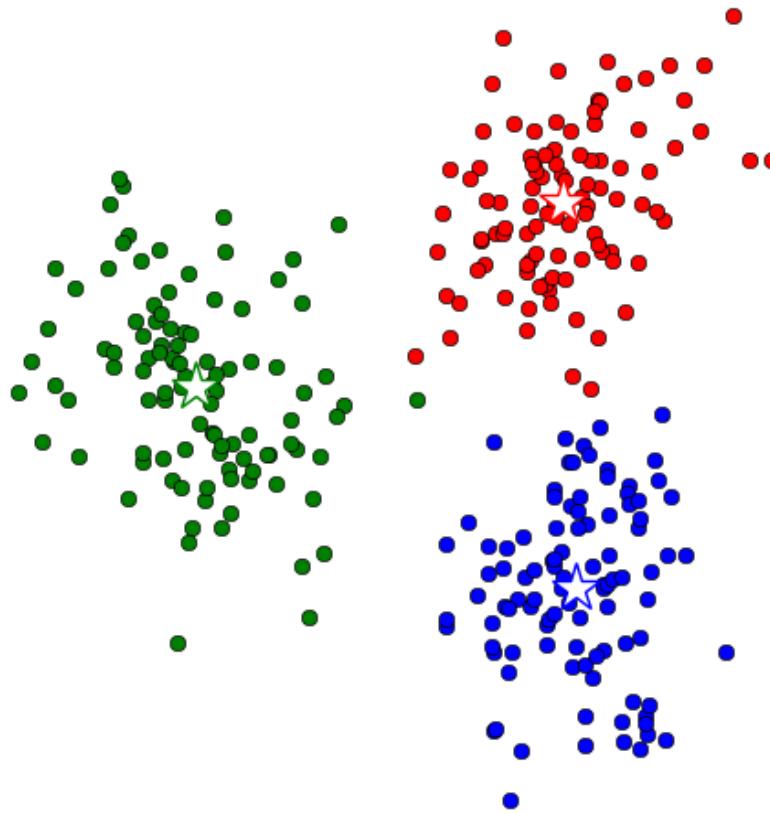
assignment to clusters

updated representatives

## Example: iteration 11

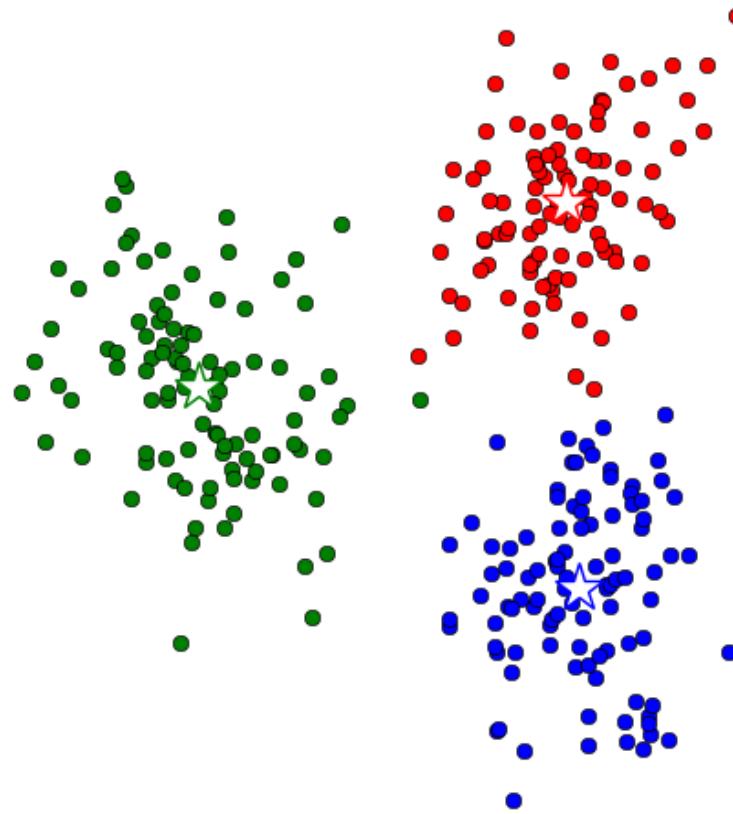


assignment to clusters

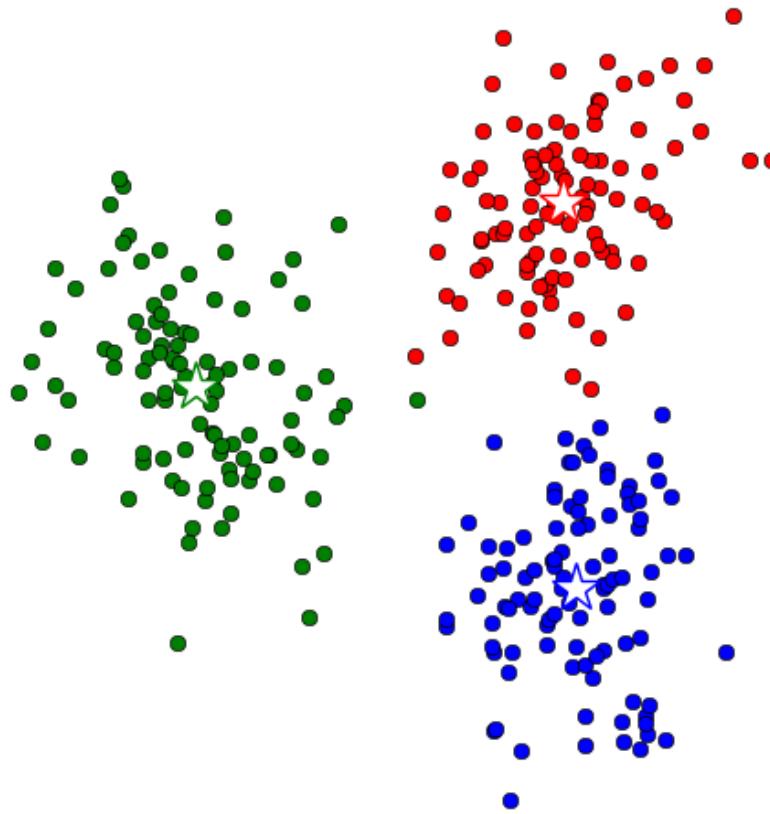


updated representatives

## Example: iteration 12



assignment to clusters



updated representatives

# Outline

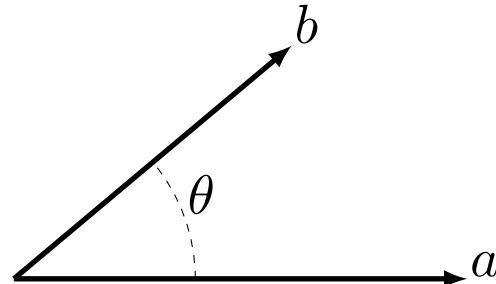
- norm
- distance
- **angle**
- hyperplanes
- complex vectors

# Angle between vectors

the angle between nonzero real vectors  $a, b$  is defined as

$$\arccos \left( \frac{a^T b}{\|a\| \|b\|} \right)$$

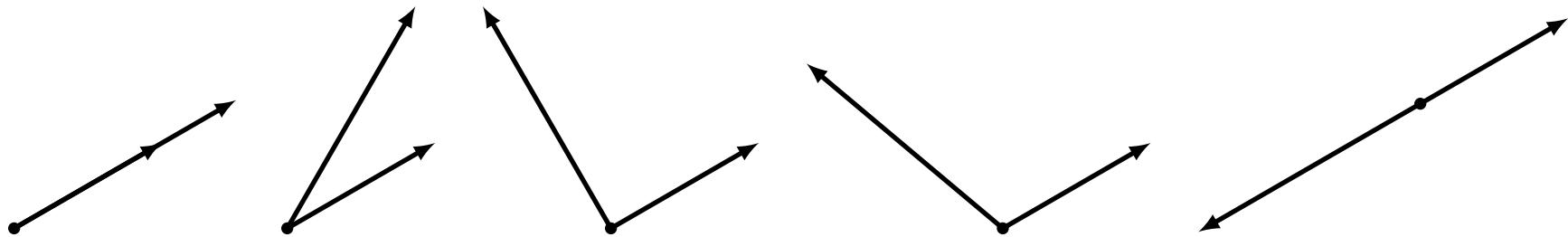
- this is the unique value of  $\theta \in [0, \pi]$  that satisfies  $a^T b = \|a\| \|b\| \cos \theta$



- Cauchy-Schwarz inequality guarantees that

$$-1 \leq \frac{a^T b}{\|a\| \|b\|} \leq 1$$

# Terminology

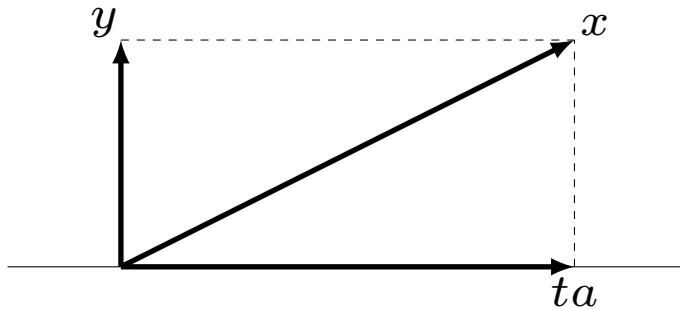


|                           |                        |  |
|---------------------------|------------------------|--|
| $\theta = 0$              | $a^T b = \ a\  \ b\ $  | vectors are aligned or parallel        |
| $0 \leq \theta < \pi/2$   | $a^T b > 0$            | vectors make an acute angle            |
| $\theta = \pi/2$          | $a^T b = 0$            | vectors are orthogonal ( $a \perp b$ ) |
| $\pi/2 < \theta \leq \pi$ | $a^T b < 0$            | vectors make an obtuse angle           |
| $\theta = \pi$            | $a^T b = -\ a\  \ b\ $ | vectors are anti-aligned or opposed    |

# Orthogonal decomposition

given a nonzero  $a \in \mathbf{R}^n$ , every  $n$ -vector  $x$  can be decomposed as

$$x = ta + y \quad \text{with } y \perp a$$



$$t = \frac{a^T x}{\|a\|^2}, \quad y = x - \frac{a^T x}{\|a\|^2} a$$

- proof is by inspection
- decomposition (*i.e.*,  $t$  and  $y$ ) exists and is unique for every  $x$
- $ta$  is projection of  $x$  on the line through  $a$  and the origin (see page 2-11)
- since  $y \perp a$ , we have  $\|x\|^2 = \|ta\|^2 + \|y\|^2$

## Correlation coefficient

the *correlation coefficient* between non-constant vectors  $a, b$  is

$$\rho_{ab} = \frac{\tilde{a}^T \tilde{b}}{\|\tilde{a}\| \|\tilde{b}\|}$$

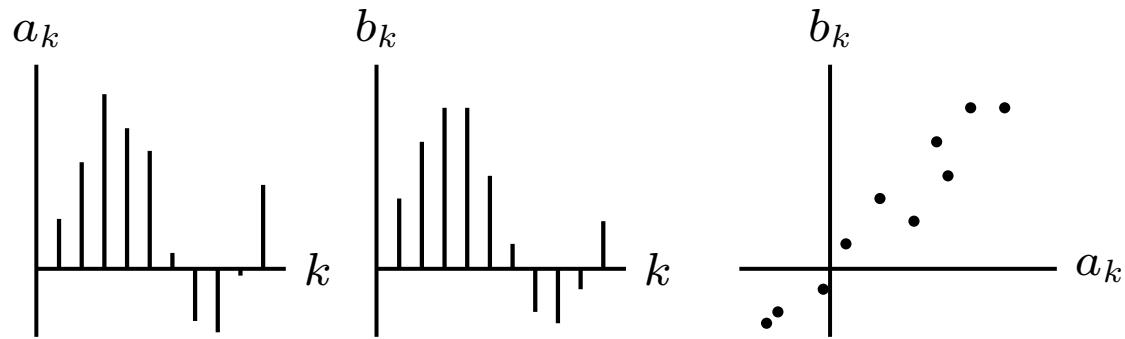
where  $\tilde{a} = a - \text{avg}(a)\mathbf{1}$  and  $\tilde{b} = b - \text{avg}(b)\mathbf{1}$  are the de-meanned vectors

- only defined when  $a$  and  $b$  are not constant ( $\tilde{a} \neq 0$  and  $\tilde{b} \neq 0$ )
- $\rho_{ab}$  is the cosine of the angle between the de-meanned vectors
- $\rho_{ab}$  is the average product of deviations from the mean in standard units

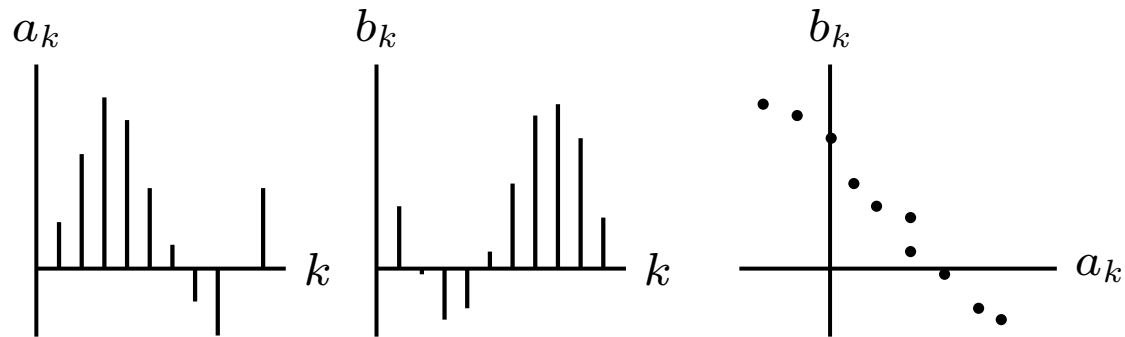
$$\rho_{ab} = \frac{1}{n} \sum_{i=1}^n \frac{(a_i - \text{avg}(a))}{\text{std}(a)} \frac{(b_i - \text{avg}(b))}{\text{std}(b)}$$

# Examples

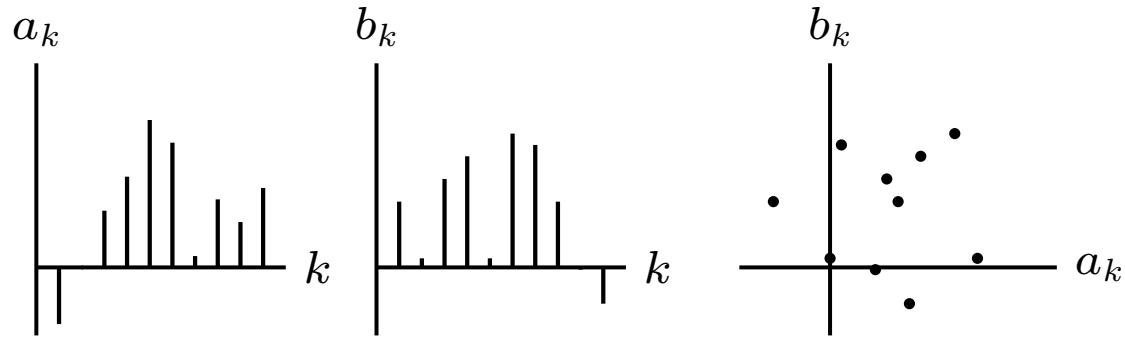
$$\rho_{ab} = 0.97$$



$$\rho_{ab} = -0.99$$



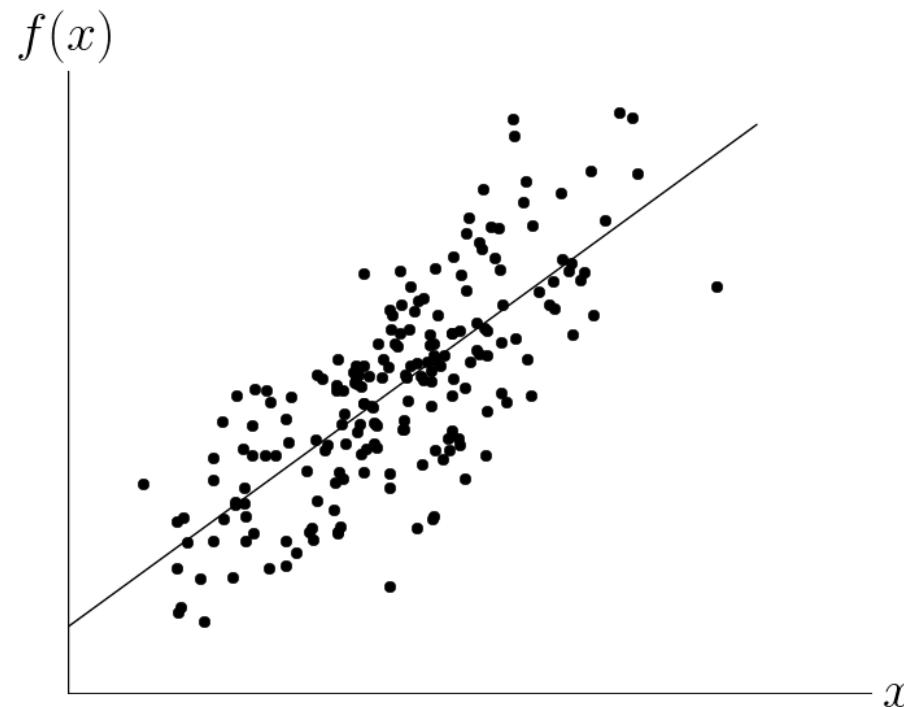
$$\rho_{ab} = 0.004$$



# Regression line

- scatterplot shows two  $n$ -vectors  $a, b$  as  $n$  points  $(a_k, b_k)$
- straight line shows affine function  $f(x) = c_1 + c_2x$  with

$$f(a_k) \approx b_k, \quad k = 1, \dots, n$$



## Least-squares regression

use coefficients  $c_1, c_2$  that minimize  $J = \frac{1}{n} \sum_{k=1}^n (f(a_k) - b_k)^2$

- $J$  is a quadratic function of  $c_1$  and  $c_2$ :

$$\begin{aligned} J &= \frac{1}{n} \sum_{k=1}^n (c_1 + c_2 a_k - b_k)^2 \\ &= (nc_1^2 + 2n \text{avg}(a)c_1 c_2 + \|a\|^2 c_2^2 - 2n \text{avg}(b)c_1 - 2a^T b c_2 + \|b\|^2) / n \end{aligned}$$

- to minimize  $J$ , set derivatives with respect to  $c_1, c_2$  to zero:

$$c_1 + \text{avg}(a)c_2 = \text{avg}(b), \quad n \text{avg}(a)c_1 + \|a\|^2 c_2 = a^T b$$

- solution is

$$c_2 = \frac{a^T b - n \text{avg}(a) \text{avg}(b)}{\|a\|^2 - n \text{avg}(a)^2}, \quad c_1 = \text{avg}(b) - \text{avg}(a)c_2$$

# Interpretation

slope  $c_2$  can be written in terms of correlation coefficient of  $a$  and  $b$ :

$$c_2 = \frac{(a - \text{avg}(a)\mathbf{1})^T(b - \text{avg}(b)\mathbf{1})}{\|a - \text{avg}(a)\mathbf{1}\|^2} = \rho_{ab} \frac{\text{std}(b)}{\text{std}(a)}$$

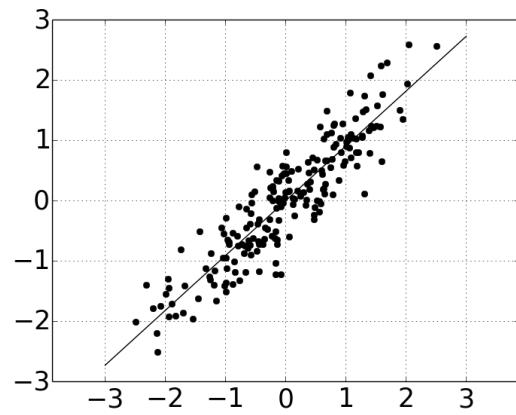
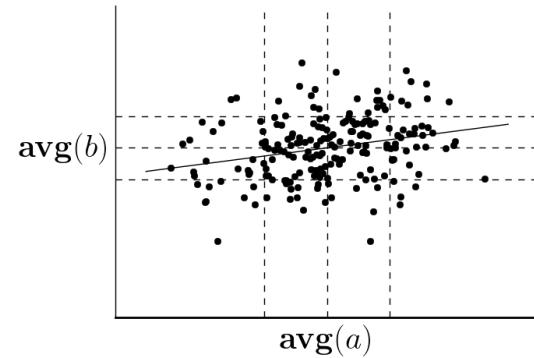
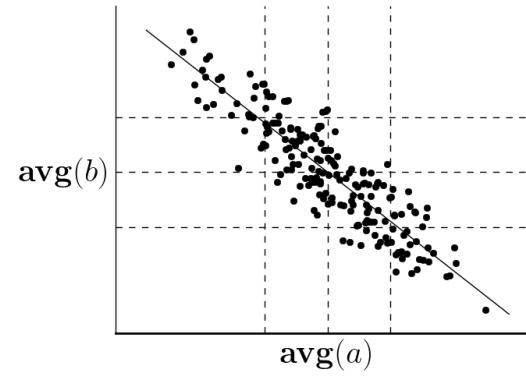
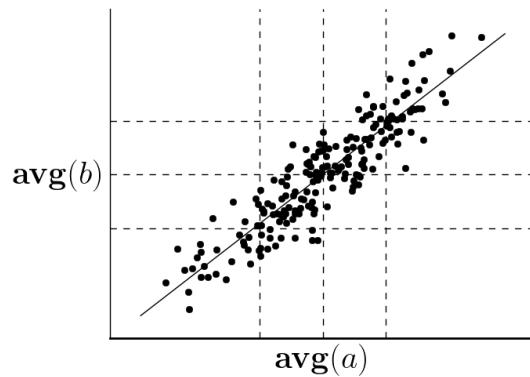
- hence, expression for regression line can be written as

$$f(x) = \text{avg}(b) + \frac{\rho_{ab} \text{std}(b)}{\text{std}(a)}(x - \text{avg}(a))$$

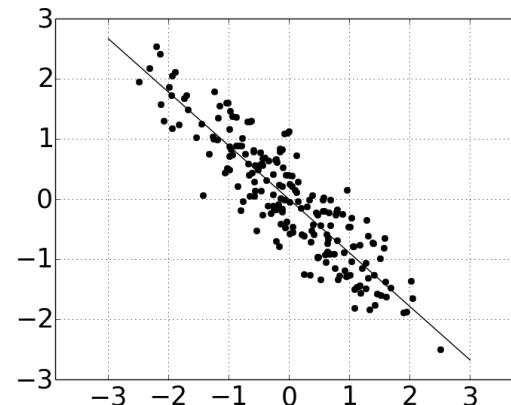
- correlation coefficient  $\rho_{ab}$  is the slope after converting to standard units:

$$\frac{f(x) - \text{avg}(b)}{\text{std}(b)} = \rho_{ab} \frac{x - \text{avg}(a)}{\text{std}(a)}$$

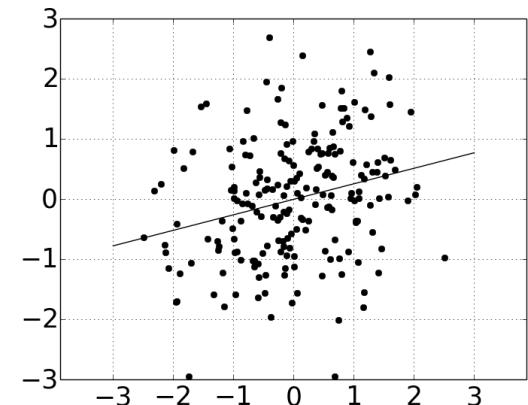
# Examples



$$\rho_{ab} = 0.91$$



$$\rho_{ab} = -0.89$$



$$\rho_{ab} = 0.25$$

- dashed lines in top row show average  $\pm$  standard deviation
- bottom row shows scatterplots of top row in standard units

# Outline

- norm
- distance
- angle
- **hyperplanes**
- complex vectors

# Hyperplane

one linear equation in  $n$  variables  $x_1, x_2, \dots, x_n$ :

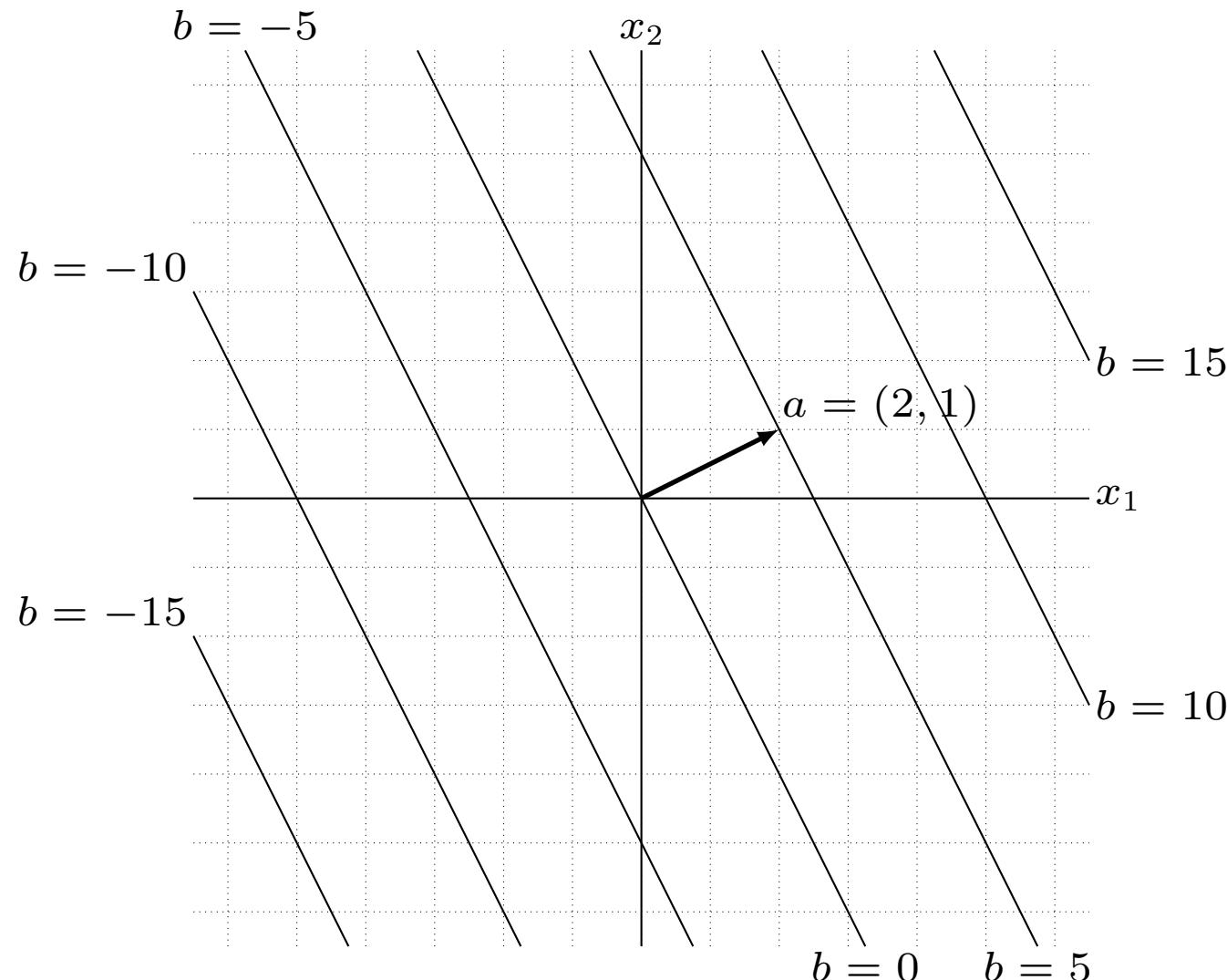
$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

in vector notation:  $a^T x = b$

let  $H$  be the set of solutions:  $H = \{x \in \mathbf{R}^n \mid a^T x = b\}$

- $H$  is empty if  $a_1 = a_2 = \cdots = a_n = 0$  and  $b \neq 0$
- $H = \mathbf{R}^n$  if  $a_1 = a_2 = \cdots = a_n = 0$  and  $b = 0$
- $H$  is called a *hyperplane* if  $a = (a_1, a_2, \dots, a_n) \neq 0$
- for  $n = 2$ , a straight line in a plane; for  $n = 3$ , a plane in 3-D space, . . .

# Example



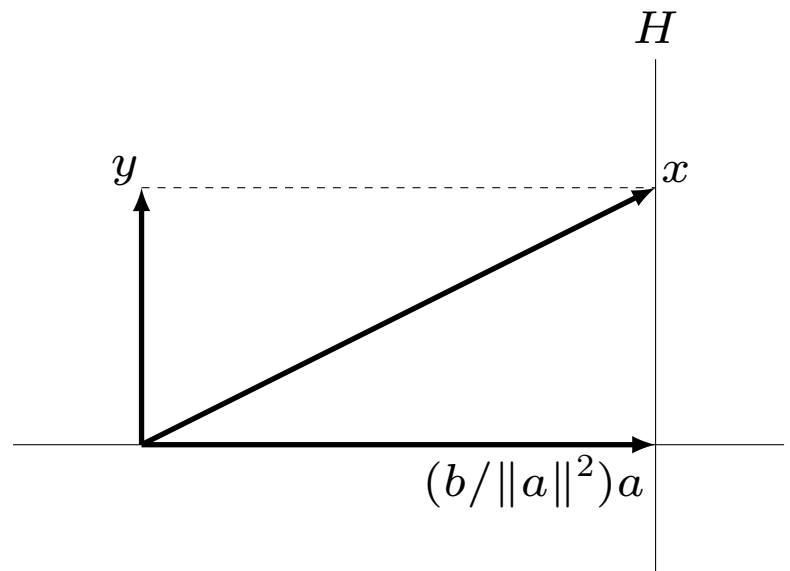
## Geometric interpretation of hyperplane

- recall formula for orthogonal decomposition of  $x$  w.r.t.  $a$  (page 2-25):

$$x = \frac{a^T x}{\|a\|^2} a + y \quad \text{with } y \perp a$$

- $x$  satisfies  $a^T x = b$  if and only if

$$x = \frac{b}{\|a\|^2} a + y \quad \text{with } y \perp a$$



- point  $(b/\|a\|^2)a$  is the intersection of hyperplane with line through  $a$
- add arbitrary vectors  $y \perp a$  to get all other points in hyperplane

## Exercise: projection on hyperplane

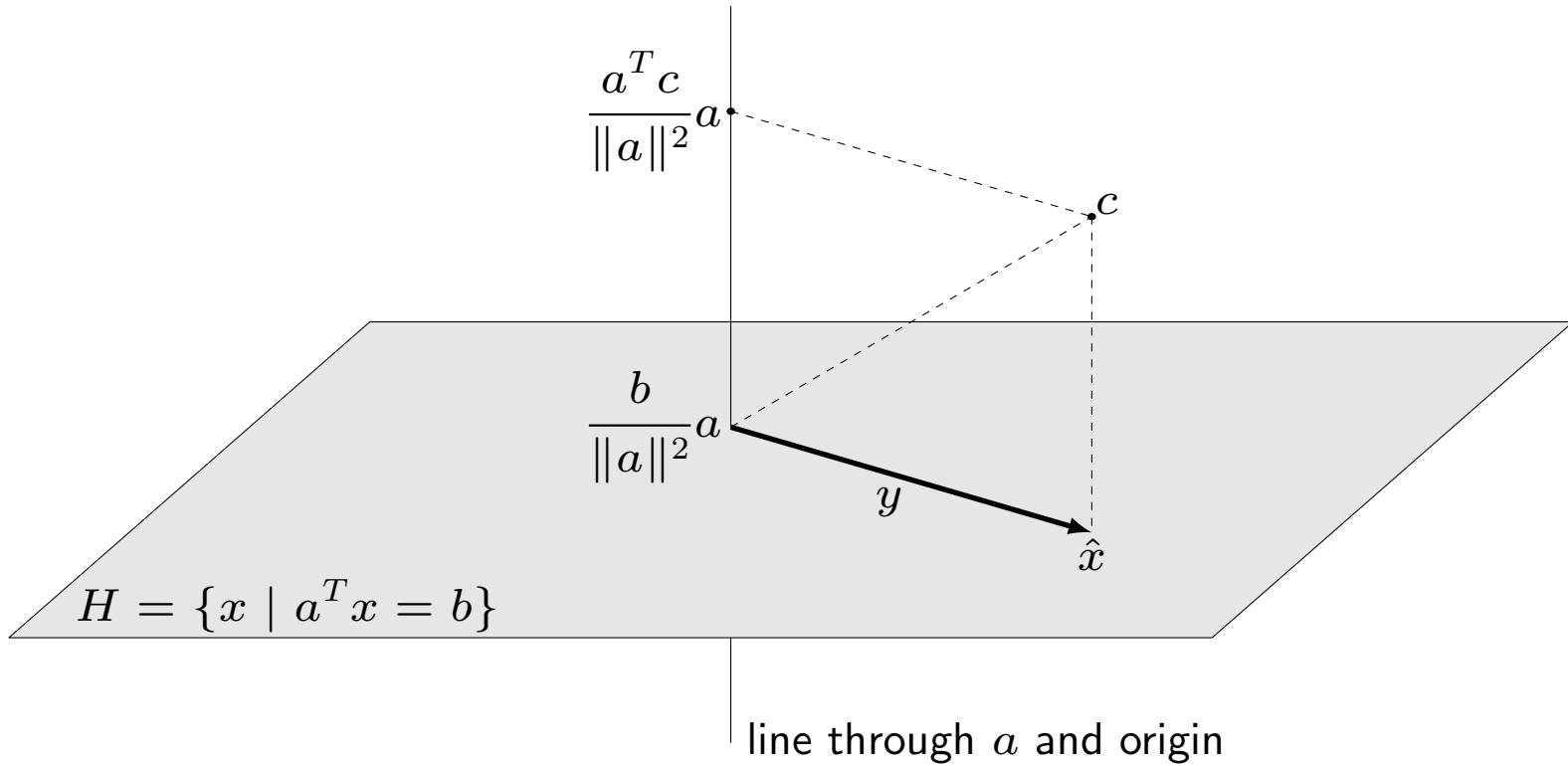
show that the point in  $H = \{x \mid a^T x = b\}$  closest to  $c \in \mathbf{R}^n$  is

$$\hat{x} = c + \frac{b - a^T c}{\|a\|^2} a$$

show that the distance of  $c$  to the hyperplane  $H = \{x \mid a^T x = b\}$  is

$$\frac{|a^T c - b|}{\|a\|}$$

# Solution



$$\hat{x} = c + \frac{b - a^T c}{\|a\|^2} a$$

## Solution

- general point  $x$  in  $H$  is

$$x = \frac{b}{\|a\|^2}a + y, \quad y \perp a$$

- decomposition of  $c$  with respect to  $a$  is

$$c = \frac{a^T c}{\|a\|^2}a + d \quad \text{with } d = c - \frac{a^T c}{\|a\|^2}a$$

- squared distance between  $x$  and  $c$  is

$$\|c - x\|^2 = \left\| \frac{a^T c - b}{a^T a}a + d - y \right\|^2 = \frac{(a^T c - b)^2}{\|a\|^2} + \|d - y\|^2$$

(2nd step because  $d - y \perp a$ ); distance is minimized by choosing  $y = d$

# Kaczmarz algorithm

**Problem:** find (one) solution of set of linear equations

$$a_1^T x = b_1, \quad a_2^T x = b_2, \quad \dots, \quad a_m^T x = b_m$$

- here  $a_1, a_2, \dots, a_m$  are nonzero  $n$ -vectors
- we assume the equations are solvable (have at least one solution)
- $n$  is huge, so we can only use simple vector operations

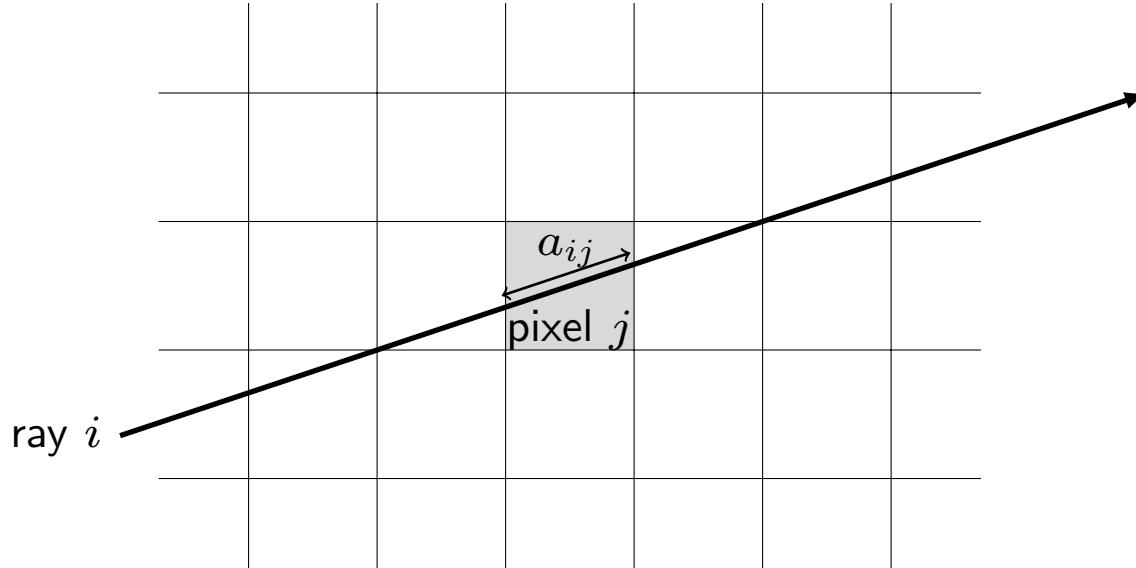
**Algorithm:** start at some initial  $x$  and repeat the following steps

- pick an index  $i = \{1, \dots, m\}$ , for example, cyclically or randomly
- replace  $x$  with projection on hyperplane  $H_i = \{\tilde{x} \mid a_i^T \tilde{x} = b_i\}$

$$x := x + \frac{b_i - a_i^T x}{\|a_i\|^2} a_i$$

# Tomography

reconstruct unknown image from line integrals



- $x$  represents unknown image with  $n$  pixels
- $a_{ij}$  is length of intersection of ray  $i$  and pixel  $j$
- $b_i$  is a measurement of the line integral  $\sum_{j=1}^n a_{ij}x_j$  along ray  $i$

Kaczmarz alg. is also known as *Algebraic Reconstruction Technique* (ART)

# Outline

- norm
- distance
- angle
- hyperplanes
- **complex vectors**

# Norm

norm of vector  $a \in \mathbf{C}^n$ :

$$\begin{aligned}\|a\| &= \sqrt{|a_1|^2 + |a_2|^2 + \cdots + |a_n|^2} \\ &= \sqrt{a^H a}\end{aligned}$$

- nonnegative definite:

$$\|a\| \geq 0 \quad \text{for all } a, \quad \|a\| = 0 \quad \text{only if } a = 0$$

- homogeneous:

$$\|\beta a\| = |\beta| \|a\| \quad \text{for all vectors } a, \text{ complex scalars } \beta$$

- *triangle inequality*

$$\|a + b\| \leq \|a\| + \|b\| \quad \text{for all vectors } a, b \text{ of equal length}$$

# Cauchy-Schwarz inequality for complex vectors

$$|a^H b| \leq \|a\| \|b\| \quad \text{for all } a, b \in \mathbf{C}^n$$

moreover, equality  $|a^H b| = \|a\| \|b\|$  holds if:

- $a = 0$  or  $b = 0$
- $a \neq 0$  and  $b \neq 0$ , and  $b = \gamma a$  for some (complex) scalar  $\gamma$
- exercise: generalize proof for real vectors on page 2-4
- we say  $a$  and  $b$  are *orthogonal* if  $a^H b = 0$
- we will not need definition of angle, correlation coefficient, . . . in  $\mathbf{C}^n$

### 3. Matrices

- notation and terminology
- matrix operations
- linear and affine functions
- complexity

# Matrix

a rectangular array of numbers, for example

$$A = \begin{bmatrix} 0 & 1 & -2.3 & 0.1 \\ 1.3 & 4 & -0.1 & 0 \\ 4.1 & -1 & 0 & 1.7 \end{bmatrix}$$

- numbers in array are the *elements* (*components*, *entries*, *coefficients*)
- $A_{ij}$  is the  $i, j$  element of  $A$ ;  $i$  is its *row index*,  $j$  the *column index*
- size of the matrix is (#rows)  $\times$  (#columns), e.g.,  $A$  is a  $3 \times 4$  matrix
- set of  $m \times n$  matrices with real elements is written  $\mathbf{R}^{m \times n}$
- set of  $m \times n$  matrices with complex elements is written  $\mathbf{C}^{m \times n}$

## Other conventions

- many authors use parentheses as delimiters:

$$A = \begin{pmatrix} 0 & 1 & -2.3 & 0.1 \\ 1.3 & 4 & -0.1 & 0 \\ 4.1 & -1 & 0 & 1.7 \end{pmatrix}$$

- often  $a_{ij}$  is used to denote the  $i, j$  element of  $A$

# Matrix shapes

**Scalar:** we don't distinguish between a  $1 \times 1$  matrix and a scalar

**Vector:** we don't distinguish between an  $n \times 1$  matrix and an  $n$ -vector

## Row and column vectors

- a  $1 \times n$ -matrix is called a *row vector*
- an  $n \times 1$ -matrix is called a *column vector* (or just *vector*)

**Tall, wide, square matrices:** an  $m \times n$ -matrix is

- *tall* if  $m > n$
- *wide* if  $m < n$
- *square* if  $m = n$

# Block matrix

- a *block matrix* is a rectangular array of matrices
- elements in the array are the *blocks* or *submatrices* of the block matrix

## Example

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$$

is a  $2 \times 2$  block matrix; if the blocks are

$$B = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 2 & 3 \\ 5 & 4 & 7 \end{bmatrix}, \quad D = \begin{bmatrix} 1 \end{bmatrix}, \quad E = \begin{bmatrix} -1 & 6 & 0 \end{bmatrix}$$

then

$$A = \begin{bmatrix} 2 & 0 & 2 & 3 \\ 1 & 5 & 4 & 7 \\ 1 & -1 & 6 & 0 \end{bmatrix}$$

**Note:** dimensions of the blocks must be compatible!

## Rows and columns

a matrix can be viewed as a block matrix with row/column vector blocks

- $m \times n$  matrix  $A$  as  $1 \times n$  block matrix

$$A = [ \begin{array}{cccc} a_1 & a_2 & \cdots & a_n \end{array} ]$$

each  $a_j$  is an  $m$ -vector (the  $j$ th *column* of  $A$ )

- $m \times n$  matrix  $A$  as  $m \times 1$  block matrix

$$A = \left[ \begin{array}{c} b_1 \\ b_2 \\ \vdots \\ b_m \end{array} \right]$$

each  $b_i$  is a  $1 \times n$  row vector (the  $i$ th *row* of  $A$ )

# Special matrices

## Zero matrix

- matrix with  $A_{ij} = 0$  for all  $i, j$
- notation:  $0$  (usually) or  $0_{m \times n}$  (if dimension is not clear from context)

## Identity matrix

- square matrix with  $A_{ij} = 1$  if  $i = j$  and  $A_{ij} = 0$  if  $i \neq j$
- notation:  $I$  (usually) or  $I_n$  (if dimension is not clear from context)
- columns of  $I_n$  are unit vectors  $e_1, e_2, \dots, e_n$ ; for example,

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [ \ e_1 \ e_2 \ e_3 \ ]$$

# Symmetric and Hermitian matrices

**Symmetric matrix:** square with  $A_{ij} = A_{ji}$

$$\begin{bmatrix} 4 & 3 & -2 \\ 3 & -1 & 5 \\ -2 & 5 & 0 \end{bmatrix}, \quad \begin{bmatrix} 4 + 3j & 3 - 2j & 0 \\ 3 - 2j & -j & -2j \\ 0 & -2j & 3 \end{bmatrix}$$

**Hermitian matrix:** square with  $A_{ij} = \bar{A}_{ji}$  (complex conjugate of  $A_{ij}$ )

$$\begin{bmatrix} 4 & 3 - 2j & -1 + j \\ 3 + 2j & -1 & 2j \\ -1 - j & -2j & 3 \end{bmatrix}$$

note: diagonal elements are real (since  $A_{ii} = \bar{A}_{ii}$ )

## Structured matrices

matrices with special patterns or structure arise in many applications

- diagonal matrix: square with  $A_{ij} = 0$  for  $i \neq j$

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -5 \end{bmatrix}, \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -5 \end{bmatrix}$$

- lower triangular matrix: square with  $A_{ij} = 0$  for  $i < j$

$$\begin{bmatrix} 4 & 0 & 0 \\ 3 & -1 & 0 \\ 0 & 5 & -2 \end{bmatrix}$$

- upper triangular matrix: square with  $A_{ij} = 0$  for  $i > j$

# Sparse matrices

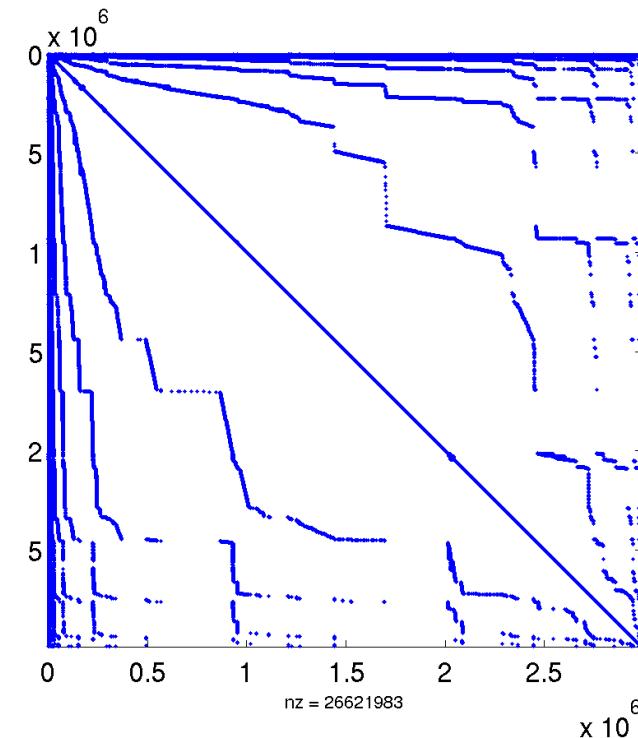
a matrix is *sparse* if most (almost all) of its elements are zero

- sparse matrix storage formats and algorithms exploit sparsity
- efficiency depends on number of nonzeros and their positions
- positions of nonzeros are visualized in a ‘spy plot’

## Example

- 2,987,012 rows and columns
- 26,621,983 nonzeros

(Freescale/FullChip matrix from Univ.  
of Florida Sparse Matrix Collection)



# Outline

- notation and terminology
- **matrix operations**
- linear and affine functions
- complexity

# Scalar-matrix multiplication and addition

## Scalar-matrix multiplication:

scalar-matrix product of  $m \times n$  matrix  $A$  with scalar  $\beta$

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

$A$  and  $\beta$  can be real or complex

**Addition:** sum of two  $m \times n$  matrices  $A$  and  $B$  (real or complex)

$$A + B = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & \cdots & A_{1n} + B_{1n} \\ A_{21} + B_{21} & A_{22} + B_{22} & \cdots & A_{2n} + B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} + B_{m1} & A_{m2} + B_{m2} & \cdots & A_{mn} + B_{mn} \end{bmatrix}$$

# Transpose

the *transpose* of an  $m \times n$  matrix  $A$  is the  $n \times m$  matrix

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

- $(A^T)^T = A$
- a symmetric matrix satisfies  $A = A^T$
- $A$  may be complex, but transpose of complex matrices is rarely needed
- transpose of matrix-scalar product and matrix sum

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

## Conjugate transpose

the *conjugate transpose* of an  $m \times n$  matrix  $A$  is the  $n \times m$  matrix

$$A^H = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{21} & \cdots & \bar{A}_{m1} \\ \bar{A}_{12} & \bar{A}_{22} & \cdots & \bar{A}_{m2} \\ \vdots & \vdots & & \vdots \\ \bar{A}_{1n} & \bar{A}_{2n} & \cdots & \bar{A}_{mn} \end{bmatrix}$$

( $\bar{A}_{ij}$  is complex conjugate of  $A_{ij}$ )

- $A^H = A^T$  if  $A$  is a real matrix
- a Hermitian matrix satisfies  $A = A^H$
- conjugate transpose of matrix-scalar product and matrix sum

$$(\beta A)^H = \bar{\beta} A^H, \quad (A + B)^H = A^H + B^H$$

## Matrix-matrix product

product of  $m \times n$  matrix  $A$  and  $n \times p$  matrix  $B$  ( $A, B$  real or complex)

$$C = AB$$

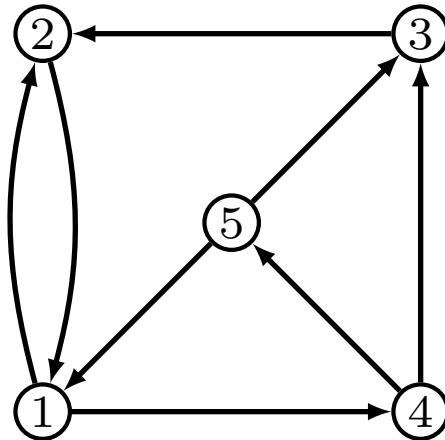
is the  $m \times p$  matrix with elements

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \cdots + A_{in}B_{nj}$$

dimensions must be compatible: #columns in  $A$  = #rows in  $B$

## Exercise: paths in directed graph

directed graph with  $n = 5$  vertices



matrix representation

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$A_{ij} = 1$  indicates an edge  $j \rightarrow i$

**Question:** give a graph interpretation of  $A^2 = AA$ ,  $A^3 = AAA, \dots$

$$A^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad A^3 = \begin{bmatrix} 1 & 1 & 0 & 1 & 2 \\ 2 & 0 & 1 & 2 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad \dots$$

## Properties of matrix-matrix product

- *associative*:  $(AB)C = A(BC)$  so we write  $ABC$
- *associative with scalar-matrix multiplication*:  $(\gamma A)B = \gamma(AB) = \gamma AB$
- *distributes with sum*:

$$A(B + C) = AB + AC, \quad (A + B)C = AC + BC$$

- *transpose and conjugate transpose of product*:

$$(AB)^T = B^T A^T, \quad (AB)^H = B^H A^H$$

- *not commutative*:  $AB \neq BA$  in general; for example,

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \neq \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

there are exceptions, e.g.,  $AI = IA$  for square  $A$

## Notation for vector inner product

- inner product of  $a, b \in \mathbf{R}^n$  (see page 1-18):

$$b^T a = b_1 a_1 + b_2 a_2 + \cdots + b_n a_n = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}^T \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

product of the transpose of column vector  $b$  and column vector  $a$

- inner product of  $a, b \in \mathbf{C}^n$  (see page 1-25):

$$b^H a = \bar{b}_1 a_1 + \bar{b}_2 a_2 + \cdots + \bar{b}_n a_n = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}^H \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

product of conjugate transpose of column vector  $b$  and column vector  $a$

# Matrix-matrix product and block matrices

block-matrices can be multiplied as regular matrices

**Example:** product of two  $2 \times 2$  block matrices

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} W & Y \\ X & Z \end{bmatrix} = \begin{bmatrix} AW + BX & AY + BZ \\ CW + DX & CY + DZ \end{bmatrix}$$

if the dimensions of the blocks are compatible

# Outline

- notation and terminology
- matrix operations
- **linear and affine functions**
- complexity

## Matrix-vector product

product of  $m \times n$  matrix  $A$  with  $n$ -vector (or  $n \times 1$  matrix)  $x$

$$Ax = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n \\ A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n \\ \vdots \\ A_{m1}x_1 + A_{m2}x_2 + \cdots + A_{mn}x_n \end{bmatrix}$$

- dimensions must be compatible: #columns of  $A$  = length of  $x$
- $Ax$  is a linear combination of the columns of  $A$ :

$$Ax = [ \quad a_1 \quad a_2 \quad \cdots \quad a_n \quad ] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = x_1a_1 + x_2a_2 + \cdots + x_na_n$$

each  $a_i$  is an  $m$ -vector ( $i$ th column of  $A$ )

## Linear function

a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  is **linear** if superposition holds:

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

for all  $n$ -vectors  $x, y$  and all scalars  $\alpha, \beta$

**Extension:** if  $f$  is linear, superposition holds for any linear combination:

$$f(\alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_p u_p) = \alpha_1 f(u_1) + \alpha_2 f(u_2) + \cdots + \alpha_p f(u_p)$$

for all scalars,  $\alpha_1, \dots, \alpha_p$  and all  $n$ -vectors  $u_1, \dots, u_p$

## Matrix-vector product function

for fixed  $A \in \mathbf{R}^{m \times n}$ , define a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  as

$$f(x) = Ax$$

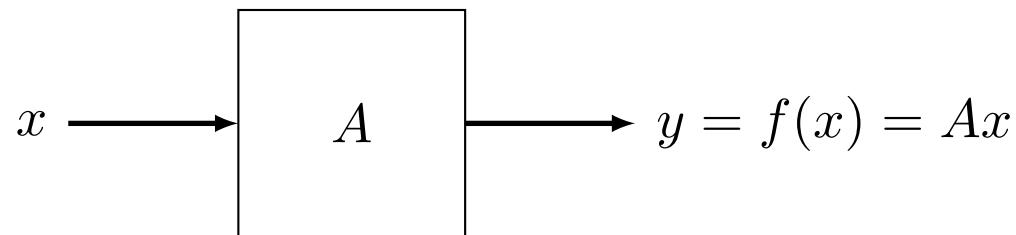
- any function of this type is linear:  $A(\alpha x + \beta y) = \alpha(Ax) + \beta(Ay)$
- every linear function can be written as a matrix-vector product function:

$$\begin{aligned} f(x) &= f(x_1 e_1 + x_2 e_2 + \cdots + x_n e_n) \\ &= x_1 f(e_1) + x_2 f(e_2) + \cdots + x_n f(e_n) \\ &= \begin{bmatrix} f(e_1) & \cdots & f(e_n) \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \end{aligned}$$

hence,  $f(x) = Ax$  with  $A = \begin{bmatrix} f(e_1) & f(e_2) & \cdots & f(e_n) \end{bmatrix}$

## Input-output (operator) interpretation

think of a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  in terms of its effect on  $x$



- signal processing/control interpretation:  $n$  inputs  $x_i$ ,  $m$  outputs  $y_i$
- $f$  is linear if we can represent its action on  $x$  as a product  $f(x) = Ax$

## Examples ( $f : \mathbf{R}^3 \rightarrow \mathbf{R}^3$ )

- $f$  reverses the order of the components of  $x$

a linear function:  $f(x) = Ax$  with

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

- $f$  sorts the components of  $x$  in decreasing order: not linear
- $f$  scales  $x_1$  by a given number  $d_1$ ,  $x_2$  by  $d_2$ ,  $x_3$  by  $d_3$

a linear function:  $f(x) = Ax$  with

$$A = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix}$$

- $f$  replaces each  $x_i$  by its absolute value  $|x_i|$ : not linear

# Operator interpretation of matrix-matrix product

explains why in general  $AB \neq BA$



## Example

$$A = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- $f(x) = ABx$  swaps the elements of  $x$ ; then changes sign of first element
- $f(x) = BAx$  changes sign of 1st element; then swaps the two elements

## Reversal and circular shift

### Reversal matrix

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}, \quad Ax = (x_n, x_{n-1}, \dots, x_2, x_1)$$

### Circular shift matrix

$$A = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}, \quad Ax = (x_n, x_1, x_2, \dots, x_{n-1})$$

# Permutation

## Permutation matrix

- a square 0-1 matrix with one element 1 per row and one 1 per column
- equivalently, an identity matrix with columns reordered
- equivalently, an identity matrix with rows reordered

$Ax$  is a permutation of the elements of  $x$

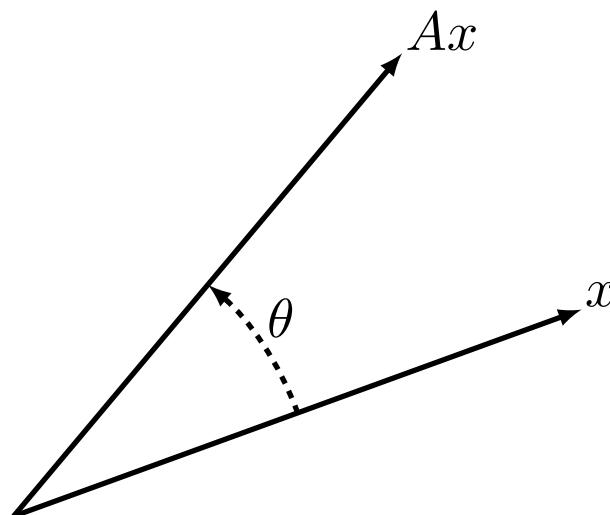
## Example

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad Ax = (x_2, x_4, x_1, x_3)$$

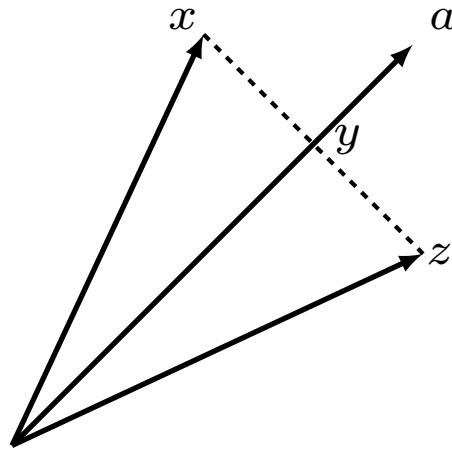
## Rotation in a plane

$$A = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

$Ax$  is  $x$  rotated counterclockwise over an angle  $\theta$



# Projection on line and reflection



- projection on line through  $a$  (see page 2-11):

$$y = \frac{a^T x}{\|a\|^2} a = Ax \quad \text{with} \quad A = \frac{1}{\|a\|^2} a a^T$$

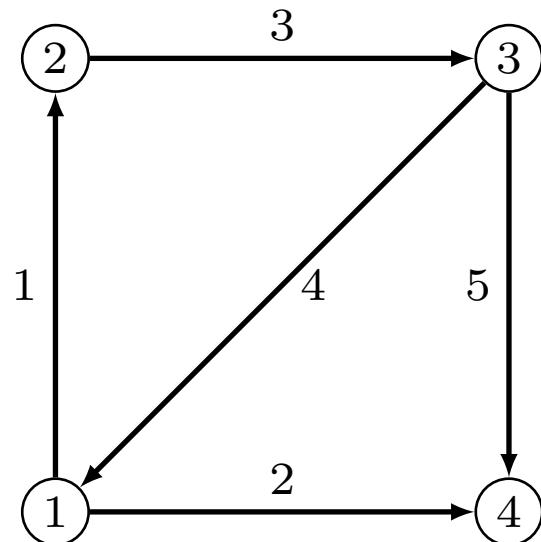
- reflection with respect to line through  $a$

$$z = x + 2(y - x) = Bx, \quad \text{with} \quad B = \frac{2}{\|a\|^2} a a^T - I$$

## Node-arc incidence matrix

- directed graph with  $m$  vertices,  $n$  arcs (directed edges)
- incidence matrix is  $m \times n$  matrix  $A$  with

$$A_{ij} = \begin{cases} 1 & \text{if arc } j \text{ enters node } i \\ -1 & \text{if arc } j \text{ leaves node } i \\ 0 & \text{otherwise} \end{cases}$$



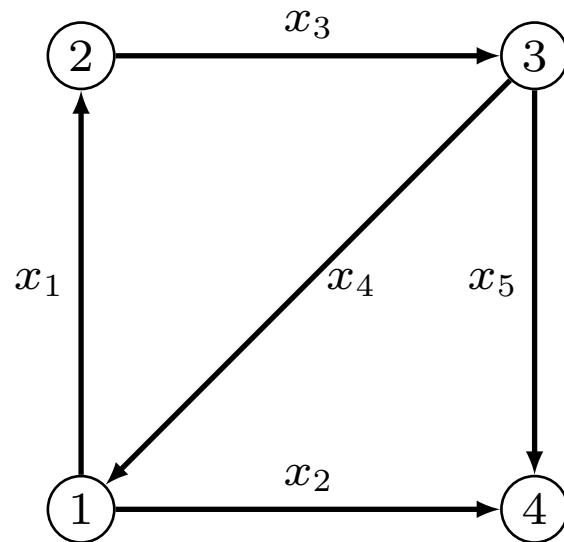
$$A = \begin{bmatrix} -1 & -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

## Kirchhoff's current law

$n$ -vector  $x = (x_1, x_2, \dots, x_n)$  with  $x_j$  the current through arc  $j$

$$(Ax)_i = \sum_{\substack{\text{arc } j \text{ enters} \\ \text{node } i}} x_j - \sum_{\substack{\text{arc } j \text{ leaves} \\ \text{node } i}} x_j$$

= total current arriving at node  $i$

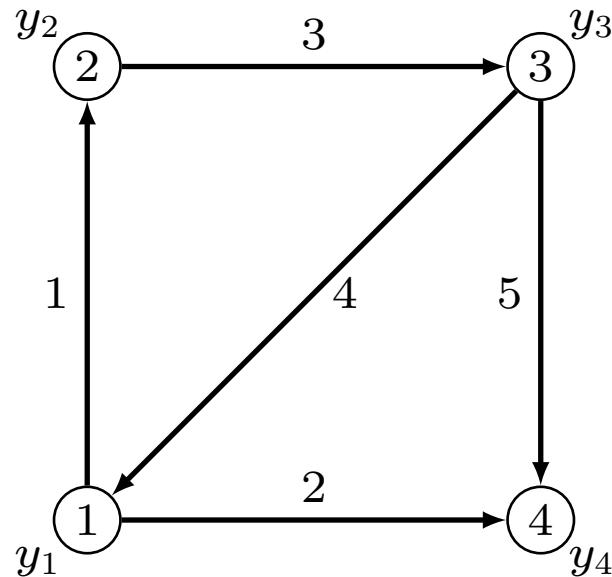


$$Ax = \begin{bmatrix} -x_1 - x_2 + x_4 \\ x_1 - x_3 \\ x_3 - x_4 - x_5 \\ x_2 + x_5 \end{bmatrix}$$

## Kirchhoff's voltage law

$m$ -vector  $y = (y_1, y_2, \dots, y_m)$  with  $y_i$  the potential at node  $i$

$$\begin{aligned}(A^T y)_j &= y_k - y_l && \text{if edge } j \text{ goes from node } l \text{ to } k \\ &= \text{negative of voltage across arc } j\end{aligned}$$



$$A^T y = \begin{bmatrix} y_2 - y_1 \\ y_4 - y_1 \\ y_3 - y_2 \\ y_1 - y_3 \\ y_4 - y_3 \end{bmatrix}$$

# Vandermonde matrix

- polynomial of degree  $n - 1$  or less with coefficients  $x_1, x_2, \dots, x_n$ :

$$p(t) = x_1 + x_2t + x_3t^2 + \cdots + x_nt^{n-1}$$

- values of  $p(t)$  at  $m$  points  $t_1, \dots, t_m$ :

$$\begin{bmatrix} p(t_1) \\ p(t_2) \\ \vdots \\ p(t_m) \end{bmatrix} = \begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & t_m & \cdots & t_m^{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = Ax$$

the matrix  $A$  is called a *Vandermonde* matrix

- $f(x) = Ax$  maps coefficients of polynomial to function values

# Discrete Fourier transform

the DFT maps a complex  $n$ -vector  $(x_1, x_2, \dots, x_n)$  to the complex  $n$ -vector

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

$$= Wx$$

where  $\omega = e^{2\pi j/n}$  (and  $j = \sqrt{-1}$ )

- DFT matrix  $W \in \mathbb{C}^{n \times n}$  has  $k, l$  element  $W_{kl} = \omega^{-(k-1)(l-1)}$
- a Vandermonde matrix with  $m = n$  and

$$t_1 = 1, \quad t_2 = \omega^{-1}, \quad t_3 = \omega^{-2}, \quad \dots, \quad t_n = \omega^{-(n-1)}$$

## Affine function

a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  is **affine** if it satisfies

$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

for all  $n$ -vectors  $x, y$  and all scalars  $\alpha, \beta$  with  $\alpha + \beta = 1$

**Extension:** if  $f$  is affine, then

$$f(\alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_m u_m) = \alpha_1 f(u_1) + \alpha_2 f(u_2) + \cdots + \alpha_m f(u_m)$$

for all  $n$ -vectors  $u_1, \dots, u_m$  and all scalars  $\alpha_1, \dots, \alpha_m$  with

$$\alpha_1 + \alpha_2 + \cdots + \alpha_m = 1$$

## Affine functions and matrix-vector product

for fixed  $A \in \mathbf{R}^{m \times n}$ ,  $b \in \mathbf{R}^m$ , define a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  by

$$f(x) = Ax + b$$

i.e., a matrix-vector product plus a constant

- any function of this type is affine: if  $\alpha + \beta = 1$  then

$$A(\alpha x + \beta y) + b = \alpha(Ax + b) + \beta(Ay + b)$$

- every affine function can be written as  $f(x) = Ax + b$  with:

$$A = \begin{bmatrix} f(e_1) - f(0) & f(e_2) - f(0) & \cdots & f(e_n) - f(0) \end{bmatrix}$$

and  $b = f(0)$

# Affine approximation

first-order Taylor approximation of differentiable  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  around  $z$ :

$$\widehat{f}_i(x) = f_i(z) + \frac{\partial f_i}{\partial x_1}(z)(x_1 - z_1) + \cdots + \frac{\partial f_i}{\partial x_n}(z)(x_n - z_n), \quad i = 1, \dots, m$$

in matrix-vector notation:  $\widehat{f}(x) = f(z) + Df(z)(x - z)$  where

$$Df(z) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(z) & \frac{\partial f_1}{\partial x_2}(z) & \cdots & \frac{\partial f_1}{\partial x_n}(z) \\ \frac{\partial f_2}{\partial x_1}(z) & \frac{\partial f_2}{\partial x_2}(z) & \cdots & \frac{\partial f_2}{\partial x_n}(z) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(z) & \frac{\partial f_m}{\partial x_2}(z) & \cdots & \frac{\partial f_m}{\partial x_n}(z) \end{bmatrix} = \begin{bmatrix} \nabla f_1(z)^T \\ \nabla f_2(z)^T \\ \vdots \\ \nabla f_m(z)^T \end{bmatrix}$$

- $Df(z)$  is called the *derivative matrix* or *Jacobian matrix* of  $f$  at  $z$
- $\widehat{f}$  is a local affine approximation of  $f$  around  $z$

# Example

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} e^{2x_1+x_2} - x_1 \\ x_1^2 - x_2 \end{bmatrix}$$

- derivative matrix

$$Df(x) = \begin{bmatrix} 2e^{2x_1+x_2} - 1 & e^{2x_1+x_2} \\ 2x_1 & -1 \end{bmatrix}$$

- first order approximation of  $f$  around  $z = 0$ :

$$\hat{f}(x) = \begin{bmatrix} \hat{f}_1(x) \\ \hat{f}_2(x) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

# Outline

- notation and terminology
- matrix operations
- linear and affine functions
- **complexity**

## Matrix-vector product

matrix-vector multiplication of  $m \times n$  matrix  $A$  and  $n$ -vector  $x$ :

$$y = Ax$$

requires  $(2n - 1)m$  flops

- $m$  elements in  $y$ ; each element requires an inner product of length  $n$
- approximately  $2mn$  for large  $n$

**Special cases:** flop count is lower for structured matrices

- $A$  diagonal:  $n$  flops
- $A$  lower triangular:  $n^2$  flops
- $A$  sparse: #flops  $\ll 2mn$

## Matrix-matrix product

product of  $m \times n$  matrix  $A$  and  $n \times p$  matrix  $B$ :

$$C = AB$$

requires  $mp(2n - 1)$  flops

- $mp$  elements in  $C$ ; each element requires an inner product of length  $n$
- approximately  $2mnp$  for large  $n$

## Exercises

1. evaluate  $y = ABx$  two ways ( $A$  and  $B$  are  $n \times n$ ,  $x$  is a vector)
  - $y = (AB)x$  (first make product  $C = AB$ , then multiply  $C$  with  $x$ )
  - $y = A(Bx)$  (first make product  $y = Bx$ , then multiply  $A$  with  $y$ )

both methods give the same answer, but which method is faster?
2. evaluate  $y = (I + uv^T)x$  where  $u, v, x$  are  $n$ -vectors
  - $A = I + uv^T$  followed by  $y = Ax$   
in MATLAB: `y = (eye(n) + u*v') * x`
  - $w = (v^T x)u$  followed by  $y = x + w$   
in MATLAB: `y = x + (v'*x) * u`

## 4. Matrix inverses

- left and right inverse
- linear independence
- nonsingular matrices
- matrices with linearly independent columns
- matrices with linearly independent rows

## Left and right inverse

$AB \neq BA$  in general, so we have to distinguish two types of inverses

**Left inverse:**  $X$  is a *left inverse* of  $A$  if

$$XA = I$$

$A$  is *left invertible* if it has at least one left inverse

**Right inverse:**  $X$  is a *right inverse* of  $A$  if

$$AX = I$$

$A$  is *right invertible* if it has at least one right inverse

## Examples

$$A = \begin{bmatrix} -3 & -4 \\ 4 & 6 \\ 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

- $A$  is left invertible; the following matrices are left inverses:

$$\frac{1}{9} \begin{bmatrix} -11 & -10 & 16 \\ 7 & 8 & -11 \end{bmatrix}, \quad \begin{bmatrix} 0 & -1/2 & 3 \\ 0 & 1/2 & -2 \end{bmatrix}$$

- $B$  is right invertible; the following matrices are right inverses:

$$\frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 & -1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

# Some immediate properties

## Dimensions

a left or right inverse of an  $m \times n$  matrix must have size  $n \times m$

## Left and right inverse of (conjugate) transpose

- $X$  is a left inverse of  $A$  if and only if  $X^T$  is a right inverse of  $A^T$

$$A^T X^T = (XA)^T = I$$

- $X$  is a left inverse of  $A$  if and only if  $X^H$  is a right inverse of  $A^H$

$$A^H X^H = (XA)^H = I$$

# Inverse

if  $A$  has a left **and** a right inverse, then they are equal and unique:

$$XA = I, \quad AY = I \quad \implies \quad X = X(AY) = (XA)Y = Y$$

- we call  $X = Y$  the **inverse** of  $A$  (notation:  $A^{-1}$ )
- $A$  is *invertible* if its inverse exists

## Example

$$A = \begin{bmatrix} -1 & 1 & -3 \\ 1 & -1 & 1 \\ 2 & 2 & 2 \end{bmatrix}, \quad A^{-1} = \frac{1}{4} \begin{bmatrix} 2 & 4 & 1 \\ 0 & -2 & 1 \\ -2 & -2 & 0 \end{bmatrix}$$

# Linear equations

set of  $m$  linear equations in  $n$  variables

$$A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n = b_1$$

$$A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n = b_2$$

⋮

$$A_{m1}x_1 + A_{m2}x_2 + \cdots + A_{mn}x_n = b_m$$

- in matrix form:  $Ax = b$
- may have no solution, a unique solution, infinitely many solutions

# Linear equations and matrix inverse

**Left invertible matrix:** if  $X$  is a left inverse of  $A$ , then

$$Ax = b \implies x = XAx = Xb$$

there is *at most one* solution (if there is a solution, it must be equal to  $Xb$ )

**Right invertible matrix:** if  $X$  is a right inverse of  $A$ , then

$$x = Xb \implies Ax = AXb = b$$

there is *at least one* solution (namely,  $x = Xb$ )

**Invertible matrix:** if  $A$  is invertible, then

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

there is a *unique* solution

# Outline

- left and right inverse
- **linear independence**
- nonsingular matrices
- matrices with linearly independent columns
- matrices with linearly independent rows

# Linear combination

a linear combination of vectors  $a_1, \dots, a_n$  is a sum of scalar products

$$x_1a_1 + x_2a_2 + \cdots + x_na_n$$

- the scalars  $x_i$  are the *coefficients* of the linear combination
- can be written as a matrix-vector product

$$x_1a_1 + x_2a_2 + \cdots + x_na_n = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- the *trivial* linear combination has coefficients  $x_1 = \cdots = x_n = 0$

(same definition holds for real and complex vectors/scalars)

# Linear dependence

a set of vectors  $a_1, a_2, \dots, a_n$  is *linearly dependent* if

$$x_1a_1 + x_2a_2 + \cdots + x_na_n = 0$$

for some scalars  $x_1, \dots, x_n$ , not all zero

- vector 0 can be written as a nontrivial linear combination of  $a_1, \dots, a_n$
- equivalently, at least one vector  $a_i$  is a linear combination of the rest:

$$a_i = -\frac{x_1}{x_i}a_1 - \cdots - \frac{x_{i-1}}{x_i}a_{i-1} - \frac{x_{i+1}}{x_i}a_{i+1} - \cdots - \frac{x_n}{x_i}a_n$$

if  $x_i \neq 0$

## Example

the vectors

$$a_1 = \begin{bmatrix} 0.2 \\ -7 \\ 8.6 \end{bmatrix}, \quad a_2 = \begin{bmatrix} -0.1 \\ 2 \\ -1 \end{bmatrix}, \quad a_3 = \begin{bmatrix} 0 \\ -1 \\ 2.2 \end{bmatrix}$$

are linearly dependent

- 0 can be expressed as a nontrivial linear combination of  $a_1, a_2, a_3$ :

$$0 = a_1 + 2a_2 - 3a_3$$

- $a_1$  can be expressed as a linear combination of  $a_2, a_3$ :

$$a_1 = -2a_2 + 3a_3$$

(and similarly  $a_2$  and  $a_3$ )

# Linear independence

$a_1, \dots, a_n$  are *linearly independent* if they are not linearly dependent

- the zero vector cannot be written as a nontrivial linear combination:

$$x_1a_1 + x_2a_2 + \cdots + x_na_n = 0 \quad \Rightarrow \quad x_1 = x_2 = \cdots = x_n = 0$$

- none of the vectors  $a_i$  is a linear combination of the other vectors

## Matrix with linearly independent columns

$$A = [ \begin{array}{cccc} a_1 & a_2 & \cdots & a_n \end{array} ]$$

has linearly independent columns if  $Ax = 0$  only for  $x = 0$

## Example

the vectors

$$a_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}, \quad a_2 = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \quad a_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

are linearly independent:

$$x_1 a_1 + x_2 a_2 + x_3 a_3 = \begin{bmatrix} x_1 - x_2 \\ -2x_1 + x_3 \\ x_2 + x_3 \end{bmatrix} = 0$$

only if  $x_1 = x_2 = x_3 = 0$

## Dimension inequality

if  $n$  vectors  $a_1, a_2, \dots, a_n$  of length  $m$  are linearly independent, then

$$n \leq m$$

(proof is in textbook)

- if an  $m \times n$  matrix has linearly independent columns then  $m \geq n$
- if an  $m \times n$  matrix has linearly independent rows then  $m \leq n$

# Outline

- left and right inverse
- linear independence
- **nonsingular matrices**
- matrices with linearly independent columns
- matrices with linearly independent rows

# Nonsingular matrix

for a **square** matrix  $A$  the following four properties are equivalent

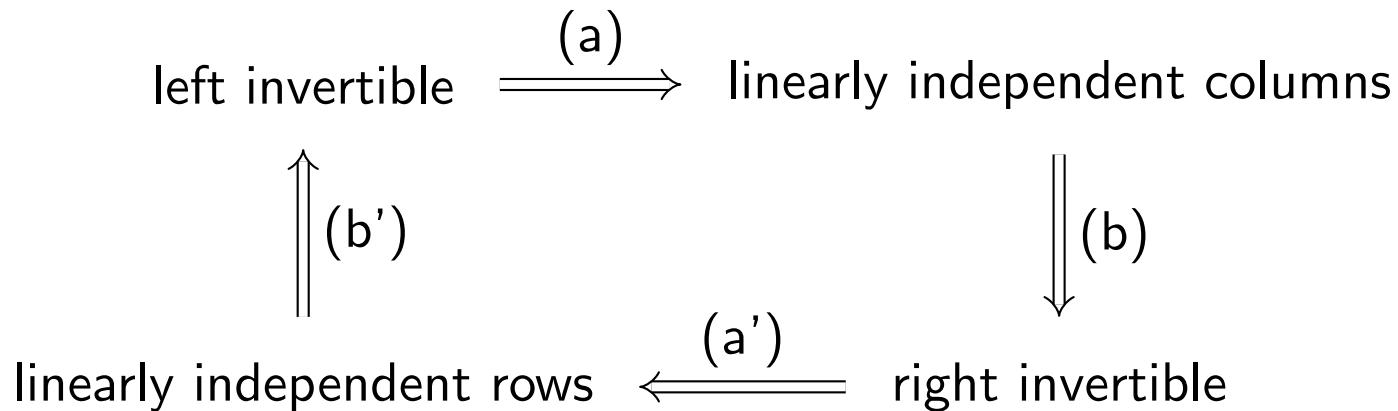
1.  $A$  is left invertible
2. the columns of  $A$  are linearly independent
3.  $A$  is right invertible
4. the rows of  $A$  are linearly independent

a square matrix with these properties is called **nonsingular**

**Nonsingular = invertible**

- if properties 1 and 3 hold, then  $A$  is invertible (page 4-5)
- if  $A$  is invertible, properties 1 and 3 hold (by definition of invertibility)

# Proof



- we show that (a) holds in general
- we show that (b) holds for square matrices
- (a') and (b') follow from (a) and (b) applied to  $A^T$

## Part a: suppose $A$ is left invertible

- if  $B$  is a left inverse of  $A$  (satisfies  $BA = I$ ), then

$$\begin{aligned} Ax = 0 &\implies BAx = 0 \\ &\implies x = 0 \end{aligned}$$

- this means that the columns of  $A$  are linearly independent: if

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

then

$$x_1a_1 + x_2a_2 + \cdots + x_na_n = 0$$

holds only for the trivial linear combination  $x_1 = x_2 = \cdots = x_n = 0$

**Part b:** suppose  $A$  is square with linearly independent columns  $a_1, \dots, a_n$

- for every  $n$ -vector  $b$  the vectors  $a_1, \dots, a_n, b$  are linearly dependent (from inequality on page 4-13)
- hence for every  $b$  there exists a nontrivial linear combination

$$x_1a_1 + x_2a_2 + \cdots + x_na_n + x_{n+1}b = 0$$

we must have  $x_{n+1} \neq 0$  because  $a_1, \dots, a_n$  are linearly independent

- hence every  $b$  can be written as a linear combination of  $a_1, \dots, a_n$
- in particular, there exist  $n$ -vectors  $c_1, \dots, c_n$  such that

$$Ac_1 = e_1, \quad Ac_2 = e_2, \quad \dots, \quad Ac_n = e_n,$$

- the matrix  $C = [ c_1 \ c_2 \ \cdots \ c_n ]$  is a right inverse of  $A$ :

$$A [ c_1 \ c_2 \ \cdots \ c_n ] = [ e_1 \ e_2 \ \cdots \ e_n ] = I$$

## Examples

$$A = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

- $A$  is nonsingular because its columns are linearly independent:

$$x_1 - x_2 + x_3 = 0, \quad -x_1 + x_2 + x_3 = 0, \quad x_1 + x_2 - x_3 = 0$$

is only possible if  $x_1 = x_2 = x_3 = 0$

- $B$  is singular because its columns are linearly dependent:

$$Bx = 0 \quad \text{for } x = (1, 1, 1, 1)$$

## Example: Vandermonde matrix

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^{n-1} \end{bmatrix} \quad \text{with } t_i \neq t_j \text{ for } i \neq j$$

we show that  $A$  is nonsingular by showing that  $Ax = 0$  only if  $x = 0$

- $Ax = 0$  means  $p(t_1) = p(t_2) = \cdots = p(t_n) = 0$  where

$$p(t) = x_1 + x_2t + x_3t^2 + \cdots + x_nt^{n-1}$$

$p(t)$  is a polynomial of degree  $n - 1$  or less

- if  $x \neq 0$ , then  $p(t)$  can not have more than  $n - 1$  distinct real roots
- therefore  $p(t_1) = \cdots = p(t_n) = 0$  is only possible if  $x = 0$

# Inverse of transpose and product

## Transpose and conjugate transpose

if  $A$  is nonsingular, then  $A^T$  and  $A^H$  are nonsingular and

$$(A^T)^{-1} = (A^{-1})^T, \quad (A^H)^{-1} = (A^{-1})^H,$$

we write these as  $A^{-T}$  and  $A^{-H}$

## Product

if  $A$  and  $B$  are nonsingular and of equal size, then  $AB$  is nonsingular with

$$(AB)^{-1} = B^{-1}A^{-1}$$

# Outline

- left and right inverse
- linear independence
- nonsingular matrices
- **matrices with linearly independent columns**
- matrices with linearly independent rows

## Gram matrix

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

the *Gram matrix* associated with  $A$  is the matrix of column inner products

- for real matrices:

$$A^T A = \begin{bmatrix} a_1^T a_1 & a_1^T a_2 & \cdots & a_1^T a_n \\ a_2^T a_1 & a_2^T a_2 & \cdots & a_2^T a_n \\ \vdots & \vdots & & \vdots \\ a_n^T a_1 & a_n^T a_2 & \cdots & a_n^T a_n \end{bmatrix}$$

- for complex matrices

$$A^H A = \begin{bmatrix} a_1^H a_1 & a_1^H a_2 & \cdots & a_1^H a_n \\ a_2^H a_1 & a_2^H a_2 & \cdots & a_2^H a_n \\ \vdots & \vdots & & \vdots \\ a_n^H a_1 & a_n^H a_2 & \cdots & a_n^H a_n \end{bmatrix}$$

## Nonsingular Gram matrix

Gram matrix is nonsingular if and only if  $A$  has linearly independent columns

- suppose  $A \in \mathbf{R}^{m \times n}$  has linearly independent columns:

$$\begin{aligned} A^T A x = 0 &\implies x^T A^T A x = (Ax)^T (Ax) = \|Ax\|^2 = 0 \\ &\implies Ax = 0 \\ &\implies x = 0 \end{aligned}$$

therefore  $A^T A$  is nonsingular

- suppose the columns of  $A \in \mathbf{R}^{m \times n}$  are linearly dependent

$$\exists x \neq 0, Ax = 0 \implies \exists x \neq 0, A^T A x = 0$$

therefore  $A^T A$  is singular

(for  $A \in \mathbf{C}^{m \times n}$ , replace  $A^T$  with  $A^H$  and  $x^T$  with  $x^H$ )

## Pseudo-inverse

- suppose  $A \in \mathbf{R}^{m \times n}$  has linearly independent columns
- this implies that  $A$  is tall or square ( $m \geq n$ ); see page 4-13

the *pseudo-inverse* of  $A$  is defined as

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

- this matrix exists, because the Gram matrix  $A^T A$  is nonsingular
- $A^\dagger$  is a left inverse of  $A$ :

$$A^\dagger A = (A^T A)^{-1} (A^T A) = I$$

(for complex  $A$  with linearly independent columns,  $A^\dagger = (A^H A)^{-1} A^H$ )

# Summary

the following three properties are equivalent

1.  $A$  is left invertible
2. the columns of  $A$  are linearly independent
3.  $A^T A$  is nonsingular

- $1 \Rightarrow 2$  was already proved on page 4-16
- $2 \Rightarrow 1$ : we have seen that the pseudo-inverse is a left inverse
- $2 \Leftrightarrow 3$ : proved on page 4-22
- a matrix with these properties must be tall or square

# Outline

- left and right inverse
- linear independence
- nonsingular matrices
- matrices with linearly independent columns
- **matrices with linearly independent rows**

## Pseudo-inverse

- suppose  $A \in \mathbf{R}^{m \times n}$  has linearly independent rows
- this implies that  $A$  is wide or square ( $m \leq n$ ); see page 4-13

the *pseudo-inverse* of  $A$  is defined as

$$A^\dagger = A^T (AA^T)^{-1}$$

- $A^T$  has linearly independent columns
- hence its Gram matrix  $AA^T$  is nonsingular, so  $A^\dagger$  exists
- $A^\dagger$  is a right inverse of  $A$ :

$$AA^\dagger = (AA^T)(AA^T)^{-1} = I$$

(for complex  $A$  with linearly independent rows,  $A^\dagger = A^H (AA^H)^{-1}$ )

# Summary

the following three properties are equivalent

1.  $A$  is right invertible
2. the rows of  $A$  are linearly independent
3.  $AA^T$  is nonsingular

- $1 \Rightarrow 2$  and  $2 \Leftrightarrow 3$ : by transposing result on page 4-24
- $2 \Rightarrow 1$ : we have seen that the pseudo-inverse is a right inverse
- a matrix with these properties must be wide or square

## 5. Orthogonal matrices

- matrices with orthonormal columns
- orthogonal matrices
- tall matrices with orthonormal columns
- complex matrices with orthonormal columns

# Orthonormal vectors

a set of real  $m$ -vectors  $\{a_1, a_2, \dots, a_n\}$  is *orthonormal* if

- the vectors have unit norm:  $\|a_i\| = 1$
- they are mutually orthogonal:  $a_i^T a_j = 0$  if  $i \neq j$

## Example

$$\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \quad \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

## Matrix with orthonormal columns

$A \in \mathbf{R}^{m \times n}$  has orthonormal columns if its Gram matrix is the identity

$$\begin{aligned} A^T A &= [a_1 \ a_2 \ \cdots \ a_n]^T [a_1 \ a_2 \ \cdots \ a_n] \\ &= \begin{bmatrix} a_1^T a_1 & a_1^T a_2 & \cdots & a_1^T a_n \\ a_2^T a_1 & a_2^T a_2 & \cdots & a_2^T a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n^T a_1 & a_n^T a_2 & \cdots & a_n^T a_n \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \end{aligned}$$

there is no standard short name for ‘matrix with orthonormal columns’

## Matrix-vector product

if  $A \in \mathbf{R}^{m \times n}$  has orthonormal columns, then the linear function  $f(x) = Ax$

- preserves inner products:

$$(Ax)^T(Ay) = x^T A^T A y = x^T y$$

- preserves norms:

$$\|Ax\| = ((Ax)^T(Ax))^{1/2} = (x^T x)^{1/2} = \|x\|$$

- preserves distances:  $\|Ax - Ay\| = \|x - y\|$

- preserves angles:

$$\angle(Ax, Ay) = \arccos \left( \frac{(Ax)^T(Ay)}{\|Ax\| \|Ay\|} \right) = \arccos \left( \frac{x^T y}{\|x\| \|y\|} \right) = \angle(x, y)$$

## Left invertibility

if  $A \in \mathbf{R}^{m \times n}$  has orthonormal columns, then

- $A$  is left invertible with left inverse  $A^T$ : by definition

$$A^T A = I$$

- $A$  has linearly independent columns (from p. 4-24 or p. 5-2):

$$Ax = 0 \quad \implies \quad A^T A x = x = 0$$

- $A$  is tall or square:  $m \geq n$  (see page 4-13)

# Outline

- matrices with orthonormal columns
- **orthogonal matrices**
- tall matrices with orthonormal columns
- complex matrices with orthonormal columns

# Orthogonal matrix

**Orthogonal matrix:** a **square** real matrix with orthonormal columns

**Nonsingularity** (from equivalences on page 4-14): if  $A$  is orthogonal, then

- $A$  is invertible, with inverse  $A^T$ :

$$\left. \begin{array}{l} A^T A = I \\ A \text{ is square} \end{array} \right\} \implies AA^T = I$$

- $A^T$  is also an orthogonal matrix
- rows of  $A$  are orthonormal (have norm one and are mutually orthogonal)

**Note:** if  $A \in \mathbf{R}^{m \times n}$  has orthonormal columns and  $m > n$ , then  $AA^T \neq I$

# Permutation matrix

- let  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$  be a permutation (reordering) of  $(1, 2, \dots, n)$
- we associate with  $\pi$  the  $n \times n$  *permutation matrix*  $A$

$$A_{i\pi_i} = 1, \quad A_{ij} = 0 \text{ if } j \neq \pi_i$$

- $Ax$  is a permutation of the elements of  $x$ :  $Ax = (x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_n})$
- $A$  has exactly one element equal to 1 in each row and each column

**Orthogonality:** permutation matrices are orthogonal

- $A^T A = I$  because  $A$  has exactly one element equal to one in each row

$$(A^T A)_{ij} = \sum_{k=1}^n A_{ki} A_{kj} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

- $A^T = A^{-1}$  is the inverse permutation matrix

## Example

- permutation on  $\{1, 2, 3, 4\}$

$$(\pi_1, \pi_2, \pi_3, \pi_4) = (2, 4, 1, 3)$$

- corresponding permutation matrix and its inverse

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad A^{-1} = A^T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

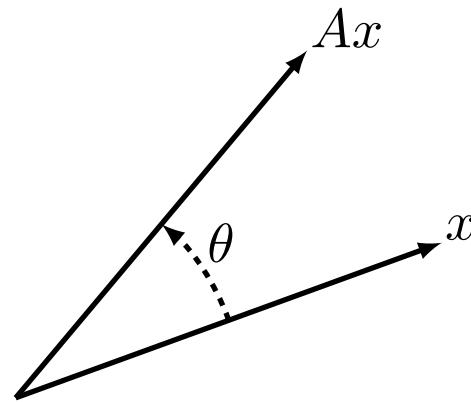
- $A^T$  is permutation matrix associated with the permutation

$$(\tilde{\pi}_1, \tilde{\pi}_2, \tilde{\pi}_3, \tilde{\pi}_4) = (3, 1, 4, 2)$$

# Plane rotation

**Rotation in a plane**

$$A = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$



**Rotation in a coordinate plane in  $\mathbf{R}^n$ :** for example,

$$A = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix}$$

describes a rotation in the  $(x_1, x_3)$  plane in  $\mathbf{R}^3$

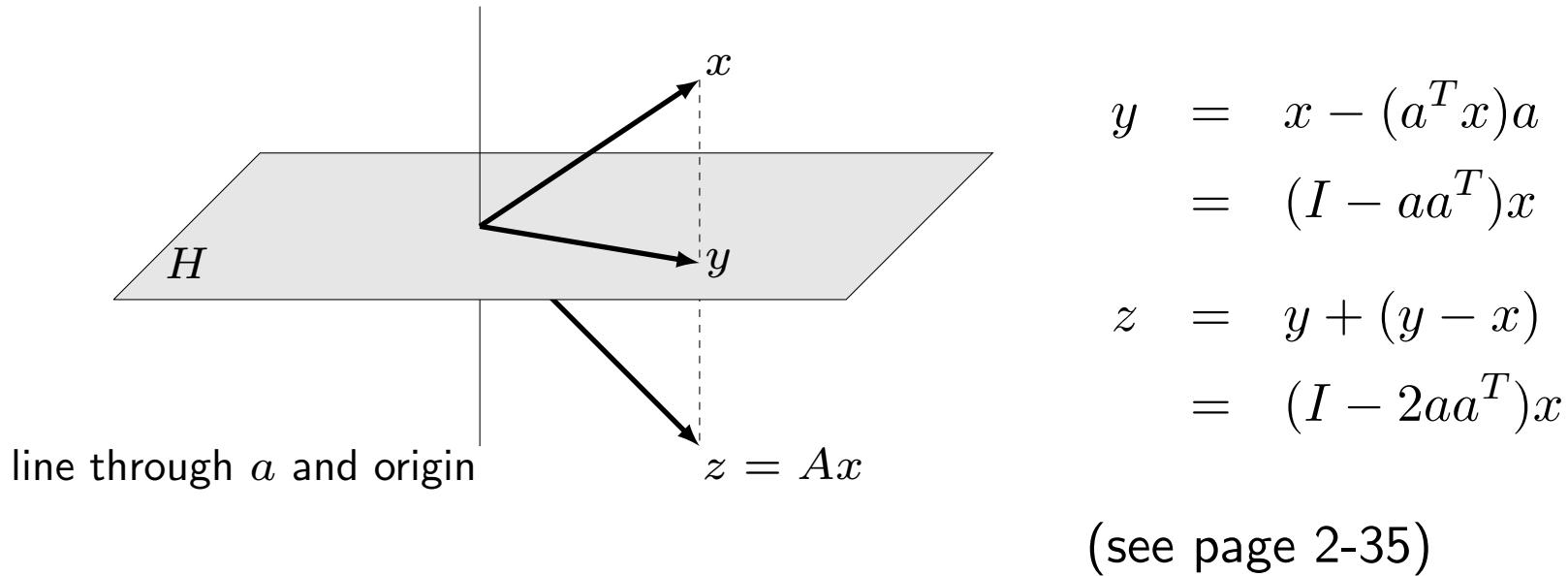
# Reflector

$$A = I - 2aa^T \quad \text{with } a \text{ a unit-norm vector } (\|a\| = 1)$$

- a reflector matrix is symmetric and orthogonal

$$A^T A = (I - 2aa^T)(I - 2aa^T) = I - 4aa^T + 4aa^T aa^T = I$$

- $Ax$  is reflection of  $x$  through the hyperplane  $H = \{u \mid a^T u = 0\}$



## Product of orthogonal matrices

if  $A_1, \dots, A_k$  are orthogonal matrices and of equal size, then the product

$$A = A_1 A_2 \cdots A_k$$

is orthogonal:

$$\begin{aligned} A^T A &= (A_1 A_2 \cdots A_k)^T (A_1 A_2 \cdots A_k) \\ &= A_k^T \cdots A_2^T A_1^T A_1 A_2 \cdots A_k \\ &= I \end{aligned}$$

## Linear equation with orthogonal matrix

linear equation with orthogonal coefficient matrix  $A$  of size  $n \times n$

$$Ax = b$$

solution is

$$x = A^{-1}b = A^T b$$

- can be computed in  $2n^2$  flops by matrix-vector multiplication
- cost is less than order  $n^2$  if  $A$  has special properties; for example,

permutation matrix: 0 flops

reflector (given  $a$ ): order  $n$  flops

plane rotation: order 1 flops

# Outline

- matrices with orthonormal columns
- orthogonal matrices
- **tall matrices with orthonormal columns**
- complex matrices with orthonormal columns

## Tall matrix with orthonormal columns

suppose  $A \in \mathbf{R}^{m \times n}$  is tall ( $m > n$ ) and has orthonormal columns

- $A^T$  is a left inverse of  $A$ :

$$A^T A = I$$

- $A$  has no right inverse; in particular

$$AA^T \neq I$$

on the next pages, we give a geometric interpretation to the matrix  $AA^T$

# Range

- the *span* of a set of vectors is the set of all their linear combinations:

$$\text{span}(a_1, a_2, \dots, a_n) = \{x_1a_1 + x_2a_2 + \cdots + x_na_n \mid x \in \mathbf{R}^n\}$$

- the *range* of a matrix  $A \in \mathbf{R}^{m \times n}$  is the span of its column vectors:

$$\text{range}(A) = \{Ax \mid x \in \mathbf{R}^n\}$$

## Example

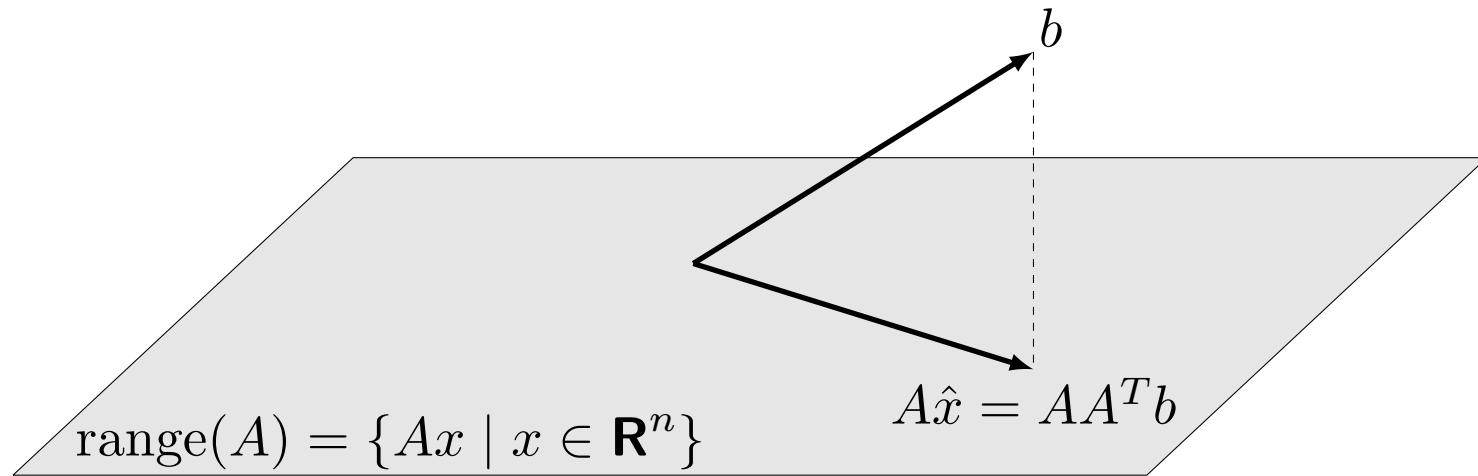
$$\begin{aligned} \text{range}\left(\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 2 \\ 0 & -1 & -1 \end{bmatrix}\right) &= \left\{ \begin{bmatrix} x_1 + x_3 \\ x_1 + x_2 + 2x_3 \\ -x_2 - x_3 \end{bmatrix} \mid x_1, x_2, x_3 \in \mathbf{R} \right\} \\ &= \{(u, u+v, -v) \mid u, v \in \mathbf{R}\} \end{aligned}$$

## Projection on range of matrix with orthonormal columns

suppose  $A \in \mathbf{R}^{m \times n}$  has orthonormal columns; we'll show that the vector

$$AA^T b$$

is the orthogonal projection of an  $m$ -vector  $b$  on  $\text{range}(A)$



- $\hat{x} = A^T b$  satisfies  $\|A\hat{x} - b\| < \|Ax - b\|$  for all  $x$
- this extends the result on page 2-11 (where  $A = (1/\|a\|)a$ )

## Proof

the squared distance of  $b$  to an arbitrary point  $Ax$  in  $\text{range}(A)$  is

$$\begin{aligned}\|Ax - b\|^2 &= \|A(x - \hat{x}) + A\hat{x} - b\|^2 \quad (\text{where } \hat{x} = A^T b) \\ &= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 + 2(x - \hat{x})^T A^T (A\hat{x} - b) \\ &= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 \\ &= \|x - \hat{x}\|^2 + \|A\hat{x} - b\|^2 \\ &\geq \|A\hat{x} - b\|^2\end{aligned}$$

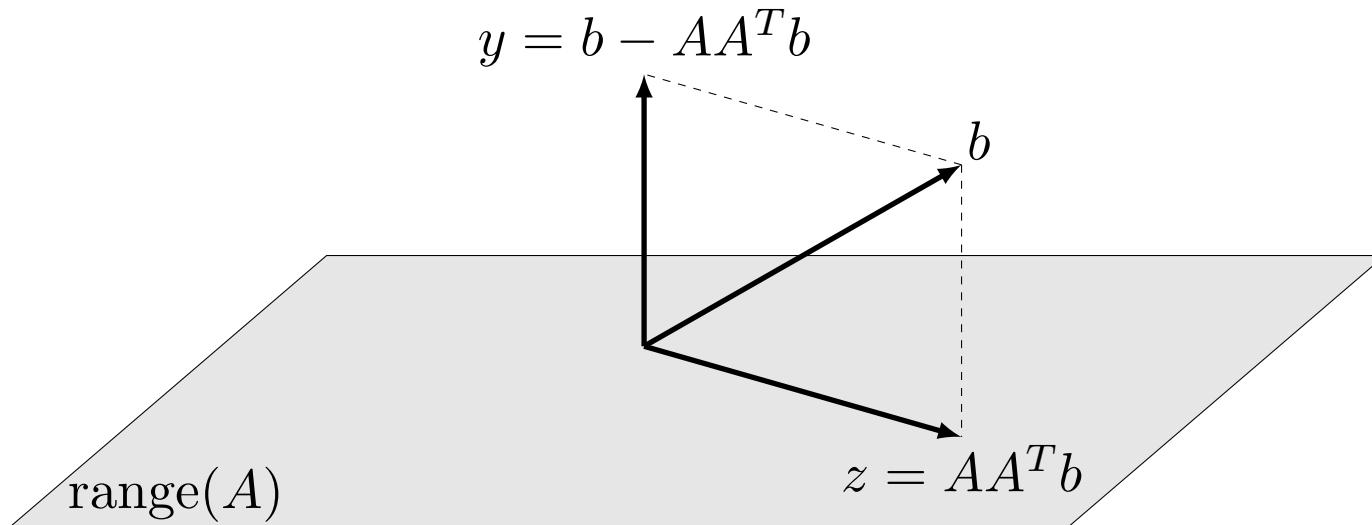
with equality only if  $x = \hat{x}$

- line 3 follows because  $A^T(A\hat{x} - b) = \hat{x} - A^T b = 0$
- line 4 follows from  $A^T A = I$

# Orthogonal decomposition

the vector  $b$  is decomposed as a sum  $b = z + y$  with

$$z \in \text{range}(A), \quad y \perp \text{range}(A)$$



such a decomposition exists and is unique for every  $b$

$$b = Ax + y, \quad A^T y = 0 \quad \iff \quad A^T b = x, \quad y = b - AA^T b$$

# Outline

- matrices with orthonormal columns
- orthogonal matrices
- tall matrices with orthonormal columns
- **complex matrices with orthonormal columns**

## Gram matrix

$A \in \mathbf{C}^{m \times n}$  has orthonormal columns if its Gram matrix is the identity:

$$\begin{aligned} A^H A &= [a_1 \ a_2 \ \cdots \ a_n]^H [a_1 \ a_2 \ \cdots \ a_n] \\ &= \begin{bmatrix} a_1^H a_1 & a_1^H a_2 & \cdots & a_1^H a_n \\ a_2^H a_1 & a_2^H a_2 & \cdots & a_2^H a_n \\ \vdots & \vdots & & \vdots \\ a_n^H a_1 & a_n^H a_2 & \cdots & a_n^H a_n \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \end{aligned}$$

- columns have unit norm:  $\|a_i\|^2 = a_i^H a_i = 1$
- columns are mutually orthogonal:  $a_i^H a_j = 0$  for  $i \neq j$

# Unitary matrix

## Unitary matrix

a *square* complex matrix with orthonormal columns is called *unitary*

## Inverse

$$\left. \begin{array}{l} A^H A = I \\ A \text{ is square} \end{array} \right\} \implies A A^H = I$$

- a unitary matrix is nonsingular with inverse  $A^H$
- if  $A$  is unitary, then  $A^H$  is unitary

## Discrete Fourier transform matrix

recall definition from page 3-33 (with  $\omega = e^{2\pi j/n}$  and  $j = \sqrt{-1}$ )

$$W = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

the matrix  $(1/\sqrt{n})W$  is unitary (proof on next page):

$$\frac{1}{n}W^H W = \frac{1}{n}WW^H = I$$

- inverse of  $W$  is  $W^{-1} = (1/n)W^H$
- inverse discrete Fourier transform of  $n$ -vector  $x$  is  $W^{-1}x = (1/n)W^Hx$

## Gram matrix of DFT matrix

we show that  $W^H W = nI$

- conjugate transpose of  $W$  is

$$W^H = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}$$

- $i, j$  element of Gram matrix is

$$(W^H W)_{ij} = 1 + \omega^{i-j} + \omega^{2(i-j)} + \cdots + \omega^{(n-1)(i-j)}$$

$$(W^H W)_{ii} = n, \quad (W^H W)_{ij} = \frac{\omega^{n(i-j)} - 1}{\omega^{i-j} - 1} = 0 \quad \text{if } i \neq j$$

(last step follows from  $\omega^n = 1$ )

## 6. QR factorization

- triangular matrices
- QR factorization
- Gram-Schmidt algorithm
- Householder algorithm

## Triangular matrix

a square matrix  $A$  is **lower triangular** if  $A_{ij} = 0$  for  $j > i$

$$A = \begin{bmatrix} A_{11} & 0 & \cdots & 0 & 0 \\ A_{21} & A_{22} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ A_{n-1,1} & A_{n-1,2} & \cdots & A_{n-1,n-1} & 0 \\ A_{n1} & A_{n2} & \cdots & A_{n,n-1} & A_{nn} \end{bmatrix}$$

$A$  is **upper triangular** if  $A_{ij} = 0$  for  $j < i$  (transpose is lower triangular)

a triangular matrix is **unit** upper/lower triangular if  $A_{ii} = 1$  for all  $i$

# Forward substitution

solve  $Ax = b$  when  $A$  is lower triangular with nonzero diagonal elements

## Algorithm

$$x_1 = b_1/A_{11}$$

$$x_2 = (b_2 - A_{21}x_1)/A_{22}$$

$$x_3 = (b_3 - A_{31}x_1 - A_{32}x_2)/A_{33}$$

$$\vdots$$

$$x_n = (b_n - A_{n1}x_1 - A_{n2}x_2 - \cdots - A_{n,n-1}x_{n-1})/A_{nn}$$

**Complexity:**  $1 + 3 + 5 + \cdots + (2n - 1) = n^2$  flops

# Back substitution

solve  $Ax = b$  when  $A$  is upper triangular with nonzero diagonal elements

## Algorithm

$$\begin{aligned}x_n &= b_n / A_{nn} \\x_{n-1} &= (b_{n-1} - A_{n-1,n}x_n) / A_{n-1,n-1} \\x_{n-2} &= (b_{n-2} - A_{n-2,n-1}x_{n-1} - A_{n-2,n}x_n) / A_{n-2,n-2} \\\vdots \\x_1 &= (b_1 - A_{12}x_2 - A_{13}x_3 - \cdots - A_{1n}x_n) / A_{11}\end{aligned}$$

**Complexity:**  $n^2$  flops

## Inverse of a triangular matrix

a triangular matrix  $A$  with nonzero diagonal elements is nonsingular:

$$Ax = 0 \implies x = 0$$

this follows from forward or back substitution applied to equation  $Ax = 0$

- inverse of  $A$  can be computed by solving  $AX = I$  column by column

$$A [ \begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} ] = [ \begin{array}{cccc} e_1 & e_2 & \cdots & e_n \end{array} ] \quad (x_i \text{ is column } i \text{ of } X)$$

- inverse of lower triangular matrix is lower triangular
- inverse of upper triangular matrix is upper triangular
- complexity of computing inverse of  $n \times n$  triangular matrix is

$$n^2 + (n-1)^2 + \cdots + 1 \approx \frac{1}{3}n^3 \text{ flops}$$

# Outline

- triangular matrices
- **QR factorization**
- Gram-Schmidt algorithm
- Householder algorithm

## QR factorization

if  $A \in \mathbb{R}^{m \times n}$  has linearly independent columns then it can be factored as

$$A = [ q_1 \quad q_2 \quad \cdots \quad q_n ] \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ 0 & R_{22} & \cdots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{nn} \end{bmatrix}$$

- vectors  $q_1, \dots, q_n$  are orthonormal  $m$ -vectors:

$$\|q_i\| = 1, \quad q_i^T q_j = 0 \quad \text{if } i \neq j$$

- diagonal elements  $R_{ii}$  are nonzero
- if  $R_{ii} < 0$ , we can switch the signs of  $R_{ii}, \dots, R_{in}$ , and the vector  $q_i$
- most definitions require  $R_{ii} > 0$ ; this makes  $Q$  and  $R$  unique

# QR factorization in matrix notation

if  $A \in \mathbf{R}^{m \times n}$  has linearly independent columns then it can be factored as

$$A = QR$$

## Q-factor

- $Q$  is  $m \times n$  with orthonormal columns ( $Q^T Q = I$ )
- if  $A$  is square ( $m = n$ ), then  $Q$  is orthogonal ( $Q^T Q = QQ^T = I$ )

## R-factor

- $R$  is  $n \times n$ , upper triangular, with nonzero diagonal elements
- $R$  is nonsingular (diagonal elements are nonzero)

## Example

$$\begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix} = \begin{bmatrix} -1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & 2 & 8 \\ 0 & 0 & 4 \end{bmatrix}$$
$$= [q_1 \quad q_2 \quad q_3] \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}$$
$$= QR$$

# Applications

in the following lectures, we will use the QR factorization to solve

- linear equations
- least squares problems
- constrained least squares problems

here, we show that it gives useful simple formulas for

- the pseudo-inverse of a matrix with linearly independent columns
- the inverse of a nonsingular matrix
- projection on the range of a matrix with linearly independent columns

## QR factorization and (pseudo-)inverse

pseudo-inverse of matrix  $A$  with linearly independent columns (page 4-23)

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

- pseudo-inverse in terms of QR factors of  $A$ :

$$\begin{aligned} A^\dagger &= ((QR)^T(QR))^{-1}(QR)^T \\ &= (R^T Q^T QR)^{-1} R^T Q^T \\ &= (R^T R)^{-1} R^T Q^T \quad (Q^T Q = I) \\ &= R^{-1} R^{-T} R^T Q^T \quad (R \text{ is nonsingular}) \\ &= R^{-1} Q^T \end{aligned}$$

- for square nonsingular  $A$  this is the inverse:

$$A^{-1} = (QR)^{-1} = R^{-1} Q^T$$

# Range

recall definition of range of a matrix  $A \in \mathbf{R}^{m \times n}$  (page 5-14):

$$\text{range}(A) = \{Ax \mid x \in \mathbf{R}^n\}$$

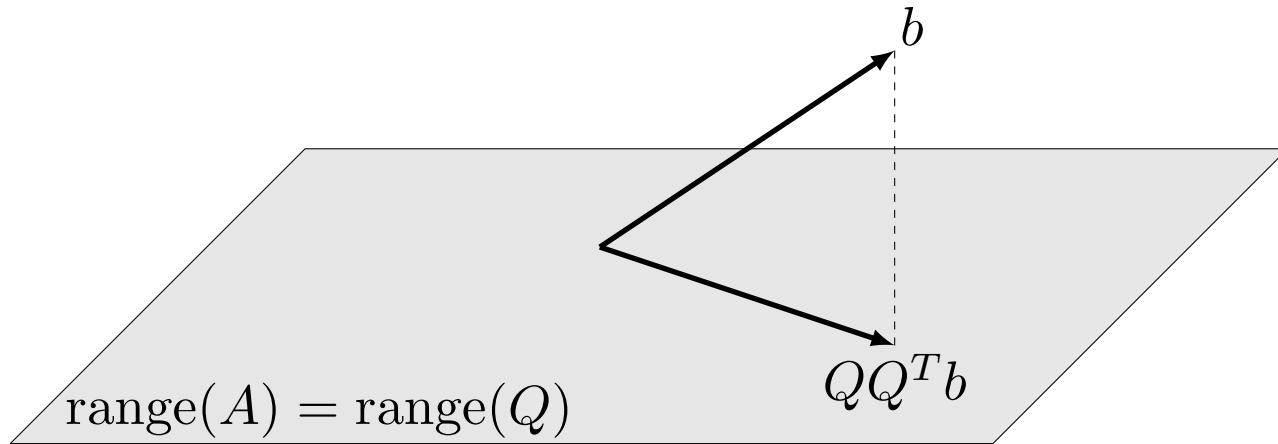
suppose  $A$  has linearly independent columns with QR factors  $Q, R$

- $Q$  has the same range as  $A$ :

$$\begin{aligned} y \in \text{range}(A) &\iff y = Ax \text{ for some } x \\ &\iff y = Qz \text{ for some } z \quad (\text{take } z = Rx) \\ &\iff y \in \text{range}(Q) \end{aligned}$$

- columns of  $Q$  are orthonormal and have the same span as columns of  $A$

## Projection on range



(from p. 5-15) projection of  $b$  on  $\text{range}(Q)$  if  $Q$  has orthonormal columns:

$$QQ^T b$$

if  $Q$  is the Q-factor of  $A$ , this is also the projection on  $\text{range}(A)$

## QR factorization of complex matrices

if  $A \in \mathbf{C}^{m \times n}$  has linearly independent columns then it can be factored as

$$A = QR$$

- $Q \in \mathbf{C}^{m \times n}$  has orthonormal columns ( $Q^H Q = I$ )
- $R \in \mathbf{C}^{n \times n}$  is upper triangular with real nonzero diagonal elements
- most definitions choose diagonal elements  $R_{ii}$  to be positive
- in the rest of the lecture we assume  $A$  is real

# Algorithms for QR factorization

## Gram-Schmidt algorithm (page 6-15)

- complexity is  $2mn^2$  flops
- not recommended in practice (sensitive to rounding errors)

## Modified Gram-Schmidt algorithm

- complexity is  $2mn^2$  flops
- better numerical properties

## Householder algorithm (page 6-25)

- complexity is  $2mn^2 - (2/3)n^3$  flops
- represents  $Q$  as a product of elementary orthogonal matrices
- the most widely used algorithm (`qr` in MATLAB)

in the rest of the course we will take  $2mn^2$  for the complexity

# Outline

- triangular matrices
- QR factorization
- **Gram-Schmidt algorithm**
- Householder algorithm

## Gram-Schmidt algorithm

Gram-Schmidt QR algorithm computes  $Q$  and  $R$  column by column

- after  $k$  steps we have a partial QR factorization

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_k \end{bmatrix} = \begin{bmatrix} q_1 & q_2 & \cdots & q_k \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1k} \\ 0 & R_{22} & \cdots & R_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{kk} \end{bmatrix}$$

- columns  $q_1, \dots, q_k$  are orthonormal;  $R_{11}, R_{22}, \dots, R_{kk} > 0$
- columns  $q_1, \dots, q_k$  have the same span as  $a_1, \dots, a_k$  (see page 6-11)

## Computing column $k$

suppose we have completed the factorization for the first  $k - 1$  columns

- column  $k$  of the equation  $A = QR$  reads

$$a_k = R_{1k}q_1 + R_{2k}q_2 + \cdots + R_{k-1,k}q_{k-1} + R_{kk}q_k$$

- regardless of the choice of  $R_{1k}, \dots, R_{k-1,k}$ , the vector

$$\tilde{q}_k = a_k - R_{1k}q_1 - R_{2k}q_2 - \cdots - R_{k-1,k}q_{k-1}$$

will be nonzero:  $a_1, a_2, \dots, a_k$  are linearly independent and therefore

$$a_k \notin \text{span}\{a_1, \dots, a_{k-1}\} = \text{span}\{q_1, \dots, q_{k-1}\}$$

- $q_k$  is  $\tilde{q}_k$  normalized: choose  $R_{kk} = \|\tilde{q}_k\|$  and  $q_k = (1/R_{kk})\tilde{q}_k$
- $\tilde{q}_k$  and  $q_k$  are orthogonal to  $q_1, \dots, q_{k-1}$  if we choose

$$R_{1k} = q_1^T a_k, \quad R_{2k} = q_2^T a_k, \quad \dots, \quad R_{k-1,k} = q_{k-1}^T a_k$$

# Gram-Schmidt algorithm

**Given:**  $m \times n$  matrix  $A$  with linearly independent columns  $a_1, \dots, a_n$

## Algorithm

for  $k = 1$  to  $n$

$$R_{1k} = q_1^T a_k$$

$$R_{2k} = q_2^T a_k$$

⋮

$$R_{k-1,k} = q_{k-1}^T a_k$$

$$\tilde{q}_k = a_k - (R_{1k}q_1 + R_{2k}q_2 + \cdots + R_{k-1,k}q_{k-1})$$

$$R_{kk} = \|\tilde{q}_k\|$$

$$q_k = \frac{1}{R_{kk}}\tilde{q}_k$$

## Example

example on page 6-8:

$$\begin{aligned} \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} &= \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix} \\ &= \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix} \end{aligned}$$

**First column of  $Q$  and  $R$**

$$\tilde{q}_1 = a_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \quad R_{11} = \|\tilde{q}_1\| = 2, \quad q_1 = \frac{1}{R_{11}}\tilde{q}_1 = \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix}$$

## Example

### Second column of $Q$ and $R$

- compute  $R_{12} = q_1^T a_2 = 4$
- compute

$$\tilde{q}_2 = a_2 - R_{12}q_1 = \begin{bmatrix} -1 \\ 3 \\ -1 \\ 3 \end{bmatrix} - 4 \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

- normalize to get

$$R_{22} = \|\tilde{q}_2\| = 2, \quad q_2 = \frac{1}{R_{22}}\tilde{q}_2 = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

# Example

## Third column of $Q$ and $R$

- compute  $R_{13} = q_1^T a_3 = 2$  and  $R_{23} = q_2^T a_3 = 8$
- compute

$$\tilde{q}_3 = a_3 - R_{13}q_1 - R_{23}q_2 = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix} - 2 \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} - 8 \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 2 \\ 2 \end{bmatrix}$$

- normalize to get

$$R_{33} = \|\tilde{q}_3\| = 4, \quad q_3 = \frac{1}{R_{33}}\tilde{q}_3 = \begin{bmatrix} -1/2 \\ -1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

# Example

Final result

$$\begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix} = [q_1 \quad q_2 \quad q_3] \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}$$
$$= \begin{bmatrix} -1/2 & 1/2 & -1/2 \\ 1/2 & 1/2 & -1/2 \\ -1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & 2 & 8 \\ 0 & 0 & 4 \end{bmatrix}$$

# Complexity

**Complexity of cycle  $k$**  (algorithm on page 6-17)

- $k - 1$  inner products with  $a_k$ :  $(k - 1)(2m - 1)$  flops
- computation of  $\tilde{q}_k$ :  $2(k - 1)m$  flops
- computing  $R_{kk}$  and  $q_k$ :  $3m$  flops

total for cycle  $k$ :  $(4m - 1)(k - 1) + 3m$  flops

**Complexity** for  $m \times n$  factorization:

$$\begin{aligned} \sum_{k=1}^n ((4m - 1)(k - 1) + 3m) &= (4m - 1) \frac{n(n - 1)}{2} + 3mn \\ &\approx 2mn^2 \text{ flops} \end{aligned}$$

# Numerical experiment

- we use the following MATLAB code

```
[m, n] = size(A);  
Q = zeros(m,n);  
R = zeros(n,n);  
for k = 1:n  
    R(1:k-1,k) = Q(:,1:k-1)' * A(:,k);  
    v = A(:,k) - Q(:,1:k-1) * R(1:k-1,k);  
    R(k,k) = norm(v);  
    Q(:,k) = v / R(k,k);  
end;
```

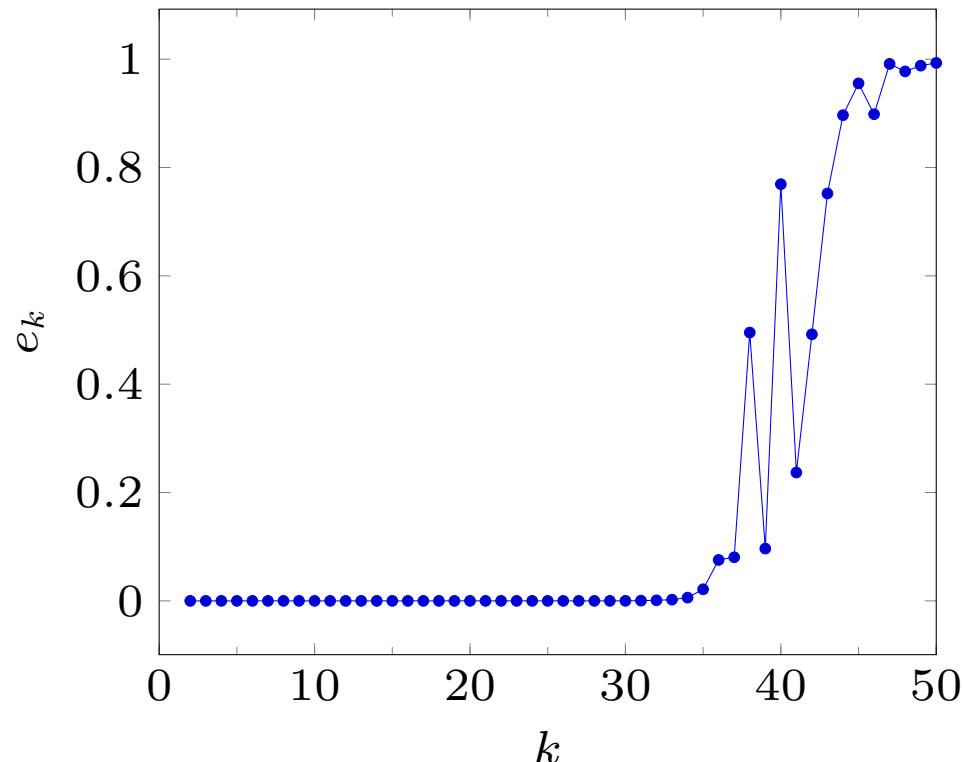
- we apply this to a square matrix  $A$  of size  $m = n = 50$
- $A$  is constructed as  $A = USV$  with  $U, V$  orthogonal,  $S$  diagonal with

$$S_{ii} = 10^{-10(i-1)/(n-1)}, \quad i = 1, \dots, n$$

## Numerical experiment

plot shows deviation from orthogonality between  $q_k$  and previous columns

$$e_k = \max_{1 \leq i < k} |q_i^T q_k|, \quad k = 2, \dots, n$$



loss of orthogonality is due to rounding error

# Outline

- triangular matrices
- QR factorization
- Gram-Schmidt algorithm
- **Householder algorithm**

## Householder algorithm

- the most widely used algorithm for QR factorization (`qr` in MATLAB)
- less sensitive to rounding error than Gram-Schmidt algorithm
- computes a ‘full’ QR factorization

$$A = [ Q \quad \tilde{Q} ] \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad [ Q \quad \tilde{Q} ] \text{ orthogonal}$$

- the full Q-factor is constructed as a product of orthogonal matrices

$$[ Q \quad \tilde{Q} ] = H_1 H_2 \cdots H_n$$

each  $H_i$  is an  $m \times m$  symmetric, orthogonal ‘reflector’ (page 5-10)

# Reflector

$$H = I - 2vv^T \quad \text{with } \|v\| = 1$$

- $Hx$  is reflection of  $x$  through hyperplane  $\{z \mid v^T z = 0\}$  (see page 5-10)
- $H$  is symmetric
- $H$  is orthogonal
- matrix-vector product  $Hx$  can be computed efficiently as

$$Hx = x - 2(v^T x)x$$

complexity is  $4p$  flops if  $v$  and  $x$  have length  $p$

## Reflection to multiple of unit vector

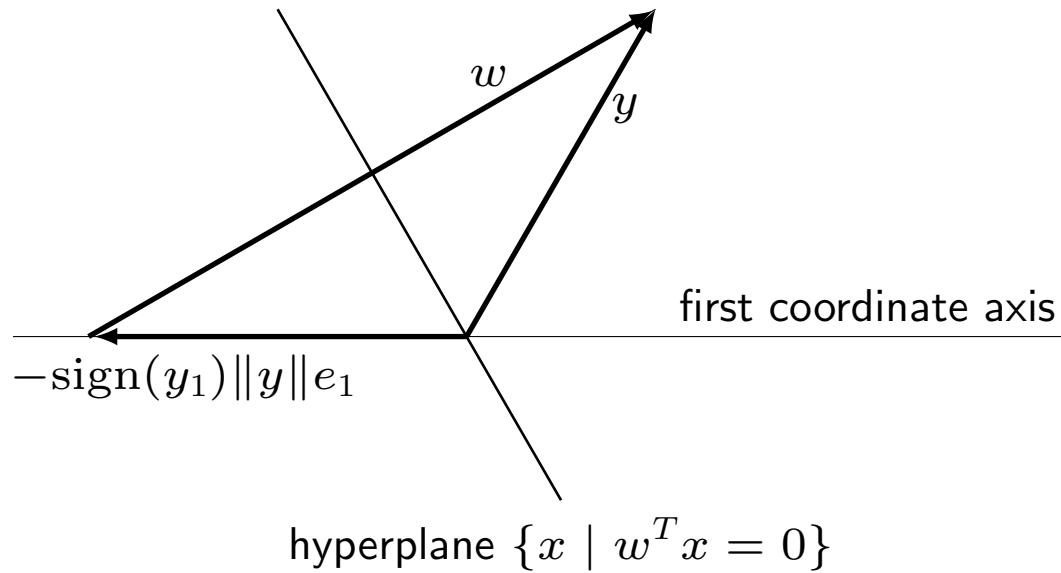
given nonzero  $p$ -vector  $y = (y_1, y_2, \dots, y_p)$ , define

$$w = \begin{bmatrix} y_1 + \text{sign}(y_1)\|y\| \\ y_2 \\ \vdots \\ y_p \end{bmatrix}, \quad v = \frac{1}{\|w\|}w$$

- we take  $\text{sign}(0) = 1$
- vector  $w$  satisfies  $\|w\|^2 = 2(w^T y) = 2\|y\|(\|y\| + |y_1|)$
- reflector  $H = I - 2vv^T$  maps  $y$  to multiple of  $e_1 = (1, 0, \dots, 0)$ :

$$Hy = y - \frac{2(w^T y)}{\|w\|^2}w = y - w = -\text{sign}(y_1)\|y\|e_1$$

# Geometry



reflection through the hyperplane  $\{x \mid w^T x = 0\}$  with normal vector

$$w = y + \text{sign}(y_1) \|y\| e_1$$

maps  $y$  to the vector  $-\text{sign}(y_1) \|y\| e_1$

# Householder triangularization

- computes reflectors  $H_1, \dots, H_n$  that reduce  $A$  to triangular form:

$$H_n H_{n-1} \cdots H_1 A = \begin{bmatrix} R \\ 0 \end{bmatrix}$$

- after step  $k$ , the matrix  $H_k \cdots H_1 A$  has the following structure:

$$H_k H_{k-1} \cdots H_1 A =$$

The diagram shows a square matrix structure. It is divided into four quadrants by lines from the top-left corner to the bottom-right corner. The top-right quadrant is shaded gray and has dimensions  $k$  by  $m - k$ . The bottom-left quadrant is white and has dimensions  $k$  by  $n - k$ . The other two quadrants are also shaded gray. Arrows indicate the dimensions:  $k$  vertically on the right,  $m - k$  vertically on the bottom,  $k$  horizontally on the left, and  $n - k$  horizontally on the bottom.

(elements in positions  $i, j$  for  $i > j$  and  $j \leq k$  are zero)

## Householder algorithm

the following algorithm overwrites  $A$  with  $\begin{bmatrix} R \\ 0 \end{bmatrix}$

**Algorithm:** for  $k = 1$  to  $n$ ,

1. define  $y = A_{k:m,k}$  and compute  $(m - k + 1)$ -vector  $v_k$ :

$$w = y + \text{sign}(y_1)\|y\|e_1, \quad v_k = \frac{1}{\|w\|}w$$

2. multiply  $A_{k:m,k:n}$  with reflector  $I - 2v_kv_k^T$ :

$$A_{k:m,k:n} := A_{k:m,k:n} - 2v_k(v_k^T A_{k:m,k:n})$$

(see page 77 in textbook for ‘slice’ notation for submatrices)

## Comments

- in step 2 we multiply  $A_{k:m,k:n}$  with the reflector  $I - 2v_k v_k^T$ :

$$(I - 2v_k v_k^T) A_{k:m,k:n} = A_{k:m,k:n} - 2v_k (v_k^T A_{k:m,k:n})$$

- this is equivalent to multiplying  $A$  with  $m \times m$  reflector

$$H_k = \begin{bmatrix} I & 0 \\ 0 & I - 2v_k v_k^T \end{bmatrix} = I - 2 \begin{bmatrix} 0 \\ v_k \end{bmatrix} \begin{bmatrix} 0 \\ v_k \end{bmatrix}^T$$

- algorithm overwrites  $A$  with

$$\begin{bmatrix} R \\ 0 \end{bmatrix}$$

and returns the vectors  $v_1, \dots, v_n$ , with  $v_k$  of length  $m - k + 1$

## Example

example on page 6-8:

$$A = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 3 & 3 \\ -1 & -1 & 5 \\ 1 & 3 & 7 \end{bmatrix} = H_1 H_2 H_3 \begin{bmatrix} R \\ 0 \end{bmatrix}$$

we compute reflectors  $H_1, H_2, H_3$  that triangularize  $A$ :

$$H_3 H_2 H_1 A = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}$$

# Example

## First column of $R$

- compute reflector that maps first column of  $A$  to multiple of  $e_1$ :

$$y = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \quad w = y - \|y\|e_1 = \begin{bmatrix} -3 \\ 1 \\ -1 \\ 1 \end{bmatrix}, \quad v_1 = \frac{1}{\|w\|}w = \frac{1}{2\sqrt{3}} \begin{bmatrix} -3 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

- overwrite  $A$  with product of  $I - 2v_1v_1^T$  and  $A$

$$A := (I - 2v_1v_1^T)A = \begin{bmatrix} 2 & 4 & 2 \\ 0 & 4/3 & 8/3 \\ 0 & 2/3 & 16/3 \\ 0 & 4/3 & 20/3 \end{bmatrix}$$

# Example

## Second column of $R$

- compute reflector that maps  $A_{2:4,2}$  to multiple of  $e_1$ :

$$y = \begin{bmatrix} 4/3 \\ 2/3 \\ 4/3 \end{bmatrix}, \quad w = y + \|y\|e_1 = \begin{bmatrix} 10/3 \\ 2/3 \\ 4/3 \end{bmatrix}, \quad v_2 = \frac{1}{\|w\|}w = \frac{1}{\sqrt{30}} \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$$

- overwrite  $A_{2:4,2:3}$  with product of  $I - 2v_2v_2^T$  and  $A_{2:4,2:3}$ :

$$A := \begin{bmatrix} 1 & 0 \\ 0 & I - 2v_2v_2^T \end{bmatrix} A = \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & 16/5 \\ 0 & 0 & 12/5 \end{bmatrix}$$

# Example

## Third column of $R$

- compute reflector that maps  $A_{3:4,3}$  to multiple of  $e_1$ :

$$y = \begin{bmatrix} 16/5 \\ 12/5 \end{bmatrix}, \quad w = y + \|y\|e_1 = \begin{bmatrix} 36/5 \\ 12/5 \end{bmatrix}, \quad v_3 = \frac{1}{\|w\|}w = \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

- overwrite  $A_{3:4,3}$  with product of  $I - 2v_3v_3^T$  and  $A_{3:4,3}$ :

$$A := \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} A = \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}$$

# Example

## Final result

$$\begin{aligned}
 H_3H_2H_1A &= \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & I - 2v_2v_2^T \end{bmatrix} (I - 2v_1v_1^T)A \\
 &= \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & I - 2v_2v_2^T \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & 4/3 & 8/3 \\ 0 & 2/3 & 16/3 \\ 0 & 4/3 & 20/3 \end{bmatrix} \\
 &= \begin{bmatrix} I & 0 \\ 0 & I - 2v_3v_3^T \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & 16/5 \\ 0 & 0 & 12/5 \end{bmatrix} \\
 &= \begin{bmatrix} 2 & 4 & 2 \\ 0 & -2 & -8 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}
 \end{aligned}$$

# Complexity

**Complexity in cycle  $k$**  (algorithm on page 6-30): the dominant terms are

- $(2(m - k + 1) - 1)(n - k + 1)$  flops for product  $v_k^T(A_{k:m,k:n})$
- $(m - k + 1)(n - k + 1)$  flops for outer product with  $v_k$
- $(m - k + 1)(n - k + 1)$  flops for subtraction from  $A_{k:m,k:n}$

sum is roughly  $4(m - k + 1)(n - k + 1)$  flops

**Total** for computing  $R$  and vectors  $v_1, \dots, v_n$ :

$$\begin{aligned} \sum_{k=1}^n 4(m - k + 1)(n - k + 1) &\approx \int_0^n 4(m - t)(n - t)dt \\ &= 2mn^2 - \frac{2}{3}n^3 \quad \text{flops} \end{aligned}$$

## Q-factor

the Householder algorithm returns the vectors  $v_1, \dots, v_n$  that define

$$[ Q \quad \tilde{Q} ] = H_1 H_2 \cdots H_n$$

- usually there is no need to compute the matrix  $[ Q \quad \tilde{Q} ]$  explicitly
- the vectors  $v_1, \dots, v_n$  are an economical representation of  $[ Q \quad \tilde{Q} ]$
- products with  $[ Q \quad \tilde{Q} ]$  or its transpose can be computed as

$$[ Q \quad \tilde{Q} ] x = H_1 H_2 \cdots H_n x$$

$$[ Q \quad \tilde{Q} ]^T y = H_n H_{n-1} \cdots H_1 y$$

## Multiplication with Q-factor

- the matrix-vector product  $H_k x$  is defined as

$$H_k x = \begin{bmatrix} I & 0 \\ 0 & I - 2v_k v_k^T \end{bmatrix} \begin{bmatrix} x_{1:k-1} \\ x_{k:m} \end{bmatrix} = \begin{bmatrix} x_{1:k-1} \\ x_{k:m} - 2(v_k^T x_{k:m}) v_k \end{bmatrix}$$

- complexity of multiplication  $H_k x$  is  $4(m - k + 1)$  flops:
- complexity of multiplication with  $H_1 H_2 \cdots H_n$  or its transpose is

$$\sum_{k=1}^n 4(m - k + 1) \approx 4mn - 2n^2 \text{ flops}$$

- roughly equal to matrix-vector product with  $m \times n$  matrix ( $2mn$  flops)

## 7. Linear equations

- QR factorization method
- factor and solve
- LU factorization

# QR factorization and inverse

## QR factorization of nonsingular matrix

every nonsingular  $A \in \mathbf{R}^{n \times n}$  has a QR factorization

$$A = QR$$

- $Q \in \mathbf{R}^{n \times n}$  is orthogonal ( $Q^T Q = QQ^T = I$ )
- $R \in \mathbf{R}^{n \times n}$  is upper triangular with positive diagonal elements

**Inverse from QR factorization:** the inverse  $A^{-1}$  can be written as

$$A^{-1} = (QR)^{-1} = R^{-1}Q^{-1} = R^{-1}Q^T$$

# Solving linear equations by QR factorization

**Algorithm:** to solve  $Ax = b$  with nonsingular  $A \in \mathbf{R}^{n \times n}$ ,

1. factor  $A$  as  $A = QR$
2. compute  $y = Q^T b$
3. solve  $Rx = y$  by back substitution

**Complexity:**  $2n^3 + 3n^2 \approx 2n^3$  flops

- QR factorization:  $2n^3$
- matrix-vector multiplication:  $2n^2$
- back substitution:  $n^2$

## Multiple right-hand sides

consider  $k$  sets of linear equations with the same coefficient matrix  $A$ :

$$Ax_1 = b_1, \quad Ax_2 = b_2, \quad \dots, \quad Ax_k = b_k$$

- can be solved in  $2n^3 + 3kn^2$  flops if we reuse the factorization  $A = QR$
- for  $k \ll n$ , cost is roughly equal to cost of solving one equation ( $2n^3$ )

**Application:** to compute  $A^{-1}$ , solve the matrix equation  $AX = I$

- equivalent to  $n$  equations

$$Rx_1 = Q^T e_1, \quad Rx_2 = Q^T e_2, \quad \dots, \quad Rx_n = Q^T e_n$$

( $x_i$  is column  $i$  of  $X$  and  $Q^T e_i$  is transpose of  $i$ th row of  $Q$ )

- complexity is  $2n^3 + n^3 = 3n^3$  (here the 2nd term  $n^3$  is not negligible)

# Outline

- QR factorization method
- **factor and solve**
- LU factorization

## Factor-solve approach

to solve  $Ax = b$ , first write  $A$  as a product of ‘simple’ matrices

$$A = A_1 A_2 \cdots A_k$$

then solve  $(A_1 A_2 \cdots A_k)x = b$  by solving  $k$  equations

$$A_1 z_1 = b, \quad A_2 z_2 = z_1, \quad \dots, \quad A_{k-1} z_{k-1} = z_{k-2}, \quad A_k x = z_{k-1}$$

## Examples

- QR factorization:  $k = 2$ ,  $A = QR$
- LU factorization (this lecture)
- Cholesky factorization (later)

## Complexity of factor-solve method

$$\#\text{flops} = f + s$$

- $f$  is cost of factoring  $A$  as  $A = A_1 A_2 \cdots A_k$  (factorization step)
- $s$  is cost of solving the  $k$  equations for  $z_1, z_2, \dots, z_{k-1}, x$  (solve step)
- usually  $f \gg s$

**Example:** solving linear equations using the QR factorization

$$f = 2n^3, \quad s = 3n^2$$

# Outline

- QR factorization method
- factor and solve
- **LU factorization**

# LU factorization

## LU factorization without pivoting

$$A = LU$$

- $L$  unit lower triangular,  $U$  upper triangular
- does not always exist (even if  $A$  is nonsingular)

## LU factorization (with row pivoting)

$$A = PLU$$

- $P$  permutation matrix,  $L$  unit lower triangular,  $U$  upper triangular
- exists if and only if  $A$  is nonsingular (see later)

**Complexity:**  $(2/3)n^3$  if  $A$  is  $n \times n$

# Solving linear equations by LU factorization

**Algorithm:** to solve  $Ax = b$  with nonsingular  $A$  of size  $n \times n$

1. factor  $A$  as  $A = PLU$  ( $(2/3)n^3$  flops)
2. solve  $(PLU)x = b$  in three steps
  - permutation:  $z_1 = P^T b$  (0 flops)
  - forward substitution: solve  $Lz_2 = z_1$  ( $n^2$  flops)
  - back substitution: solve  $Ux = z_2$  ( $n^2$  flops)

**Complexity:**  $(2/3)n^3 + 2n^2 \approx (2/3)n^3$  flops

this is the standard method for solving  $Ax = b$

## Multiple right-hand sides

two equations with the same matrix  $A$  (nonsingular and  $n \times n$ ):

$$Ax = b, \quad A\tilde{x} = \tilde{b}$$

- factor  $A$  once
- forward/back substitution to get  $x$
- forward/back substitution to get  $\tilde{x}$

complexity:  $(2/3)n^3 + 4n^2 \approx (2/3)n^3$

**Exercise:** propose an efficient method for solving

$$Ax = b, \quad A^T\tilde{x} = \tilde{b}$$

## LU factorization and inverse

suppose  $A$  is nonsingular and  $n \times n$ , with LU factorization

$$A = PLU$$

- inverse from LU factorization

$$A^{-1} = (PLU)^{-1} = U^{-1}L^{-1}P^T$$

- gives interpretation of solve step: we evaluate

$$x = A^{-1}b = U^{-1}L^{-1}P^T b$$

in three steps

$$z_1 = P^T b, \quad z_2 = L^{-1} z_1, \quad x = U^{-1} z_2$$

## Computing the inverse

solve  $AX = I$  column by column

- one LU factorization of  $A$ :  $2n^3/3$  flops
- $n$  solve steps:  $2n^3$  flops
- total:  $(8/3)n^3$  flops

slightly faster methods exist that exploit structure in right-hand side  $I$

**Conclusion:** do not solve  $Ax = b$  by multiplying  $A^{-1}$  with  $b$

# LU factorization without pivoting

$$\begin{bmatrix} A_{11} & A_{1,2:n} \\ A_{2:n,1} & A_{2:n,2:n} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{2:n,1} & L_{2:n,2:n} \end{bmatrix} \begin{bmatrix} U_{11} & U_{1,2:n} \\ 0 & U_{2:n,2:n} \end{bmatrix}$$
$$= \begin{bmatrix} U_{11} & U_{1,2:n} \\ U_{11}L_{2:n,1} & L_{2:n,1}U_{1,2:n} + L_{2:n,2:n}U_{2:n,2:n} \end{bmatrix}$$

## Recursive algorithm

- determine first row of  $U$  and first column of  $L$

$$U_{11} = A_{11}, \quad U_{1,2:n} = A_{1,2:n}, \quad L_{2:n,1} = \frac{1}{A_{11}}A_{2:n,1}$$

- factor the  $(n - 1) \times (n - 1)$ -matrix  $A_{2:n,2:n} - L_{2:n,1}U_{1,2:n}$  as

$$A_{2:n,2:n} - L_{2:n,1}U_{1,2:n} = L_{2:n,2:n}U_{2:n,2:n}$$

this is an LU factorization (without pivoting) of size  $(n - 1) \times (n - 1)$

## Example

LU factorization (without pivoting) of

$$A = \begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix}$$

write as  $A = LU$  with  $L$  unit lower triangular,  $U$  upper triangular

$$A = \begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

- first row of  $U$ , first column of  $L$ :

$$\begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 3/4 & L_{32} & 1 \end{bmatrix} \begin{bmatrix} 8 & 2 & 9 \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

- second row of  $U$ , second column of  $L$ :

$$\begin{bmatrix} 9 & 4 \\ 7 & 9 \end{bmatrix} - \begin{bmatrix} 1/2 \\ 3/4 \end{bmatrix} \begin{bmatrix} 2 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{22} & U_{23} \\ 0 & U_{33} \end{bmatrix}$$

$$\begin{bmatrix} 8 & -1/2 \\ 11/2 & 9/4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 11/16 & 1 \end{bmatrix} \begin{bmatrix} 8 & -1/2 \\ 0 & U_{33} \end{bmatrix}$$

- third row of  $U$ :  $U_{33} = 9/4 + 11/32 = 83/32$

## Conclusion

$$A = \begin{bmatrix} 8 & 2 & 9 \\ 4 & 9 & 4 \\ 6 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 3/4 & 11/16 & 1 \end{bmatrix} \begin{bmatrix} 8 & 2 & 9 \\ 0 & 8 & -1/2 \\ 0 & 0 & 83/32 \end{bmatrix}$$

## Not every nonsingular $A$ can be factored as $A = LU$

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

- first row of  $U$ , first column of  $L$ :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & L_{32} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

- second row of  $U$ , second column of  $L$ :

$$\begin{bmatrix} 0 & 2 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{32} & 1 \end{bmatrix} \begin{bmatrix} U_{22} & U_{23} \\ 0 & U_{33} \end{bmatrix}$$

$$U_{22} = 0, U_{23} = 2, L_{32} \cdot 0 = 1 ?$$

## LU factorization (with row pivoting)

if  $A$  is  $n \times n$  and nonsingular, then it can be factored as

$$A = PLU$$

$P$  is a permutation matrix,  $L$  is unit lower triangular,  $U$  is upper triangular

- not unique; there may be several possible choices for  $P$ ,  $L$ ,  $U$
- interpretation: permute the rows of  $A$  and factor  $P^T A$  as  $P^T A = LU$
- also known as *Gaussian elimination with partial pivoting* (GEPP)
- complexity:  $(2/3)n^3$  flops

we skip the details of calculating  $P$ ,  $L$ ,  $U$

## Example

$$\begin{bmatrix} 0 & 5 & 5 \\ 2 & 9 & 0 \\ 6 & 8 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 0 & 15/19 & 1 \end{bmatrix} \begin{bmatrix} 6 & 8 & 8 \\ 0 & 19/3 & -8/3 \\ 0 & 0 & 135/19 \end{bmatrix}$$

the factorization is not unique; the same matrix can be factored as

$$\begin{bmatrix} 0 & 5 & 5 \\ 2 & 9 & 0 \\ 6 & 8 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & -19/5 & 1 \end{bmatrix} \begin{bmatrix} 2 & 9 & 0 \\ 0 & 5 & 5 \\ 0 & 0 & 27 \end{bmatrix}$$

## Effect of rounding error

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

solution:

$$x_1 = \frac{-1}{1 - 10^{-5}}, \quad x_2 = \frac{1}{1 - 10^{-5}}$$

- let us solve using LU factorization for the two possible permutations:

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{or} \quad P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- we round intermediate results to four significant decimal digits

**First choice:**  $P = I$  (no pivoting)

$$\begin{bmatrix} 10^{-5} & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^5 & 1 \end{bmatrix} \begin{bmatrix} 10^{-5} & 1 \\ 0 & 1 - 10^5 \end{bmatrix}$$

- $L, U$  rounded to 4 decimal significant digits

$$L = \begin{bmatrix} 1 & 0 \\ 10^5 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 10^{-5} & 1 \\ 0 & -10^5 \end{bmatrix}$$

- forward substitution

$$\begin{bmatrix} 1 & 0 \\ 10^5 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \implies z_1 = 1, \quad z_2 = -10^5$$

- back substitution

$$\begin{bmatrix} 10^{-5} & 1 \\ 0 & -10^5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -10^5 \end{bmatrix} \implies x_1 = 0, \quad x_2 = 1$$

error in  $x_1$  is 100%

## Second choice: interchange rows

$$\begin{bmatrix} 1 & 1 \\ 10^{-5} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 10^{-5} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 - 10^{-5} \end{bmatrix}$$

- $L, U$  rounded to 4 decimal significant digits

$$L = \begin{bmatrix} 1 & 0 \\ 10^{-5} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

- forward substitution

$$\begin{bmatrix} 1 & 0 \\ 10^{-5} & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \implies z_1 = 0, \quad z_2 = 1$$

- backward substitution

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \implies x_1 = -1, \quad x_2 = 1$$

error in  $x_1, x_2$  is about  $10^{-5}$

## Conclusion

- for some choices of  $P$ , small errors in the algorithm can cause very large errors in the solution
- this is called *numerical instability*: for the first choice of  $P$ , the algorithm is unstable; for the second choice of  $P$ , it is stable
- from numerical analysis: there is a simple rule for selecting a good permutation  
(we skip the details, since we skipped the details of the factorization)

## Sparse linear equations

if  $A$  is sparse, it is usually factored as

$$A = P_1 L U P_2$$

$P_1$  and  $P_2$  are permutation matrices

- interpretation: permute rows and columns of  $A$  and factor  $\tilde{A} = P_1^T A P_2^T$

$$\tilde{A} = L U$$

- choice of  $P_1$  and  $P_2$  greatly affects the sparsity of  $L$  and  $U$ : many heuristic methods exist for selecting good permutations
- in practice: #flops  $\ll (2/3)n^3$ ; exact value is a complicated function of  $n$ , number of nonzero elements, sparsity pattern

# Conclusion

different levels of detail in understanding how linear equation solvers work

## Highest level

- $x = A \setminus b$  costs  $(2/3)n^3$
- more efficient than  $x = \text{inv}(A) * b$

**Intermediate level:** factorization step  $A = PLU$  followed by solve step

**Lowest level:** details of factorization  $A = PLU$

- for most applications, level 1 is sufficient
- in some situations (*e.g.*, multiple right-hand sides) level 2 is useful
- level 3 is important for experts who write numerical libraries

## 8. Least squares

- least squares problem
- solution of a least squares problem
- solving least squares problems

## Least squares problem

given  $A \in \mathbf{R}^{m \times n}$  and  $b \in \mathbf{R}^m$ , find vector  $x \in \mathbf{R}^n$  that minimizes

$$\|Ax - b\|^2 = \sum_{i=1}^m \left( \sum_{j=1}^n A_{ij}x_j - b_i \right)^2$$

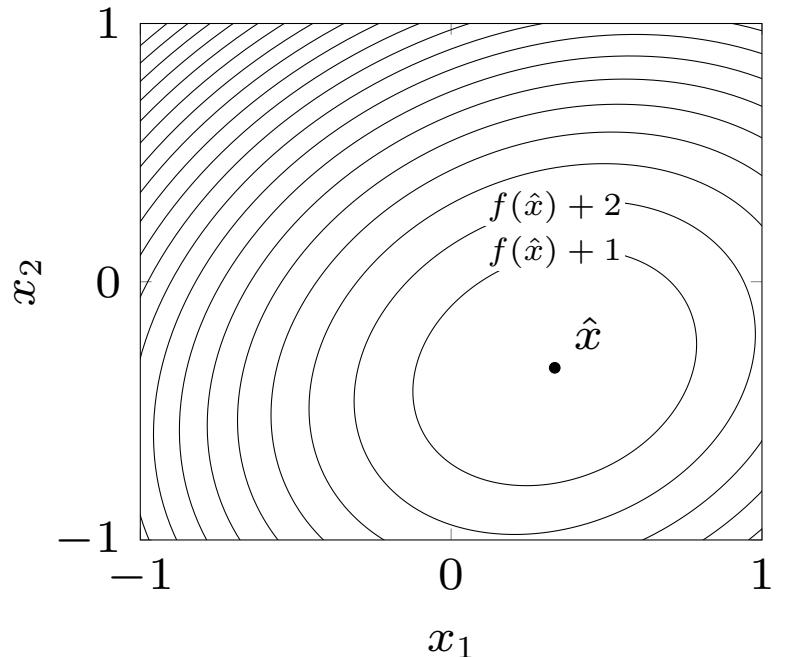
- ‘least squares’ because we minimize a sum of squares of affine functions:

$$\|Ax - b\|^2 = \sum_{i=1}^m r_i(x)^2, \quad r_i(x) = \sum_{j=1}^n A_{ij}x_j - b_i$$

- also called *linear* least squares problem
- *nonlinear* least squares problem is to minimize  $\sum_i r_i(x)^2$  with general  $r_i$

# Example

$$A = \begin{bmatrix} 2 & 0 \\ -1 & 1 \\ 0 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$



- least squares solution  $\hat{x}$  minimizes

$$f(x) = \|Ax - b\|^2 = (2x_1 - 1)^2 + (-x_1 + x_2)^2 + (2x_2 + 1)^2$$

- to find  $\hat{x}$ , set derivatives with respect to  $x_1$  and  $x_2$  equal to zero:

$$10x_1 - 2x_2 - 4 = 0, \quad -2x_1 + 10x_2 + 4 = 0$$

solution is  $\hat{x}_1 = 1/3$ ,  $\hat{x}_2 = -1/3$

## Example: polynomial approximation

fit a polynomial of degree less than  $n$

$$p(t) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1}$$

to  $m$  data points  $(t_1, y_1), \dots, (t_m, y_m)$

- least squares formulation: choose coefficients  $x_1, \dots, x_n$  that minimize

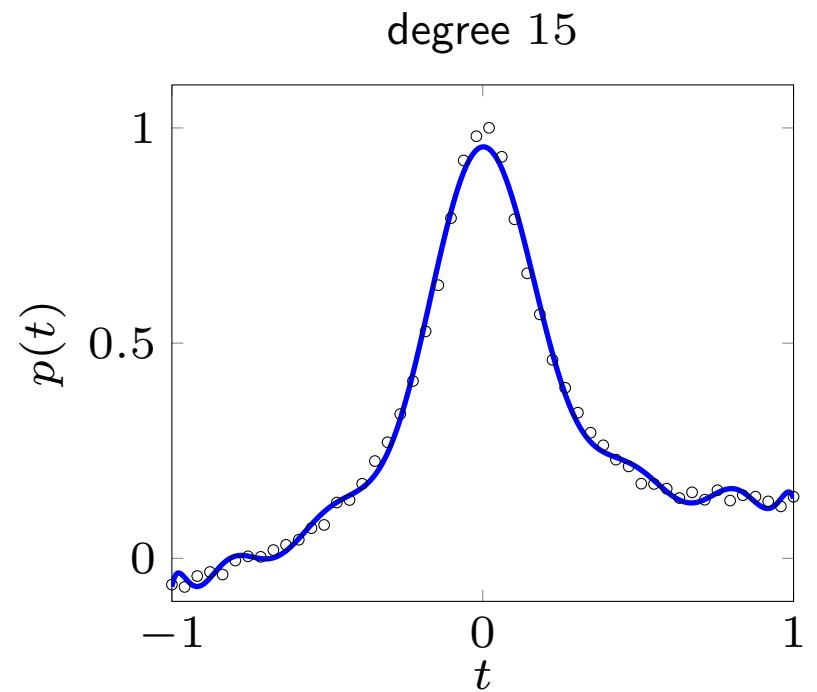
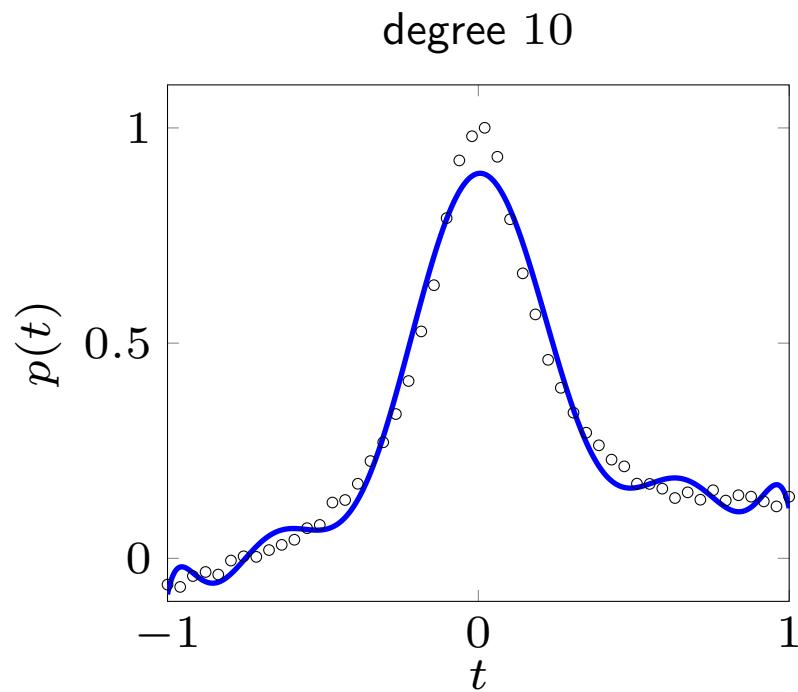
$$(p(t_1) - y_1)^2 + (p(t_2) - y_2)^2 + \cdots + (p(t_m) - y_m)^2$$

- in matrix form: minimize  $\|Ax - b\|^2$  with

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^{n-1} \\ 1 & t_2 & t_2^2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & t_m & t_m^2 & \cdots & t_m^{n-1} \end{bmatrix}, \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

# Example

$m = 50$  points, polynomials of degree 10 ( $n = 11$ ) and 15 ( $n = 16$ )



# Least squares and linear equations

any  $\hat{x}$  that satisfies

$$\|A\hat{x} - b\| \leq \|Ax - b\| \quad \text{for all } x$$

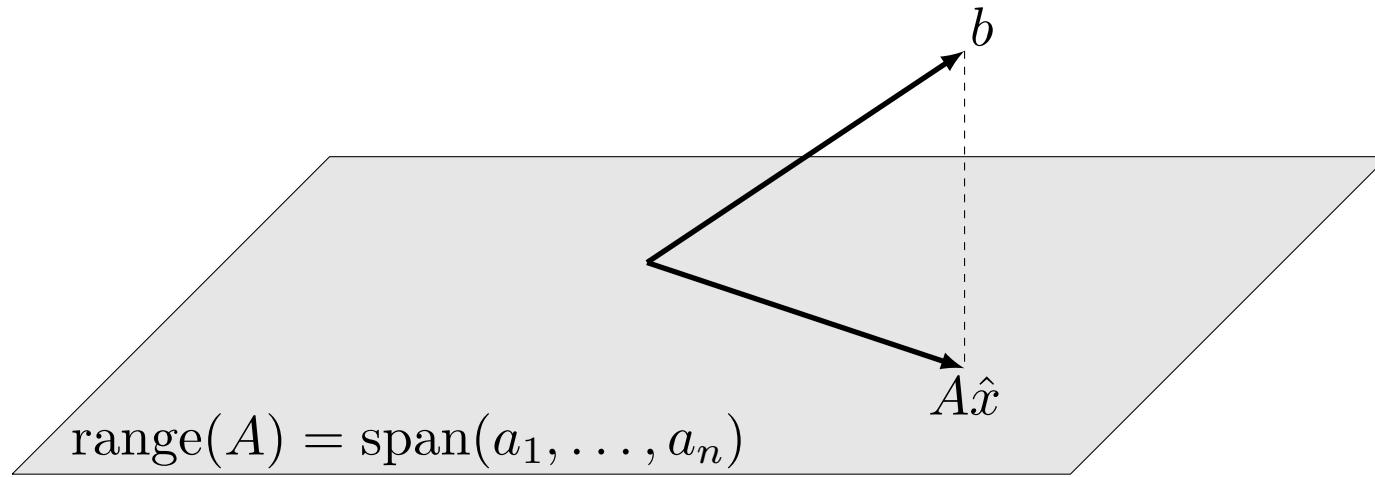
is a *solution* of the least squares problem

- $\hat{r} = b - A\hat{x}$  is the *residual vector*
- if  $\hat{r} = 0$ , then  $\hat{x}$  solves the linear equation  $Ax = b$
- if  $\hat{r} \neq 0$ , then  $\hat{x}$  is a *least squares approximate solution* of the equation
- in most least squares applications,  $m > n$  and  $Ax = b$  has no solution

## Column interpretation

least squares problem in terms of columns  $a_1, a_2, \dots, a_n$  of  $A$ :

$$\text{minimize} \quad \|Ax - b\|^2 = \left\| \sum_{j=1}^n a_j x_j - b \right\|^2$$



- $A\hat{x}$  is the vector in  $\text{range } A = \text{span}(a_1, a_2, \dots, a_n)$  closest to  $b$
- geometric intuition suggests that  $\hat{r} = b - A\hat{x}$  is orthogonal to  $\text{range}(A)$

## Row interpretation

least squares problem in terms of rows  $\tilde{a}_1^T, \tilde{a}_2^T, \dots, \tilde{a}_m^T$  of  $A$

$$\text{minimize} \quad \|Ax - b\|^2 = (\tilde{a}_1^T x - b_1)^2 + \dots + (\tilde{a}_m^T x - b_m)^2$$

- if  $\tilde{a}_i \neq 0$ , distance of  $x$  to hyperplane  $H_i = \{y \mid \tilde{a}_i^T y = b_i\}$  is

$$d_i(x) = \frac{|\tilde{a}_i^T x - b_i|}{\|\tilde{a}_i\|}$$

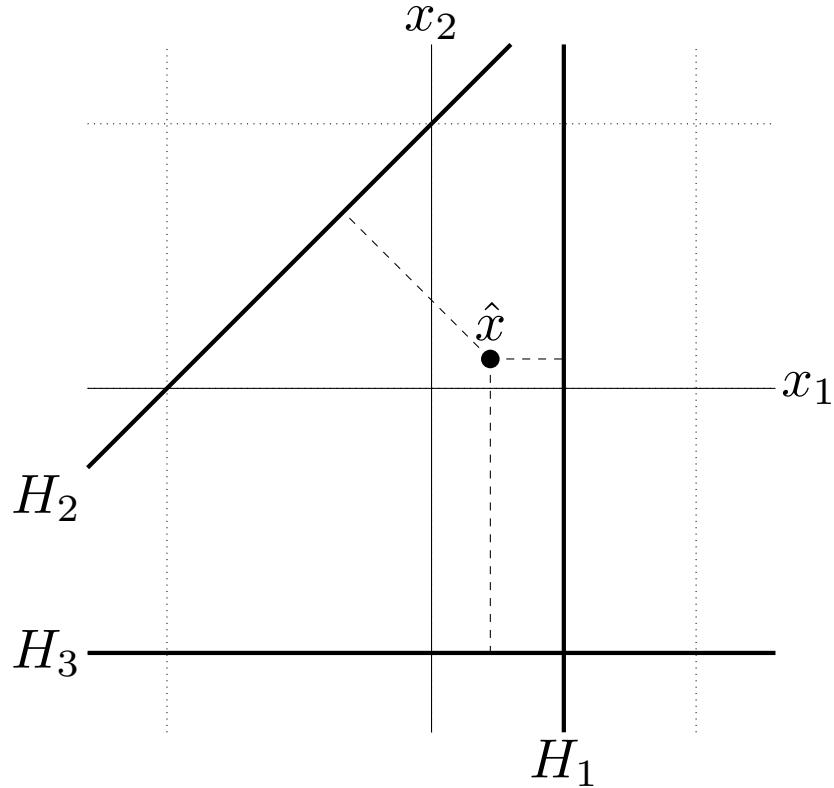
- least squares solution minimizes weighted sum of squared distances

$$\|Ax - b\|^2 = \sum_{i=1}^m w_i d_i(x)^2 \quad \text{with weights } w_i = \|\tilde{a}_i\|^2$$

- if row norms are equal, LS solution minimizes sum of squared distances

# Example

$$A = \begin{bmatrix} 2 & 0 \\ 1 & -1 \\ 0 & -1 \end{bmatrix}$$
$$b = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$



$$\|Ax - b\|^2 = 4d_1(x)^2 + 2d_2(x)^2 + d_3(x)^2$$

$d_1(x)$  is distance to  $H_1$ ,  $d_2(x)$  is distance to  $H_2$ ,  $d_3(x)$  is distance to  $H_3$

# Outline

- least squares problem
- **solution of a least squares problem**
- solving least squares problems

## Solution of a least squares problem

if  $A$  has linearly independent columns (is left invertible), then the vector

$$\begin{aligned}\hat{x} &= (A^T A)^{-1} A^T b \\ &= A^\dagger b\end{aligned}$$

is the unique solution of the least squares problem

$$\text{minimize } \|Ax - b\|^2$$

- in other words, if  $x \neq \hat{x}$ , then  $\|Ax - b\|^2 > \|\hat{x} - b\|^2$
- recall from page 4-23 that

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

is the *pseudo-inverse* of a left invertible matrix

## Proof

we show that  $\|Ax - b\|^2 > \|A\hat{x} - b\|^2$  for  $x \neq \hat{x}$ :

$$\begin{aligned}\|Ax - b\|^2 &= \|A(x - \hat{x}) + (A\hat{x} - b)\|^2 \\ &= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 \\ &> \|A\hat{x} - b\|^2\end{aligned}$$

- 2nd step follows from  $A(x - \hat{x}) \perp (A\hat{x} - b)$ :

$$(A(x - \hat{x}))^T (A\hat{x} - b) = (x - \hat{x})^T (A^T A\hat{x} - A^T b) = 0$$

- 3rd step follows from linear independence of columns of  $A$ :

$$A(x - \hat{x}) \neq 0 \quad \text{if } x \neq \hat{x}$$

## Derivation from calculus

$$f(x) = \|Ax - b\|^2 = \sum_{i=1}^m \left( \sum_{j=1}^n A_{ij}x_j - b_i \right)^2$$

- partial derivative of  $f$  with respect to  $x_k$

$$\frac{\partial f}{\partial x_k}(x) = 2 \sum_{i=1}^m A_{ik} \left( \sum_{j=1}^n A_{ij}x_j - b_i \right) = 2(A^T(Ax - b))_k$$

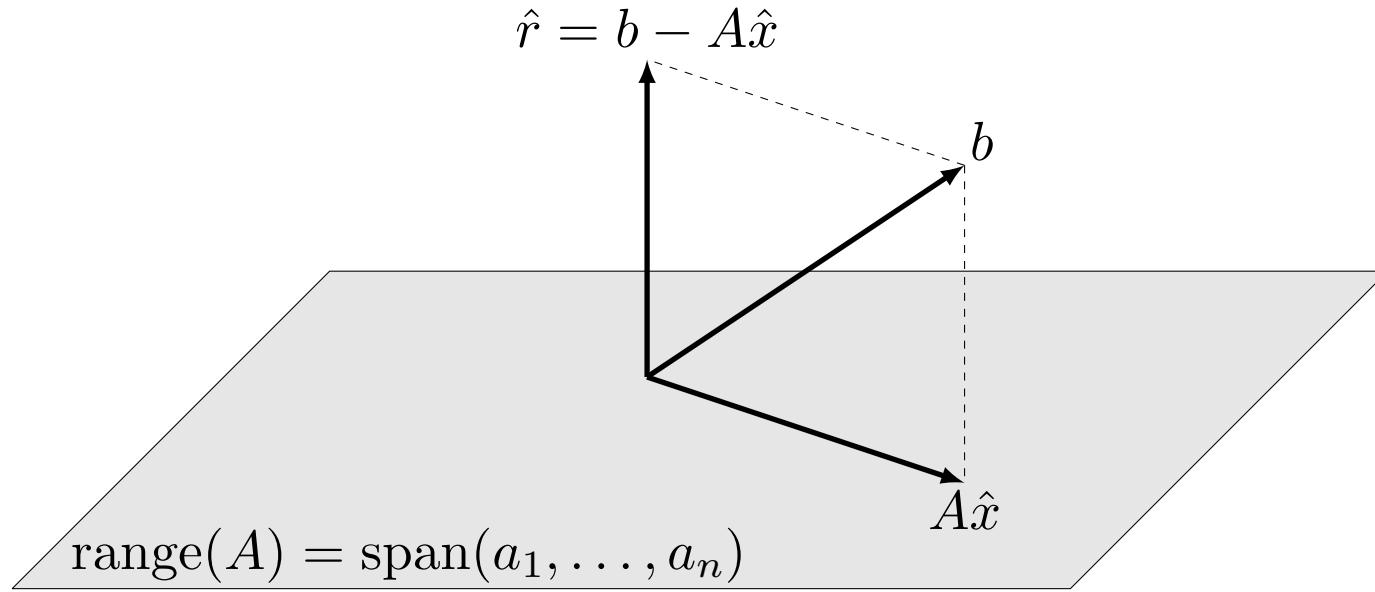
- gradient of  $f$  is

$$\nabla f(x) = \left( \frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \dots, \frac{\partial f}{\partial x_n}(x) \right) = 2A^T(Ax - b)$$

- minimizer  $\hat{x}$  of  $f(x)$  satisfies  $\nabla f(\hat{x}) = 2A^T(A\hat{x} - b) = 0$

## Geometric interpretation

residual vector  $\hat{r} = b - A\hat{x}$  satisfies  $A^T\hat{r} = A^T(b - A\hat{x}) = 0$



- residual vector is orthogonal to every column of  $A$ ; hence, to  $\text{range}(A)$
- projection on  $\text{range}(A)$  is a matrix-vector multiplication with the matrix

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

# Outline

- least squares problem
- solution of a least squares problem
- **solving least squares problems**

## Normal equations

$$A^T A x = A^T b$$

- these equations are called the *normal equations* of the LS problem
- coefficient matrix is the Gram matrix  $A^T A$  of  $A$
- equivalent to  $\nabla f(x) = 0$  where  $f(x) = \|Ax - b\|^2$
- all solutions of the least squares problem satisfy the normal equations

if  $A$  has linearly independent columns, then:

- $A^T A$  is nonsingular
- normal equations have a unique solution  $\hat{x} = (A^T A)^{-1} A^T b$

## QR factorization method

rewrite least squares solution using QR factorization  $A = QR$

$$\begin{aligned}\hat{x} = (A^T A)^{-1} A^T b &= ((QR)^T (QR))^{-1} (QR)^T b \\ &= (R^T Q^T QR)^{-1} R^T Q^T b \\ &= (R^T R)^{-1} R^T Q^T b \\ &= R^{-1} R^{-T} R^T Q^T b \\ &= R^{-1} Q^T b\end{aligned}$$

### Algorithm

1. compute QR factorization  $A = QR$  ( $2mn^2$  flops if  $A$  is  $m \times n$ )
2. matrix-vector product  $d = Q^T b$  ( $2mn$  flops)
3. solve  $Rx = d$  by back substitution ( $n^2$  flops)

complexity:  $2mn^2$  flops

## Example

$$A = \begin{bmatrix} 3 & -6 \\ 4 & -8 \\ 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} -1 \\ 7 \\ 2 \end{bmatrix}$$

1. QR factorization:  $A = QR$  with

$$Q = \begin{bmatrix} 3/5 & 0 \\ 4/5 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 5 & -10 \\ 0 & 1 \end{bmatrix}$$

2. calculate  $d = Q^T b = (5, 2)$

3. solve  $Rx = d$

$$\begin{bmatrix} 5 & -10 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

solution is  $x_1 = 5, x_2 = 2$

## Solving the normal equations

why not solve the normal equations

$$A^T A x = A^T b$$

as a set of linear equations?

**Example:** a  $3 \times 2$  matrix with ‘almost linearly dependent’ columns

$$A = \begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \\ 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 10^{-5} \\ 1 \end{bmatrix},$$

we round intermediate results to 8 significant decimal digits

## Solving the normal equations

**Method 1:** form Gram matrix  $A^T A$  and solve normal equations

$$A^T A = \begin{bmatrix} 1 & -1 \\ -1 & 1 + 10^{-10} \end{bmatrix} \rightsquigarrow \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad A^T b = \begin{bmatrix} 0 \\ 10^{-10} \end{bmatrix}$$

after rounding, the Gram matrix is singular; hence method fails

**Method 2:** QR factorization of  $A$  is

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \end{bmatrix}$$

rounding does not change any values (in this example)

- QR factorization method is more stable
- problem with method 1 occurs when forming Gram matrix  $A^T A$

## 9. Least squares applications

- model fitting
- multiobjective least squares
- control
- estimation
- statistics

# Model fitting

suppose a scalar quantity  $y$  and an  $n$ -vector  $x$  are related as

$$y \approx f(x)$$

- $y$  is the *outcome*, or *response variable*, or *dependent variable*
- $x$  is vector of *independent variables* or *explanatory variables*
- we don't know  $f$ , but have some idea about its general form

## Model fitting

- find an approximate *model*  $\hat{f} : \mathbf{R}^n \rightarrow \mathbf{R}$  for  $f$ , based on observations
- we use the notation  $\hat{y}$  for the model *prediction* of the outcome  $y$ :

$$\hat{y} = \hat{f}(x)$$

## Prediction error

we have  $N$  data points

$$(x_1, y_1), \quad (x_1, y_1), \quad \dots, \quad (x_N, y_N) \quad (\text{with each } x_i \in \mathbf{R}^n, y_i \in \mathbf{R})$$

data points are also called *observations*, *examples*, *samples*, *measurements*

- model prediction for sample  $i$  is  $\hat{y}_i = \hat{f}(x_i)$
- the *prediction error* or *residual* for sample  $i$  is

$$r_i = y_i - \hat{y}_i = y_i - \hat{f}(x_i)$$

- the model  $\hat{f}$  fits the data well if the  $N$  residuals  $r_i$  are small

## Linearly parameterized model

suppose we choose  $\hat{f}$  from a linearly parameterized family of models

$$\hat{f}(x) = \hat{\theta}_1 f_1(x) + \hat{\theta}_2 f_2(x) + \cdots + \hat{\theta}_p f_p(x)$$

- functions  $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$  are *basis functions* (chosen by us)
- coefficients  $\hat{\theta}_1, \dots, \hat{\theta}_p$  are estimated model *parameters*
- vector of residuals  $r = (r_1, \dots, r_N)$  is *affine* in the parameters  $\hat{\theta}$ :

$$\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_p(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_p(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_N) & f_2(x_N) & \cdots & f_p(x_N) \end{bmatrix} \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ \vdots \\ \hat{\theta}_p \end{bmatrix}$$

## Least squares model fitting

select model parameters  $\hat{\theta}$  by minimizing the *mean square error* (MSE)

$$\frac{1}{N}(r_1^2 + r_2^2 + \cdots + r_N^2) = \frac{1}{N}\|r\|^2$$

this is a linear least squares problem:  $\hat{\theta}$  is the solution of

$$\text{minimize } \|A\theta - y\|^2$$

with variable  $\theta$  and

$$A = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_p(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_p(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_N) & f_2(x_N) & \cdots & f_p(x_N) \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

# Generalization and validation

**Generalization ability:** ability of model to predict outcomes for new data

**Model validation:** to assess generalization ability,

- divide data in two sets: *training set* and *test (or validation)* set
- use training set to fit model
- use test set to get an idea of generalization ability

## Over-fit model

- model with low prediction error on training set, bad generalization ability
- prediction error on training set is much smaller than on test set

## Example: polynomial fitting

model is a polynomial of degree less than  $p$ :

$$\hat{f}(x) = \hat{\theta}_1 + \hat{\theta}_2 x + \hat{\theta}_3 x^2 + \cdots + \hat{\theta}_p x^{p-1}$$

- this is a special case of page 9-4 with  $n = 1$  and basis functions

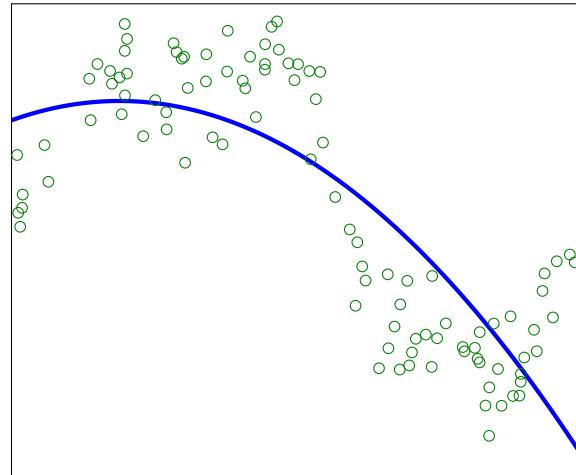
$$f_i(x) = x^{i-1}, \quad i = 1, \dots, p$$

- matrix  $A$  is the Vandermonde matrix

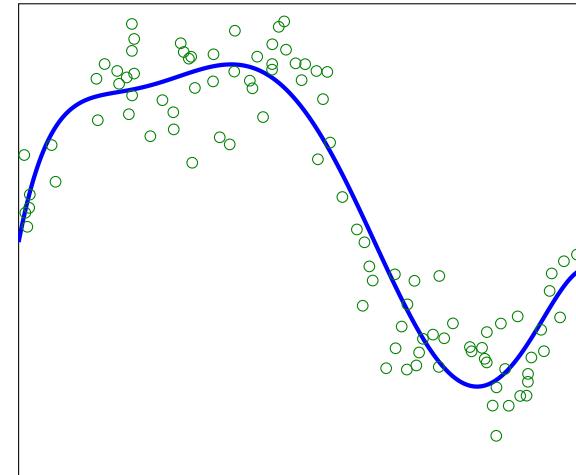
$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{p-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{p-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^{p-1} \end{bmatrix}$$

# Example

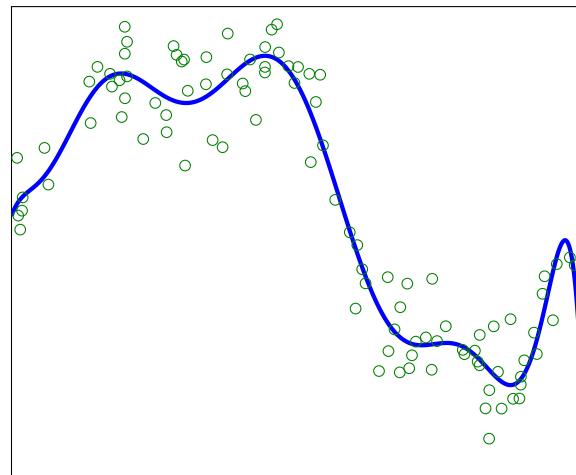
$p = 3$



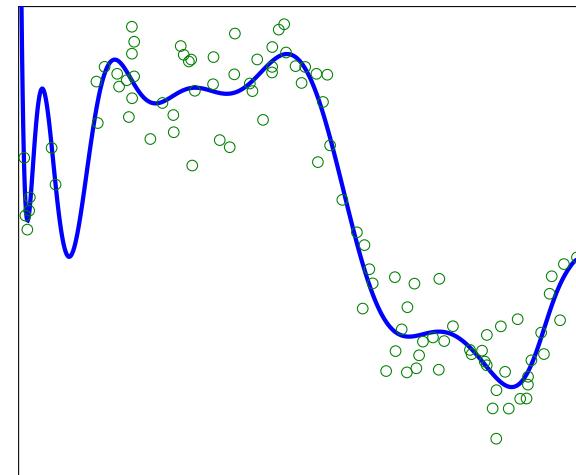
$p = 7$



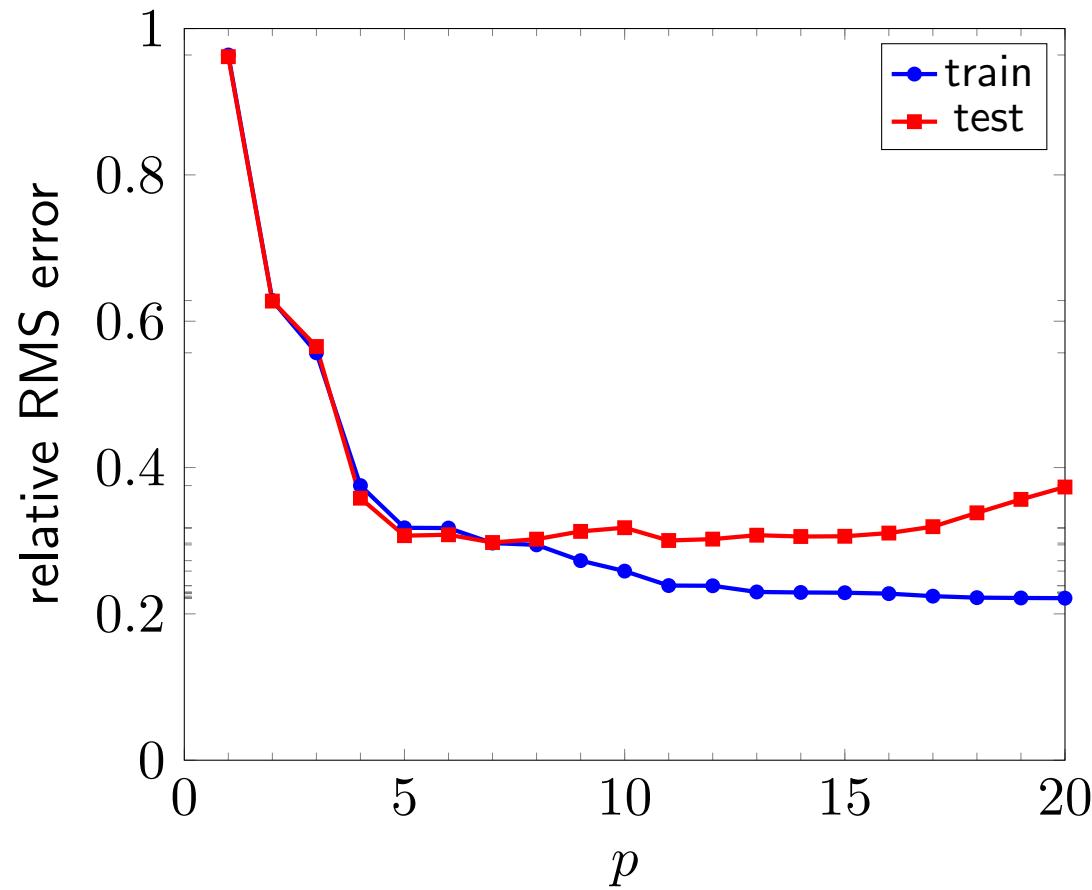
$p = 12$



$p = 20$



## Prediction error on training and test set

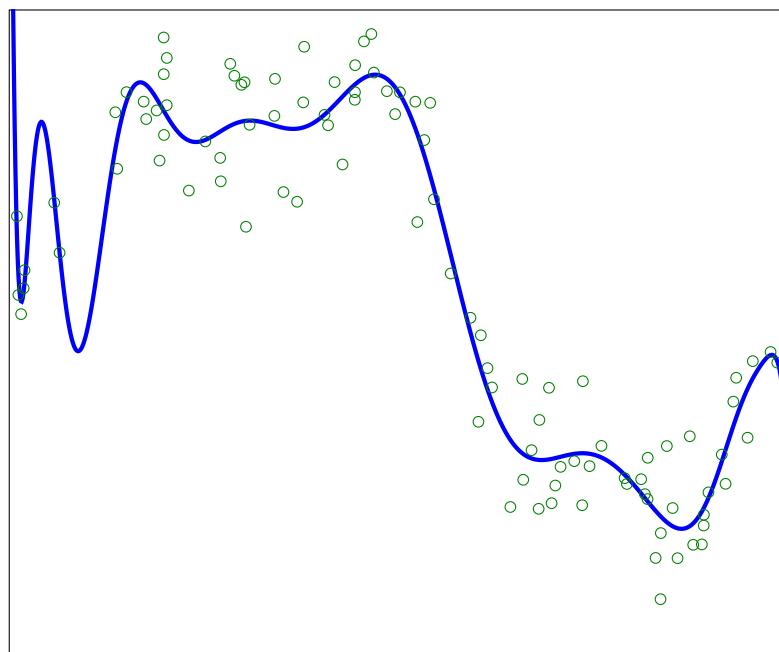


- plot shows relative RMS error  $\|y - A\hat{\theta}\| / \|y\|$  versus  $p$
- suggests choosing  $p = 7$

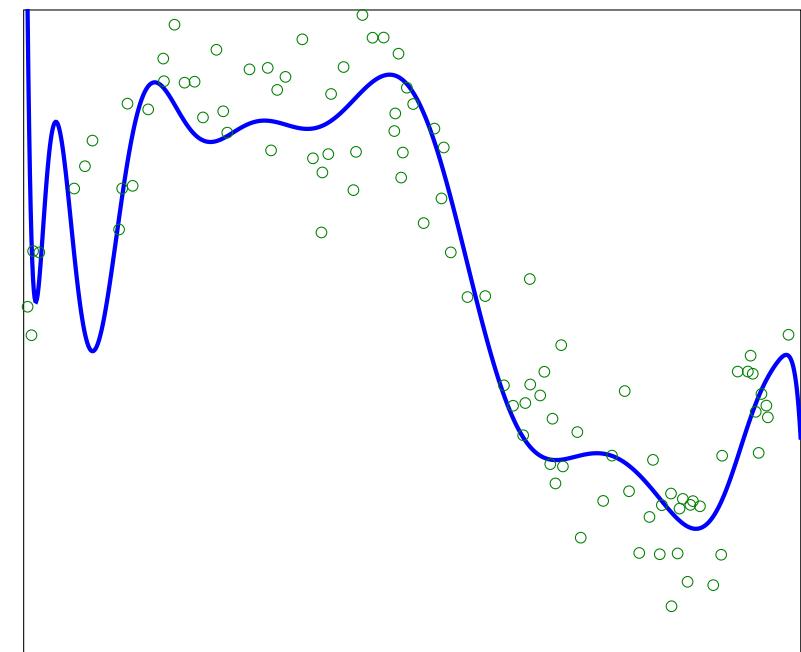
# Overfitting

polynomial of degree 19 ( $p = 20$ ) on training and test set

training set



test set



over-fitting is evident at the left end of the interval

# Outline

- model fitting
- **multiobjective least squares**
- control
- estimation
- statistics

## Weighted least squares problem

find  $x$  that minimizes

$$\lambda_1 \|A_1x - b_1\|^2 + \lambda_2 \|A_2x - b_2\|^2 + \cdots + \lambda_k \|A_kx - b_k\|^2$$

- matrix  $A_i$  has size  $m_i \times n$ ; vector  $b_i$  has length  $m_i$
- coefficients  $\lambda_i$  are positive weights
- goal is to find one  $x$  that makes the  $k$  objectives  $\|A_i x - b_i\|^2$  small
- in general, there is a trade-off between the  $k$  objectives
- weights  $\lambda_i$  express relative importance of different objectives
- without loss of generality, can choose  $\lambda_1 = 1$

## Solution of weighted least squares

- weighted least squares is equivalent to a standard least squares problem

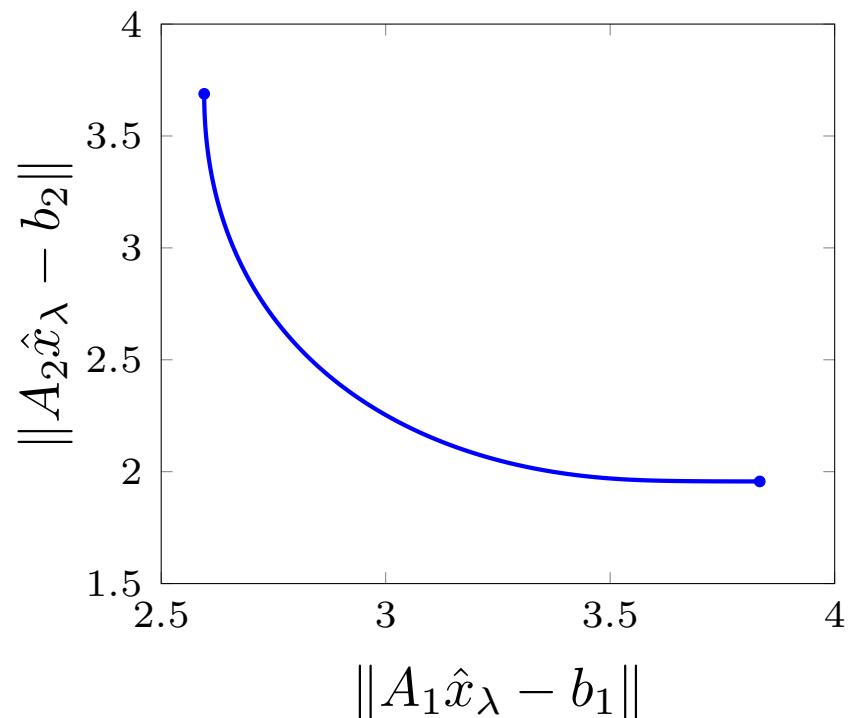
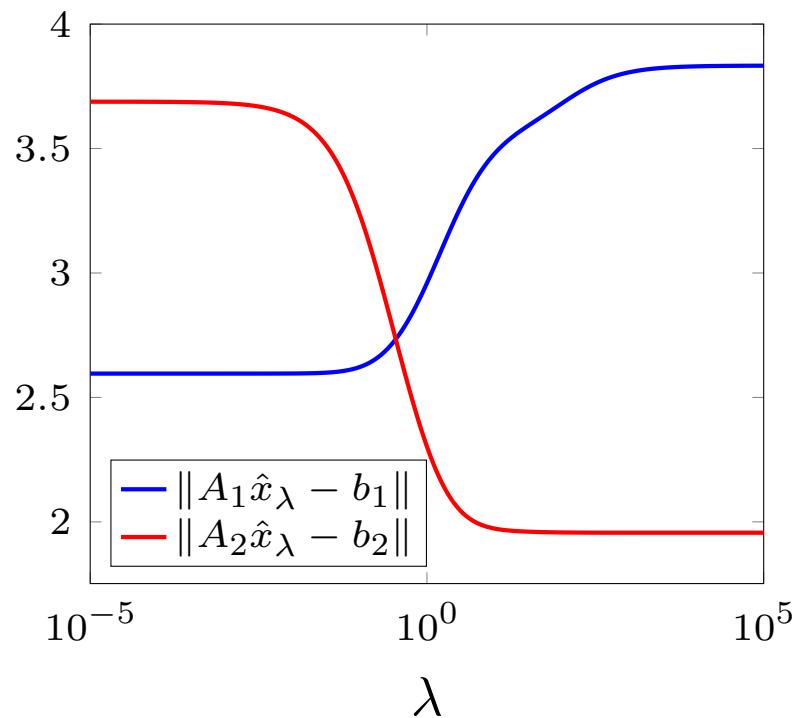
$$\text{minimize} \quad \left\| \begin{bmatrix} \sqrt{\lambda_1}A_1 \\ \sqrt{\lambda_2}A_2 \\ \vdots \\ \sqrt{\lambda_k}A_k \end{bmatrix} x - \begin{bmatrix} \sqrt{\lambda_1}b_1 \\ \sqrt{\lambda_2}b_2 \\ \vdots \\ \sqrt{\lambda_k}b_k \end{bmatrix} \right\|^2$$

- solution is unique if *stacked matrix* has linearly independent columns
- each matrix  $A_i$  may have linearly dependent columns (or be wide)
- solution satisfies normal equations

$$(\lambda_1 A_1^T A_1 + \cdots + \lambda_k A_k^T A_k) x = \lambda_1 A_1^T b_1 + \cdots + \lambda_k A_k^T b_k$$

# Example

$$\text{minimize} \quad \|A_1x - b_1\|^2 + \lambda \|A_2x - b_2\|^2$$



$\hat{x}_\lambda$  is solution of weighted least squares problem with weight  $\lambda$

# Outline

- model fitting
- multiobjective least squares
- **control**
- estimation
- statistics

# Control

$$y = Ax + b$$

- $x$  is  $n$ -vector of *actions* or *inputs*
- $y$  is  $m$ -vector of *results* or *outputs*
- relation between inputs and outputs is known affine function

goal is to choose inputs to optimize different objectives on  $x$  and  $y$

# Optimal input design

## Linear dynamical system

$$y(t) = h_0 u(t) + h_1 u(t-1) + h_2 u(t-2) + \cdots + h_t u(0)$$

- output  $y(t)$  and input  $u(t)$  are scalar
- we assume input  $u(t)$  is zero for  $t < 0$
- output is linear combination of current and previous inputs
- coefficients  $h_0, h_1, \dots$  are the *impulse response coefficients*

## Optimal input design

- optimization variable is the input sequence  $x = (u(0), u(1), \dots, u(N))$
- goal is to track a desired output using a small and slowly varying input

# Input design objectives

$$\text{minimize } J_t(x) + \lambda_v J_v(x) + \lambda_m J_m(x)$$

- track desired output  $y_{\text{des}}$  over an interval  $[0, N]$ :

$$J_t(x) = \sum_{t=0}^N (y(t) - y_{\text{des}}(t))^2$$

- use a small and slowly varying input signal:

$$J_m(x) = \sum_{t=0}^N u(t)^2, \quad J_v(x) = \sum_{t=0}^{N-1} (u(t+1) - u(t))^2$$

# Tracking error

$$\begin{aligned}
 J_t(x) &= \sum_{t=0}^N (y(t) - y_{\text{des}}(t))^2 \\
 &= \|A_t x - b_t\|^2
 \end{aligned}$$

with

$$A_t = \begin{bmatrix} h_0 & 0 & 0 & \cdots & 0 & 0 \\ h_1 & h_0 & 0 & \cdots & 0 & 0 \\ h_2 & h_1 & h_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_{N-1} & h_{N-2} & h_{N-3} & \cdots & h_0 & 0 \\ h_N & h_{N-1} & h_{N-2} & \cdots & h_1 & h_0 \end{bmatrix}, \quad b_t = \begin{bmatrix} y_{\text{des}}(0) \\ y_{\text{des}}(1) \\ y_{\text{des}}(2) \\ \vdots \\ y_{\text{des}}(N-1) \\ y_{\text{des}}(N) \end{bmatrix}$$

# Input variation and magnitude

## Input variation

$$J_v(x) = \sum_{t=0}^{N-1} (u(t+1) - u(t))^2 = \|Dx\|^2$$

with  $D$  the  $N \times (N + 1)$  matrix

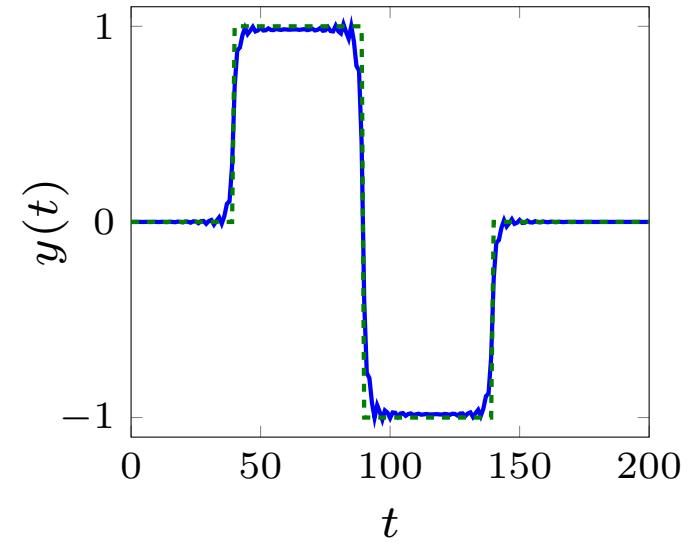
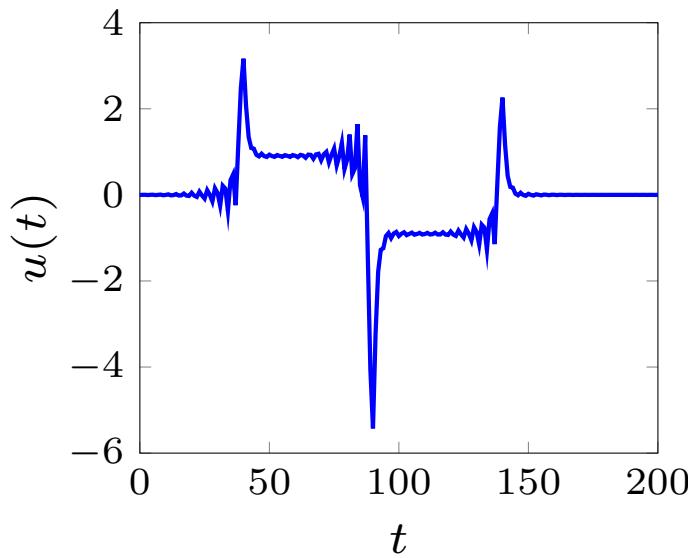
$$D = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 1 \end{bmatrix}$$

## Input magnitude

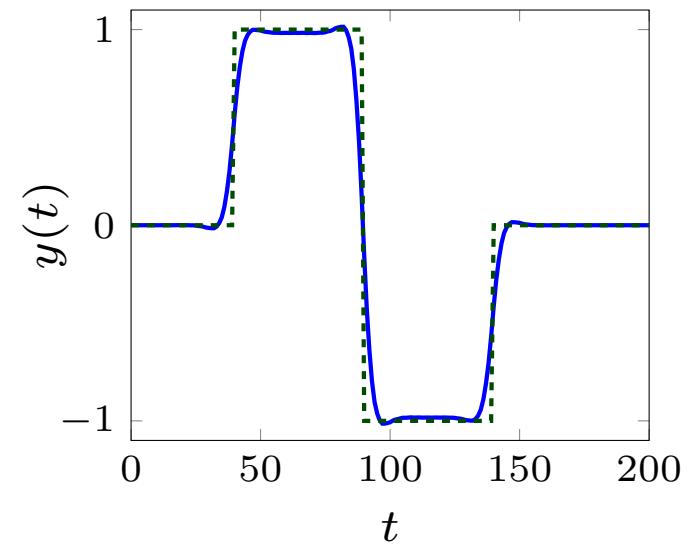
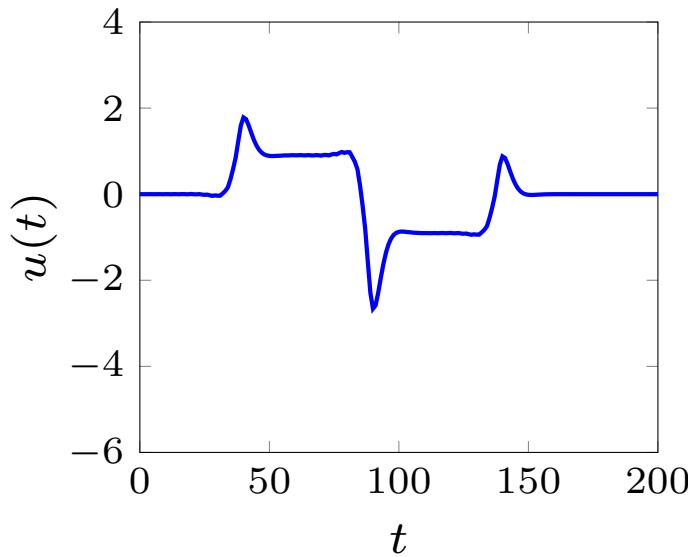
$$J_m(x) = \sum_{t=0}^N u(t)^2 = \|x\|^2$$

# Example

$\lambda_v = 0,$   
small  $\lambda_m$



larger  $\lambda_v$



# Outline

- model fitting
- multiobjective least squares
- control
- **estimation**
- statistics

# Estimation

## Linear measurement model

$$y = Ax + v$$

- $n$ -vector  $x$  contains parameters that we want to estimate
- $m$ -vector  $v$  is unknown measurement error or noise
- $m$ -vector  $y$  contains measurements
- $m \times n$  matrix  $A$  relates measurements and parameters

**Least squares estimate:** use as estimate of  $x$  the solution  $\hat{x}$  of

$$\text{minimize } \|Ax - y\|^2$$

# Regularized estimation

add other terms to  $\|Ax - y\|^2$  to include information about parameters

## Example: Tikhonov regularization

$$\text{minimize} \quad \|Ax - y\|^2 + \lambda\|x\|^2$$

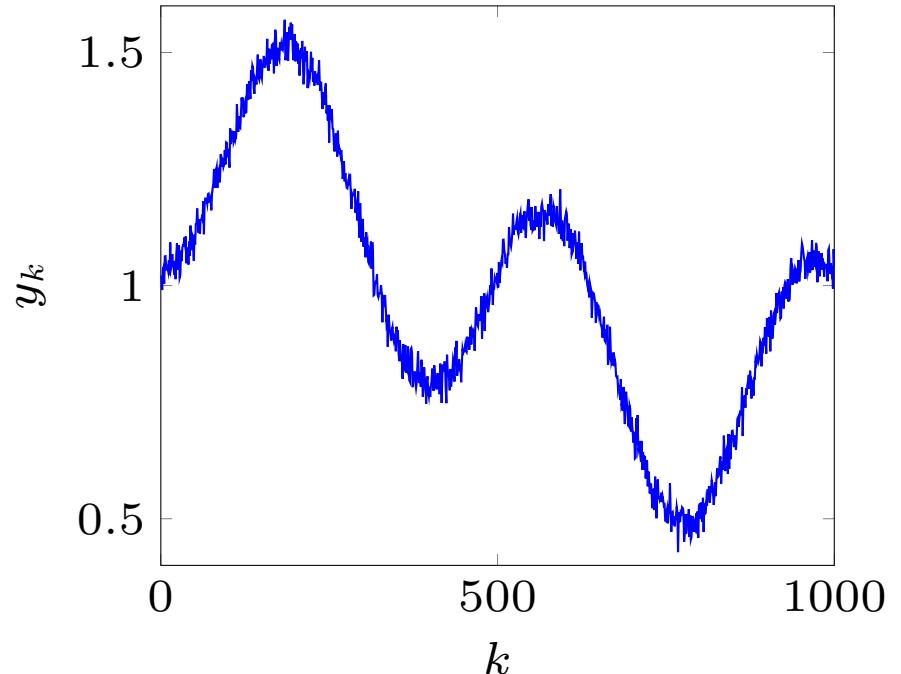
- goal is to make  $\|Ax - y\|$  small with small  $x$
- equivalent to solving  $(A^T A + \lambda I)x = A^T y$
- solution is unique (if  $\lambda > 0$ ) even when  $A$  has dependent columns

# Signal denoising

- observed signal is  $n$ -vector

$$y = x + v$$

- $x$  is unknown signal
- $v$  is noise



## Least squares denoising

$$\text{minimize} \quad \|x - y\|^2 + \lambda \sum_{i=1}^{n-1} (x_i - x_{i-1})^2$$

goal is to find slowly varying signal  $\hat{x}$ , close to observed signal  $y$

# Matrix formulation

$$\text{minimize} \quad \left\| \begin{bmatrix} I \\ \sqrt{\lambda}D \end{bmatrix} x - \begin{bmatrix} y \\ 0 \end{bmatrix} \right\|^2$$

- $D$  is  $(n - 1) \times n$  finite difference matrix

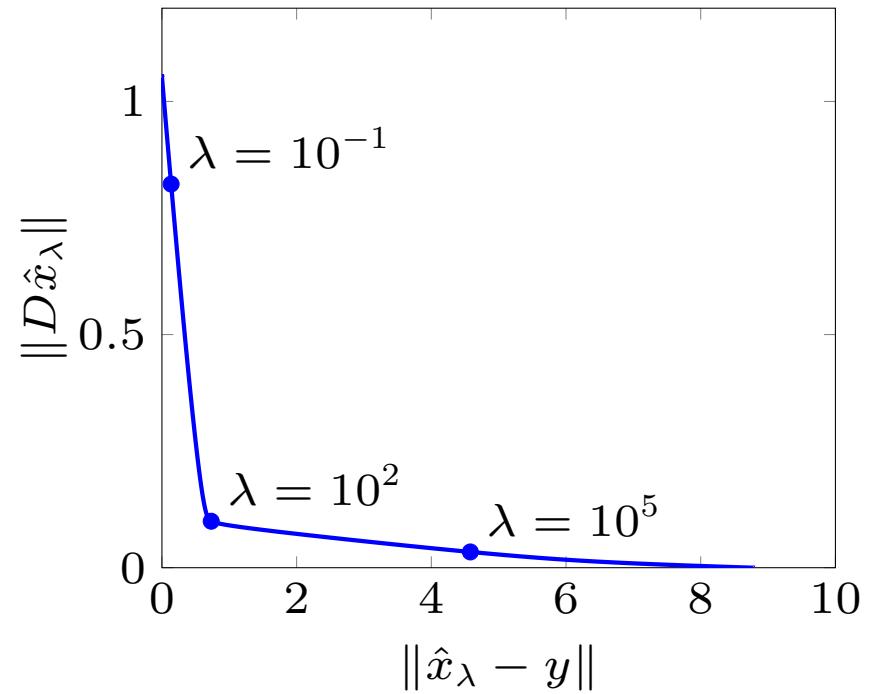
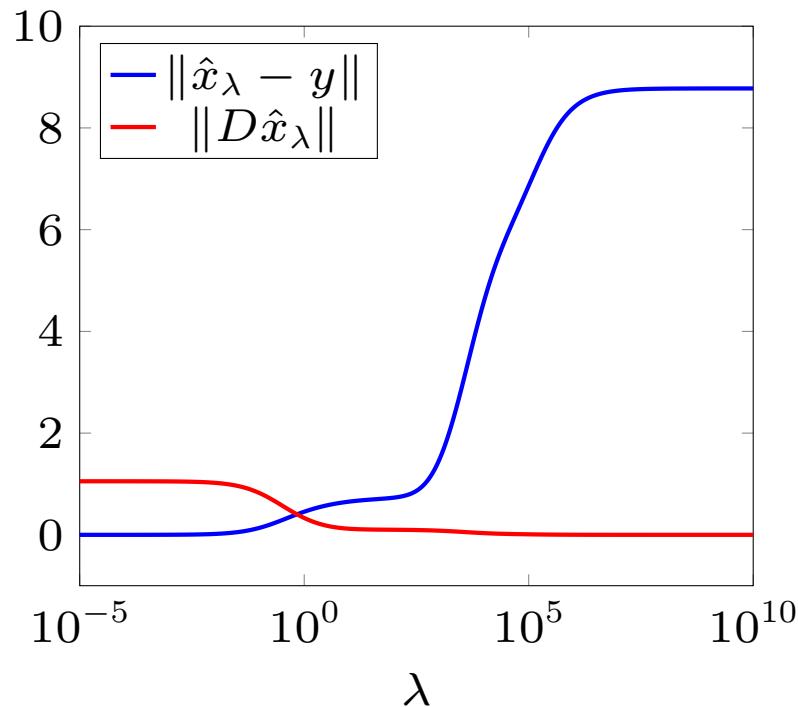
$$D = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 \end{bmatrix}$$

- equivalent to linear equation

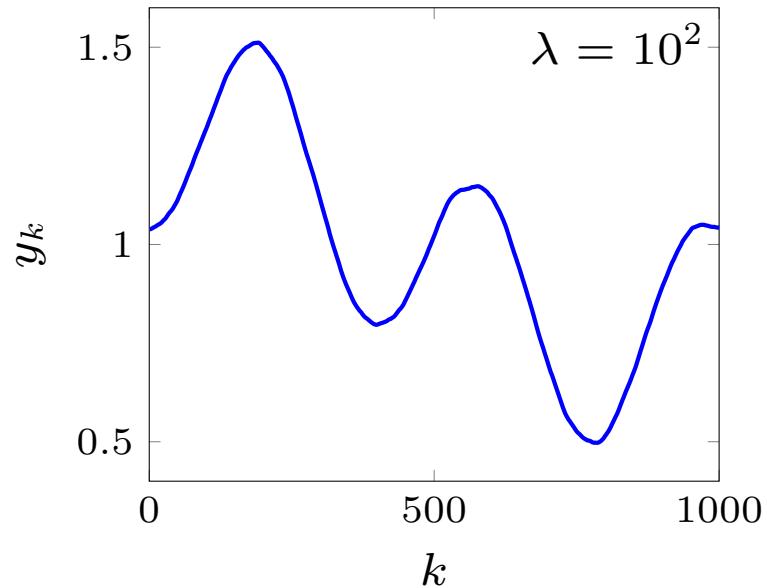
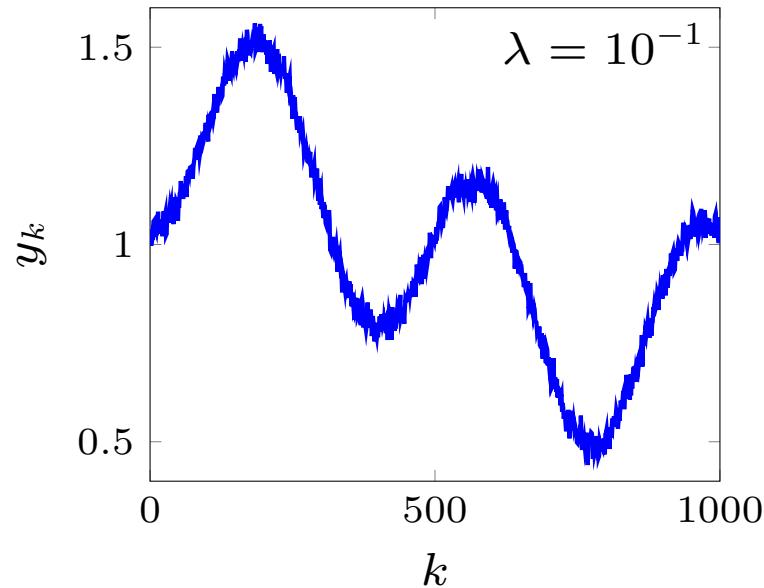
$$(I + \lambda D^T D)x = y$$

# Trade-off

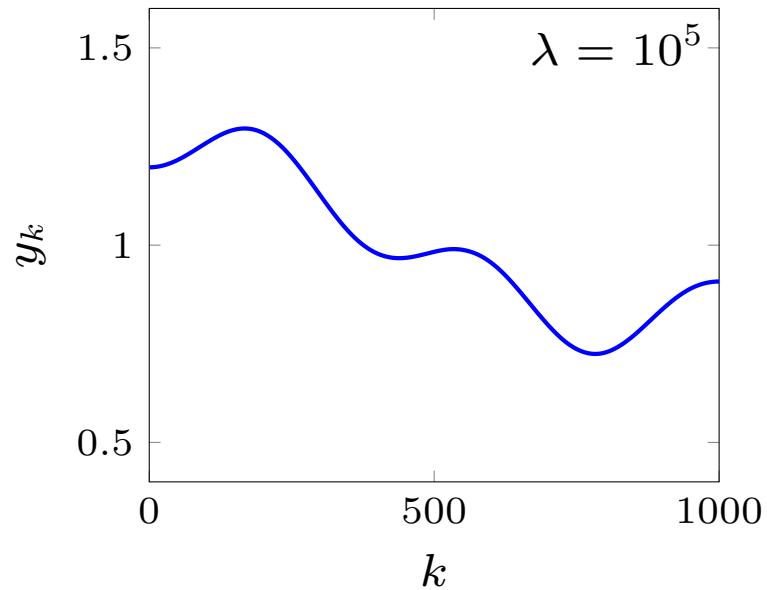
the two objectives  $\|\hat{x}_\lambda - y\|$  and  $\|D\hat{x}_\lambda\|$  for varying  $\lambda$



## Three solutions



- $\hat{x}_\lambda \rightarrow y$  for  $\lambda \rightarrow 0$
- $\hat{x}_\lambda \rightarrow \text{avg}(y)\mathbf{1}$  for  $\lambda \rightarrow \infty$
- $\lambda \approx 10^2$  is good compromise



# Image deblurring

$$y = Ax + v$$

- $x$  is unknown image,  $y$  is observed image
- $A$  is (known) blurring matrix,  $v$  is (unknown) noise
- images are  $M \times N$ , stored as  $MN$ -vectors



blurred, noisy image  $y$



deblurred image  $\hat{x}$

## Least squares deblurring

$$\text{minimize} \quad \|Ax - y\|^2 + \lambda(\|D_v x\|^2 + \|D_h x\|^2)$$

- 1st term is '*data fidelity*' term: ensures  $A\hat{x} \approx y$
- 2nd term penalizes differences between values at neighboring pixels

$$\begin{aligned} & \|D_h x\|^2 + \|D_v x\|^2 \\ = & \sum_{i=1}^M \sum_{j=1}^{N-1} (X_{ij} - X_{i,j+1})^2 + \sum_{i=1}^{M-1} \sum_{j=1}^N (X_{ij} - X_{i+1,j})^2 \end{aligned}$$

if  $X$  is the  $M \times N$  image stored in the  $MN$ -vector  $x$

## Differencing operations in matrix notation

suppose  $x$  is the  $M \times N$  image  $X$ , stored column-wise as  $MN$ -vector

$$x = (X_{1:M,1}, X_{1:M,2}, \dots, X_{1:M,N})$$

- horizontal differencing:  $(N - 1) \times N$  block matrix with  $M \times M$  blocks

$$D_h = \begin{bmatrix} I & -I & 0 & \cdots & 0 & 0 & 0 \\ 0 & I & -I & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & I & -I \end{bmatrix}$$

- vertical differencing:  $M \times M$  block matrix with  $(N - 1) \times N$  blocks

$$D_v = \begin{bmatrix} D & 0 & \cdots & 0 \\ 0 & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D \end{bmatrix}, \quad D = \begin{bmatrix} 1 & -1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 \end{bmatrix}$$

# Deblurred images

results for  $\lambda = 10^{-6}$ ,  $\lambda = 10^{-4}$ ,  $\lambda = 10^{-2}$ ,  $\lambda = 1$



# Outline

- model fitting
- multiobjective least squares
- control
- estimation
- **statistics**

# Linear regression model

$$y = X\beta + \epsilon$$

- $\epsilon$  is a random  $n$ -vector (*random error* or *disturbance*)
- $\beta$  is (non-random)  $p$ -vector of unknown *parameters*
- $X$  is an  $n \times p$  matrix ('*design*' matrix, i.e., result of experiment design)
- $y$  is an observable random  $n$ -vector
- this notation differs from previous sections but is common in statistics
- we discuss methods for estimating parameters  $\beta$  from observations of  $y$

# Assumptions

- $X$  is tall ( $n > p$ ) with linearly independent columns
- random disturbances  $\epsilon_i$  have zero mean

$$\mathbf{E} \epsilon_i = 0 \quad \text{for } i = 1, \dots, n$$

- random disturbances have equal variances  $\sigma^2$

$$\mathbf{E} \epsilon_i^2 = \sigma^2 \quad \text{for } i = 1, \dots, n$$

- random disturbances are uncorrelated (have zero covariances)

$$\mathbf{E}(\epsilon_i \epsilon_j) = 0 \quad \text{for } i, j = 1, \dots, n \text{ and } i \neq j$$

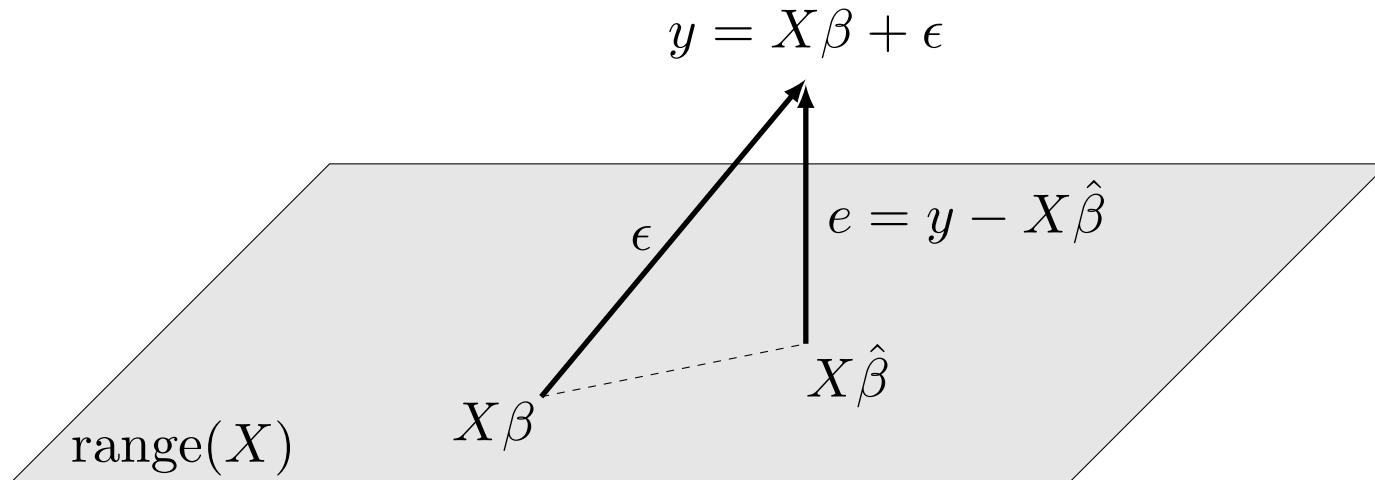
last three assumptions can be combined using matrix and vector notation:

$$\mathbf{E} \epsilon = 0, \quad \mathbf{E} \epsilon \epsilon^T = \sigma^2 I$$

## Least squares estimator

least squares estimate  $\hat{\beta}$  of parameters  $\beta$ , given the observations  $y$ , is

$$\hat{\beta} = X^\dagger y = (X^T X)^{-1} X^T y$$



- $X\hat{\beta}$  is the orthogonal projection of  $y$  on  $\text{range}(X)$
- residual  $e = y - X\hat{\beta}$  is an (observable) random variable

# Mean and covariance of least squares estimate

$$\hat{\beta} = X^\dagger(X\beta + \epsilon) = \beta + X^\dagger\epsilon$$

- least squares estimator is *unbiased*:  $\mathbf{E}\hat{\beta} = \beta$
- covariance matrix of least squares estimate is

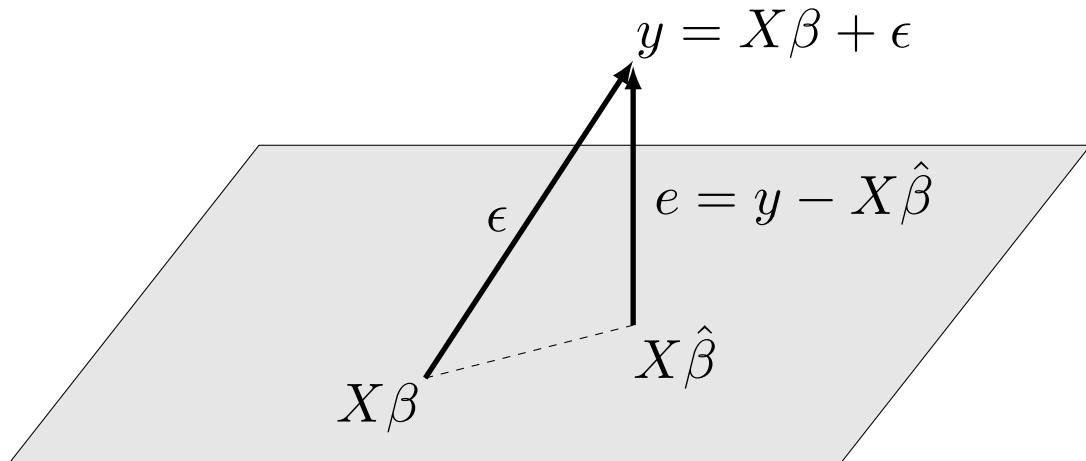
$$\begin{aligned}\mathbf{E}(\hat{\beta} - \beta)(\hat{\beta} - \beta)^T &= \mathbf{E}((X^\dagger\epsilon)(X^\dagger\epsilon)^T) \\ &= \mathbf{E}((X^T X)^{-1} X^T \epsilon \epsilon^T X (X^T X)^{-1}) \\ &= \sigma^2 (X^T X)^{-1}\end{aligned}$$

- covariance of  $\hat{\beta}_i$  and  $\hat{\beta}_j$  ( $i \neq j$ ) is

$$\mathbf{E}((\hat{\beta}_i - \beta_i)(\hat{\beta}_j - \beta_j)) = \sigma^2 ((X^T X)^{-1})_{ij}$$

for  $i = j$ , this is the variance of  $\hat{\beta}_i$

## Estimate of $\sigma^2$



$$\mathbf{E} \|\epsilon\|^2 = n\sigma^2$$

$$\mathbf{E} \|e\|^2 = (n - p)\sigma^2$$

$$\mathbf{E} \|X(\hat{\beta} - \beta)\|^2 = p\sigma^2$$

(proof on next page)

- define estimate  $\hat{\sigma}$  of  $\sigma$  as

$$\hat{\sigma} = \frac{\|e\|}{\sqrt{n - p}}$$

- $\hat{\sigma}^2$  is an unbiased estimate of  $\sigma^2$ :

$$\mathbf{E} \hat{\sigma}^2 = \frac{1}{n - p} \mathbf{E} \|e\|^2 = \sigma^2$$

*Proof.*

first expression is immediate:  $\mathbf{E} \|\epsilon\|^2 = \sum_{i=1}^n \mathbf{E} \epsilon_i^2 = n\sigma^2$

- to show that  $\mathbf{E} \|X(\hat{\beta} - \beta)\|^2 = p\sigma^2$ , first note that

$$\begin{aligned} X(\hat{\beta} - \beta) &= XX^\dagger y - X\beta \\ &= XX^\dagger(X\beta + \epsilon) - X\beta \\ &= XX^\dagger\epsilon \\ &= X(X^T X)^{-1} X^T \epsilon \end{aligned}$$

on line 3 we used  $X^\dagger X = I$  (however, note that  $XX^\dagger \neq I$  if  $X$  is tall!)

- squared norm of  $X(\beta - \hat{\beta})$  is

$$\|X(\hat{\beta} - \beta)\|^2 = \epsilon^T (XX^\dagger)^2 \epsilon = \epsilon^T XX^\dagger \epsilon$$

first step uses symmetry of  $XX^\dagger$ ; second step,  $X^\dagger X = I$

- expected value of squared norm is

$$\begin{aligned}
 \mathbf{E} \|X(\hat{\beta} - \beta)\|^2 &= \mathbf{E} (\epsilon^T X X^\dagger \epsilon) = \sum_{i,j} \mathbf{E}(\epsilon_i \epsilon_j) (X X^\dagger)_{ij} \\
 &= \sigma^2 \sum_{i=1}^n (X X^\dagger)_{ii} \\
 &= \sigma^2 \sum_{i=1}^n \sum_{j=1}^p X_{ij} (X^\dagger)_{ji} \\
 &= \sigma^2 \sum_{j=1}^p (X^\dagger X)_{jj} \\
 &= p\sigma^2
 \end{aligned}$$

- expression  $\mathbf{E} \|e\|^2 = (n - p)\sigma^2$  on page 9-34 now follows from

$$\|\epsilon\|^2 = \|e + X\hat{\beta} - X\beta\|^2 = \|e\|^2 + \|X(\hat{\beta} - \beta)\|^2$$

## Linear estimator

linear regression model (p.9-30), with same assumptions as before (p.9-31):

$$y = X\beta + \epsilon$$

a *linear estimator* of  $\beta$  maps observations  $y$  to the estimate

$$\hat{\beta} = By$$

- estimator is defined by the  $p \times n$  matrix  $B$
- least squares estimator is an example with  $B = X^\dagger$

## Unbiased linear estimator

if  $B$  is a left inverse of  $X$ , then estimator  $\hat{\beta} = By$  can be written as:

$$\hat{\beta} = By = B(X\beta + \epsilon) = \beta + B\epsilon$$

- this shows that the linear estimator is *unbiased* ( $\mathbf{E}\hat{\beta} = \beta$ ) if  $BX = I$
- covariance matrix of unbiased linear estimator is

$$\mathbf{E}((\hat{\beta} - \beta)(\hat{\beta} - \beta)^T) = \mathbf{E}(B\epsilon\epsilon^T B^T) = \sigma^2 BB^T$$

- if  $c$  is an (non-random)  $p$ -vector, then estimate  $c^T\hat{\beta}$  of  $c^T\beta$  has variance

$$\mathbf{E}(c^T\hat{\beta} - c^T\beta)^2 = \sigma^2 c^T BB^T c$$

least squares estimator is an example with  $B = X^\dagger$  and  $BB^T = (X^T X)^{-1}$

## Best linear unbiased estimator

if  $B$  is a left inverse of  $X$  then for all  $p$ -vectors  $c$

$$c^T BB^T c \geq c^T (X^T X)^{-1} c$$

(proof on next page)

- left-hand side gives variance of  $c^T \hat{\beta}$  for linear unbiased estimator

$$\hat{\beta} = By$$

- right-hand side gives variance of  $c^T \hat{\beta}_{ls}$  for least squares estimator

$$\hat{\beta}_{ls} = X^\dagger y$$

- least squares estimator is the '*best linear unbiased estimator*' (BLUE)

this is known as the Gauss-Markov theorem

*Proof.*

- use  $BX = I$  to write  $BB^T$  as

$$\begin{aligned} BB^T &= (B - (X^T X)^{-1} X^T)(B^T - X(X^T X)^{-1}) + (X^T X)^{-1} \\ &= (B - X^\dagger)(B - X^\dagger)^T + (X^T X)^{-1} \end{aligned}$$

- hence,

$$\begin{aligned} c^T BB^T C &= c^T (B - X^\dagger)(B - X^\dagger)^T c + c^T (X^T X)^{-1} c \\ &= \|(B - X^\dagger)^T c\|^2 + c^T (X^T X)^{-1} c \\ &\geq c^T (X^T X)^{-1} c \end{aligned}$$

with equality if  $B = X^\dagger$

# 10. Constrained least squares

- least-norm problem
- least squares problem with equality constraints

## Least-norm problem

$$\begin{aligned} & \text{minimize} && \|x\|^2 \\ & \text{subject to} && Cx = d \end{aligned}$$

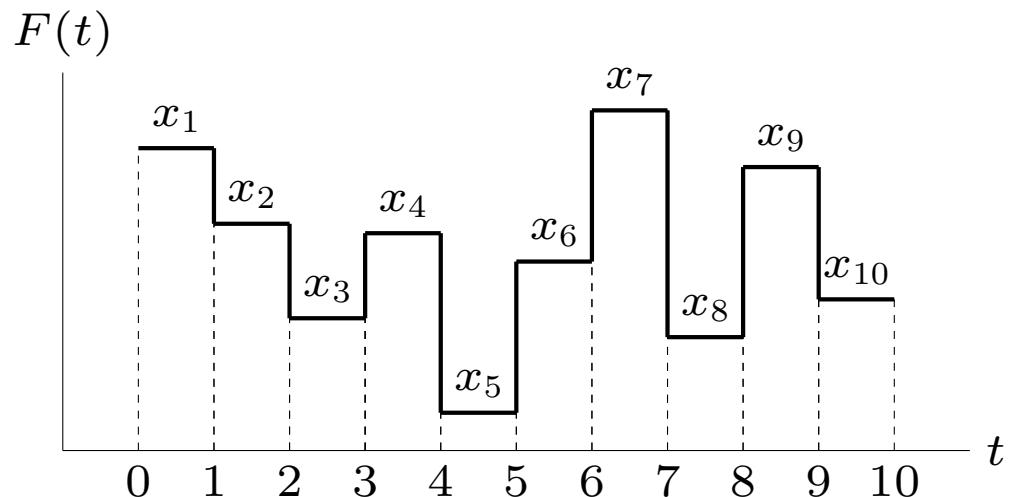
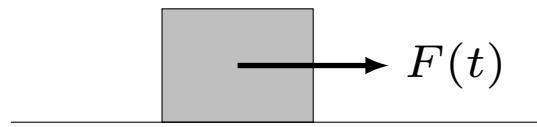
- $C$  is a  $p \times n$  matrix;  $d$  is a  $p$ -vector
- in most applications  $p < n$  and the equation  $Cx = d$  is *underdetermined*
- the goal is to find the solution of  $Cx = d$  with smallest norm

we will assume that  $C$  is right invertible (has linearly independent rows)

- equation  $Cx = d$  has at least one solution for every  $d$
- $C$  is wide or square ( $p \leq n$ )
- if  $p < n$  there are infinitely many solutions

# Example

example of page 1-27



- unit mass, with zero initial position and velocity
- piecewise-constant force  $F(t) = x_j$  for  $t \in [j-1, j)$ ,  $j = 1, \dots, 10$
- position and velocity at  $t = 10$  are given by  $y = Cx$  where

$$C = \begin{bmatrix} 19/2 & 17/2 & 15/2 & \dots & 1/2 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

## Example

forces that move mass over a unit distance with zero final velocity satisfy

$$\begin{bmatrix} 19/2 & 17/2 & 15/2 & \cdots & 1/2 \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix} x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

some interesting solutions:

- solutions with only two nonzero elements:

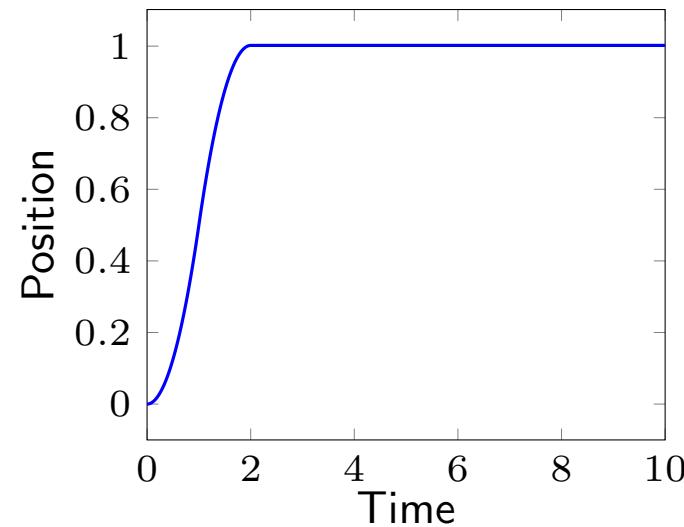
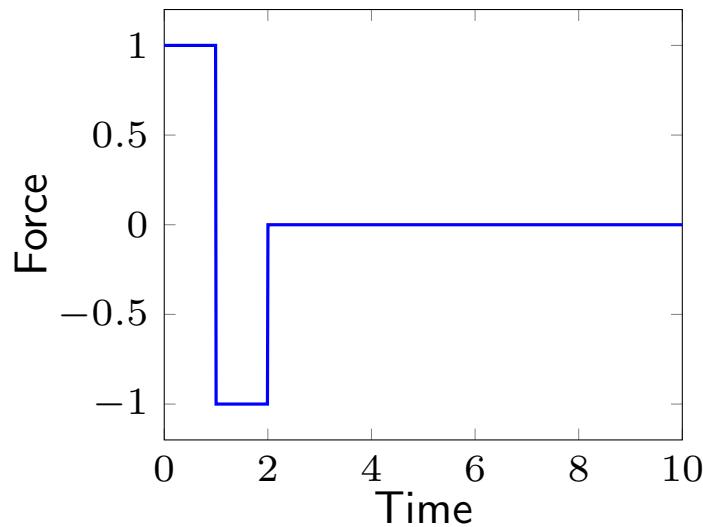
$$x = (1, -1, 0, \dots, 0), \quad x = (0, 1, -1, \dots, 0), \quad \dots,$$

- least-norm solution: minimizes

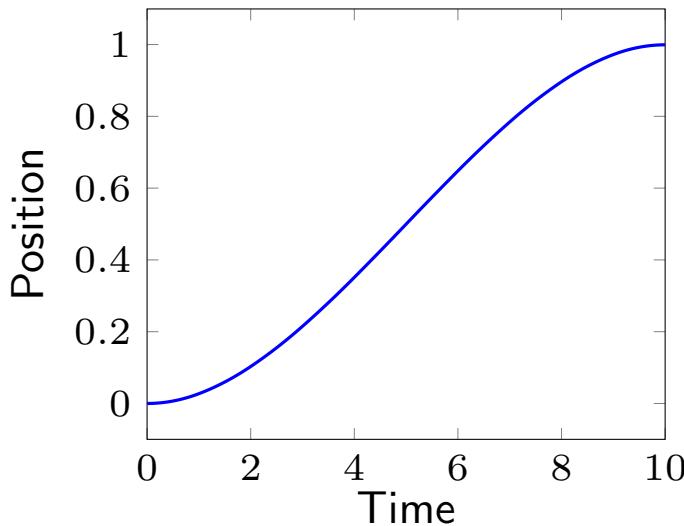
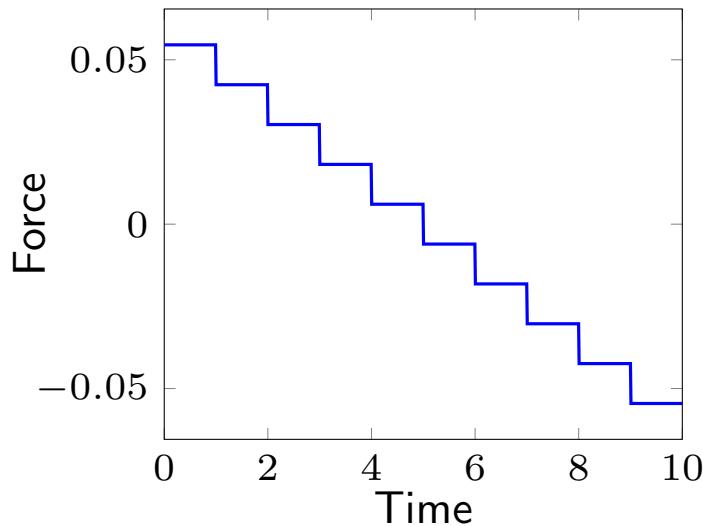
$$\int_0^{10} F(t)^2 dt = x_1^2 + x_2^2 + \cdots + x_{10}^2$$

# Example

$$x = (1, -1, 0, \dots, 0)$$



Least norm-solution



## Least-distance solution

as a variation, we can minimize the distance to a given point  $a \neq 0$ :

$$\begin{aligned} & \text{minimize} && \|x - a\|^2 \\ & \text{subject to} && Cx = d \end{aligned}$$

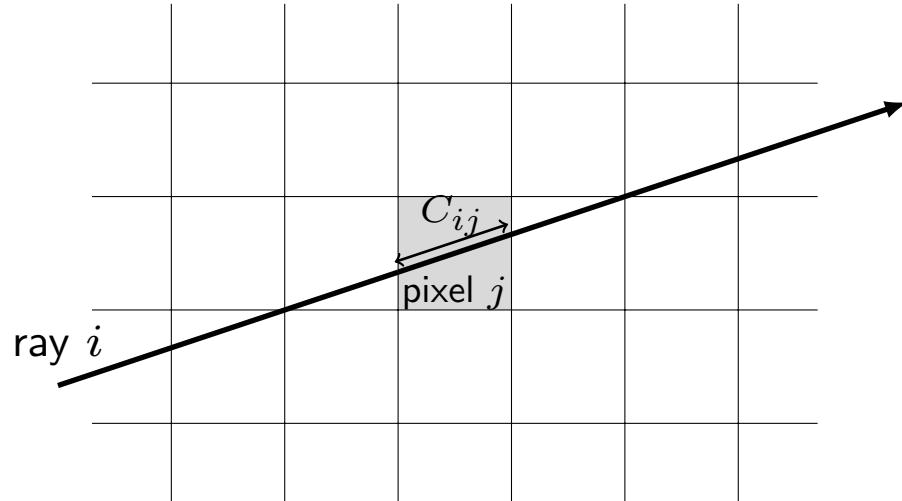
- reduces to least-norm problem by a change of variables  $y = x - a$

$$\begin{aligned} & \text{minimize} && \|y\|^2 \\ & \text{subject to} && Cy = d - Ca \end{aligned}$$

- from least-norm solution  $y$ , we obtain solution  $x = y + a$  of first problem

# Tomography

reconstruction of unknown image from line integrals (see page 2-39)



- $x$  is unknown image with  $n$  pixels
- one linear equation per line
- in practice, very underdetermined

to find estimate  $\hat{x}$ , solve

$$\begin{array}{ll}\text{minimize} & \|x - a\|^2 \\ \text{subject to} & Cx = d\end{array}$$

- $a$  is prior guess (for example, uniform intensity)
- goal is to find  $\hat{x}$  consistent with measurements, close to prior guess

## Solution of least-norm problem

if  $C$  has linearly independent rows (is right invertible), then

$$\hat{x} = C^T(CC^T)^{-1}d$$

is the unique solution of the least-norm problem

$$\begin{aligned} & \text{minimize} && \|x\|^2 \\ & \text{subject to} && Cx = d \end{aligned}$$

- in other words if  $Cx = d$  and  $x \neq \hat{x}$ , then  $\|x\| > \|\hat{x}\|$
- recall from page 4-26 that

$$C^T(CC^T)^{-1} = C^\dagger$$

is the pseudoinverse of a right invertible matrix  $C$

## Proof

1. we first verify that  $\hat{x}$  satisfies the equation:

$$C\hat{x} = CC^T(CC^T)^{-1}d = d$$

2. next we show that  $\|x\| > \|\hat{x}\|$  if  $Cx = d$  and  $x \neq \hat{x}$

$$\begin{aligned}\|x\|^2 &= \|\hat{x} + x - \hat{x}\|^2 \\ &= \|\hat{x}\|^2 + 2\hat{x}^T(x - \hat{x}) + \|x - \hat{x}\|^2 \\ &= \|\hat{x}\|^2 + \|x - \hat{x}\|^2 \\ &\geq \|\hat{x}\|^2\end{aligned}$$

with equality only if  $x = \hat{x}$

on line 3 we use  $Cx = C\hat{x} = d$  in

$$\hat{x}^T(x - \hat{x}) = d^T(CC^T)^{-1}C(x - \hat{x}) = 0$$

## QR factorization method

use the QR factorization  $C^T = QR$  of the left invertible matrix  $C^T$ :

$$\begin{aligned}\hat{x} &= C^T(CC^T)^{-1}d \\ &= QR(R^TQ^TQR)^{-1}d \\ &= QR(R^TR)^{-1}d \\ &= QR^{-T}d\end{aligned}$$

### Algorithm

1. compute QR factorization  $C^T = QR$  ( $2p^2n$  flops)
2. solve  $R^Tz = d$  by forward substitution ( $p^2$  flops)
3. matrix-vector product  $\hat{x} = Qz$  ( $2pn$  flops)

complexity:  $2p^2n$  flops

## Example

$$C = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & 0 & 1/2 & 1/2 \end{bmatrix}, \quad d = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- QR factorization  $C^T = QR$

$$\begin{bmatrix} 1 & 1 \\ -1 & 0 \\ 1 & 1/2 \\ 1 & 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/\sqrt{2} \\ -1/2 & 1/\sqrt{2} \\ 1/2 & 0 \\ 1/2 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 0 & 1/\sqrt{2} \end{bmatrix}$$

- solve  $R^T z = b$

$$\begin{bmatrix} 2 & 0 \\ 1 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$z_1 = 0, z_2 = \sqrt{2}$$

- evaluate  $\hat{x} = Qz = (1, 1, 0, 0)$

# Outline

- least-norm problem
- **least squares problem with equality constraints**

## Least squares with equality constraints

$$\begin{aligned} & \text{minimize} && \|Ax - b\|^2 \\ & \text{subject to} && Cx = d \end{aligned}$$

- $A$  is an  $m \times n$  matrix,  $C$  is  $p \times n$ ,  $b$  is an  $m$ -vector,  $d$  is a  $p$ -vector
- in most applications  $p < n$ , so equations are underdetermined
- the goal is to find the solution with smallest value of  $\|Ax - b\|$
- we do not assume that  $A$  is tall (let alone left-invertible)

**Special cases:** this extends least squares and least-norm problems

- least squares problem is a special case with  $p = 0$
- least-norm problem is a special case with  $A = I$ ,  $b = 0$

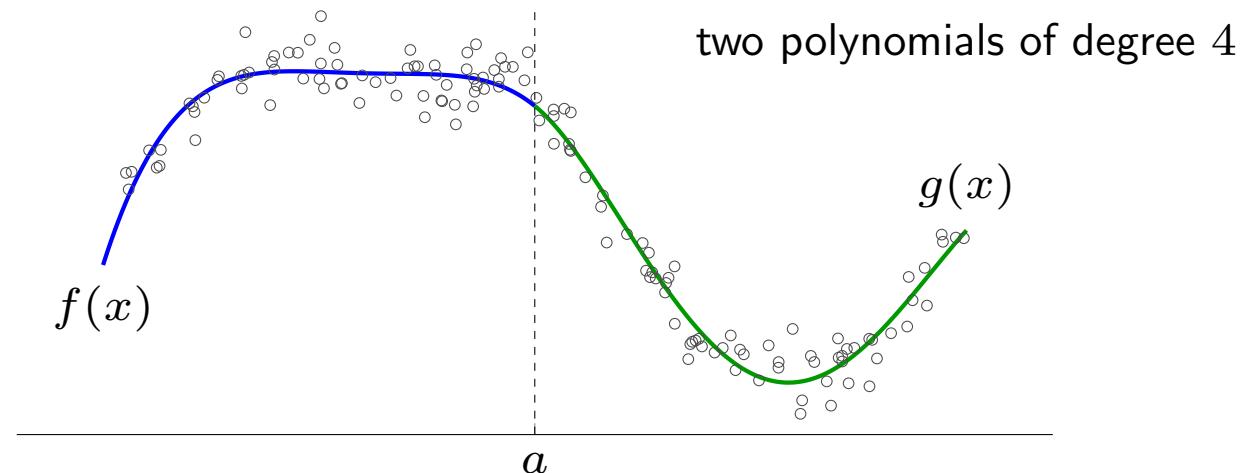
# Example

- fit two polynomials  $f(x)$ ,  $g(x)$  to points  $(x_i, y_i)$

$$f(x_i) \approx y_i \quad \text{for points } x_i \leq a, \quad g(x_i) \approx y_i \quad \text{for points } x_i > a$$

- make the values and derivatives continuous at the boundary point  $a$ :

$$f(a) = g(a), \quad f'(a) = g'(a)$$



# Assumptions

$$\begin{aligned} & \text{minimize} && \|Ax - b\|^2 \\ & \text{subject to} && Cx = d \end{aligned}$$

we will make the following two assumptions:

1. the stacked  $(m + p) \times n$  matrix

$$\left[ \begin{array}{c} A \\ C \end{array} \right]$$

has linearly independent columns (is left invertible)

2.  $C$  has linearly independent rows (is right invertible)

- note that assumption 1 is a weaker than left invertibility of  $A$
- assumptions imply that  $p \leq n \leq m + p$

# Optimality conditions

$\hat{x}$  solves the constrained LS problem if and only if there exists a  $z$  such that

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

(proof on next page)

- this is a set of  $n + p$  linear equations in  $n + p$  variables
- we'll see that the matrix on the left-hand side is nonsingular

## Special cases

- least squares: when  $p = 0$ , reduces to normal equations  $A^T A \hat{x} = A^T b$
- least-norm: when  $A = I$ ,  $b = 0$ , reduces to  $C \hat{x} = d$  and  $\hat{x} + C^T z = 0$

## Proof

suppose  $x$  satisfies  $Cx = d$ , and  $(\hat{x}, z)$  satisfies the equation on page 10-15

$$\begin{aligned}\|Ax - b\|^2 &= \|A(x - \hat{x}) + A\hat{x} - b\|^2 \\&= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 + 2(x - \hat{x})^T A^T (A\hat{x} - b) \\&= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 - 2(x - \hat{x})^T C^T z \\&= \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 \\&\geq \|A\hat{x} - b\|^2\end{aligned}$$

- on line 3 we use  $A^T A \hat{x} + C^T z = A^T b$ ; on line 4,  $Cx = C\hat{x} = d$
- inequality shows that  $\hat{x}$  is optimal
- $\hat{x}$  is the unique optimum because equality holds only if

$$A(x - \hat{x}) = 0, \quad C(x - \hat{x}) = 0 \quad \implies \quad x = \hat{x}$$

by the first assumption on page 10-14

## Nonsingularity

if the two assumptions hold, then the matrix

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix}$$

is nonsingular

*Proof.*

$$\begin{aligned} \begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = 0 &\implies x^T(A^T A x + C^T z) = 0, \quad Cx = 0 \\ &\implies \|Ax\|^2 = 0, \quad Cx = 0 \\ &\implies Ax = 0, \quad Cx = 0 \\ &\implies x = 0 \quad \text{by assumption 1} \end{aligned}$$

if  $x = 0$ , we have  $C^T z = -A^T Ax = 0$ ; hence also  $z = 0$  by assumption 2

## Nonsingularity

if the assumptions do not hold, then the matrix

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix}$$

is singular

- if 1 does not hold, there exists an  $x \neq 0$  with  $Ax = 0$  and  $Cx = 0$ ; then

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = 0$$

- if 2 does not hold there exists a  $z \neq 0$  with  $C^T z = 0$ ; then

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} 0 \\ z \end{bmatrix} = 0$$

in both cases, this shows that the matrix is singular

## Solution by LU factorization

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

### Algorithm

1. compute  $H = A^T A$  ( $mn^2$  flops)
2. compute  $c = A^T b$  ( $2mn$  flops)
3. solve the linear equation

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}$$

by the LU factorization  $((2/3)(p + n)^3$  flops)

complexity:  $mn^2 + (2/3)(p + n)^3$  flops

## Solution by QR factorization

we derive one of several possible methods based on the QR factorization

$$\begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

- if we define  $w = z - d$ , the equation can be written equivalently as

$$\begin{bmatrix} A^T A + C^T C & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ w \end{bmatrix} = \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

- assumption 1 guarantees that QR factorization of stacked matrix exists:

$$\begin{bmatrix} A \\ C \end{bmatrix} = QR = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R$$

## Solution by QR factorization

substituting the QR factorization gives the equation

$$\begin{bmatrix} R^T R & R^T Q_2^T \\ Q_2 R & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ w \end{bmatrix} = \begin{bmatrix} R^T Q_1^T b \\ d \end{bmatrix}$$

- multiply first equation with  $R^{-T}$  and make change of variables  $y = R\hat{x}$ :

$$\begin{bmatrix} I & Q_2^T \\ Q_2 & 0 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} = \begin{bmatrix} Q_1^T b \\ d \end{bmatrix}$$

- next we note that the matrix  $Q_2 = CR^{-1}$  has linearly independent rows:

$$Q_2^T u = R^{-T} C^T u = 0 \implies C^T u = 0 \implies u = 0$$

because  $C$  has linearly independent rows (assumption 2)

## Solution by QR factorization

we use the QR factorization of  $Q_2^T$  to solve

$$\begin{bmatrix} I & Q_2^T \\ Q_2 & 0 \end{bmatrix} \begin{bmatrix} y \\ w \end{bmatrix} = \begin{bmatrix} Q_1^T b \\ d \end{bmatrix}$$

- from the 1st block row,  $y = Q_1^T b - Q_2^T w$ ; substitute this in the 2nd row:

$$Q_2 Q_2^T w = Q_2 Q_1^T b - d$$

- we solve this equation for  $w$  using the QR factorization  $Q_2^T = \tilde{Q} \tilde{R}$ :

$$\tilde{R}^T \tilde{R} w = \tilde{R}^T \tilde{Q}^T Q_1^T b - d$$

which can be simplified to

$$\tilde{R} w = \tilde{Q}^T Q_1^T b - \tilde{R}^{-T} d$$

# Summary of QR factorization method

$$\begin{bmatrix} A^T A + C^T C & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ w \end{bmatrix} = \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

## Algorithm

1. compute the two QR factorizations

$$\begin{bmatrix} A \\ C \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \quad Q_2^T = \tilde{Q} \tilde{R}$$

2. solve  $\tilde{R}^T u = d$  by forward substitution and compute  $c = \tilde{Q}^T Q_1^T b - u$
3. solve  $\tilde{R}w = c$  by back substitution and compute  $y = Q_1^T b - Q_2^T w$
4. compute  $R\hat{x} = y$  by back substitution

complexity:  $2(p + m)n^2 + 2np^2$  flops for the QR factorizations

## Comparison of the two methods

**Complexity:** roughly the same

- LU factorization

$$mn^2 + \frac{2}{3}(p+n)^3 \leq mn^2 + \frac{16}{3}n^3 \text{ flops}$$

- QR factorization

$$2(p+m)n^2 + 2np^2 \leq 2mn^2 + 4n^3 \text{ flops}$$

upper bounds follow from  $p \leq n$  (assumption 2)

**Stability:** 2nd method avoids calculation of Gram matrix  $A^T A$

## Exercise

$A$  has linearly independent rows;  $f(\hat{x})$  denotes the solution of

$$\begin{array}{ll}\text{minimize} & \|x - \hat{x}\|^2 \\ \text{subject to} & Ax = b\end{array}$$

1. show that  $f(\hat{x}) = A^\dagger b + (I - A^\dagger A)\hat{x}$
2. consider one step in the Kaczmarz algorithm (with  $\|a_i\| = 1$ ):

$$x^{(j+1)} = x^{(j)} + (b_i - a_i^T x^{(j)}) a_i$$

show that  $f(x^{(j+1)}) = f(x^{(j)})$

3. let  $y$  be any point satisfying  $Ay = b$ ; show that

$$\|x^{(j+1)} - y\|^2 = \|x^{(j)} - y\|^2 - (b_i - a_i^T x^{(j)})^2$$

# 11. Cholesky factorization

- positive definite matrices
- examples
- Cholesky factorization
- complex positive definite matrices
- kernel methods
- least squares classification

# Definitions

- a symmetric matrix  $A \in \mathbf{R}^{n \times n}$  is *positive semidefinite* if

$$x^T A x \geq 0 \quad \text{for all } x$$

- a symmetric matrix  $A \in \mathbf{R}^{n \times n}$  is *positive definite* if

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

this is a subset of the positive semidefinite matrices

note: if  $A$  is symmetric and  $n \times n$ , then  $x^T A x$  is the function

$$x^T A x = \sum_{i=1}^n \sum_{j=1}^n A_{ij} x_i x_j = \sum_{i=1}^n A_{ii} x_i^2 + 2 \sum_{i>j} A_{ij} x_i x_j$$

this is called a *quadratic form*

## Example

$$A = \begin{bmatrix} 9 & 6 \\ 6 & a \end{bmatrix}$$

$$x^T A x = 9x_1^2 + 12x_1x_2 + ax_2^2 = (3x_1 + 2x_2)^2 + (a - 4)x_2^2$$

- $A$  is positive definite for  $a > 4$

$$x^T A x > 0 \quad \text{for all nonzero } x$$

- $A$  is positive semidefinite but not positive definite for  $a = 4$

$$x^T A x \geq 0 \quad \text{for all } x, \quad x^T A x = 0 \quad \text{for } x = (2, -3)$$

- $A$  is not positive semidefinite for  $a < 4$

$$x^T A x < 0 \quad \text{for } x = (2, -3)$$

## Simple properties

- every positive definite matrix  $A$  is nonsingular

$$Ax = 0 \implies x^T Ax = 0 \implies x = 0$$

(last step follows from positive definiteness)

- every positive definite matrix  $A$  has positive diagonal elements

$$A_{ii} = e_i^T A e_i > 0$$

- every positive semidefinite matrix  $A$  has nonnegative diagonal elements

$$A_{ii} = e_i^T A e_i \geq 0$$

# Schur complement

partition  $n \times n$  symmetric matrix  $A$  as

$$A = \begin{bmatrix} A_{11} & A_{2:n,1}^T \\ A_{2:n,1} & A_{2:n,2:n} \end{bmatrix}$$

- the *Schur complement* of  $A_{11}$  is defined as the  $(n - 1) \times (n - 1)$  matrix

$$S = A_{2:n,2:n} - \frac{1}{A_{11}} A_{2:n,1} A_{2:n,1}^T$$

- if  $A$  is positive definite, then  $S$  is positive definite

to see this, take any  $x \neq 0$  and define  $y = -(A_{2:n,1}^T x)/A_{11}$ ; then

$$x^T S x = \begin{bmatrix} y \\ x \end{bmatrix}^T \begin{bmatrix} A_{11} & A_{2:n,1}^T \\ A_{2:n,1} & A_{2:n,2:n} \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} > 0$$

because  $A$  is positive definite

## Singular positive semidefinite matrices

- we have seen that positive definite matrices are nonsingular (page 11-4)
- if  $A$  is positive semidefinite, but not positive definite, then it is singular

to see this, suppose  $A$  is positive semidefinite but not positive definite

- there exists a nonzero  $x$  with  $x^T Ax = 0$
- since  $A$  is positive semidefinite the following function is nonnegative:

$$\begin{aligned} f(t) &= (x - tAx)^T A(x - tAx) \\ &= x^T Ax - 2tx^T A^2 x + t^2 x^T A^3 x \\ &= -2t\|Ax\|^2 + t^2 x^T A^3 x \end{aligned}$$

- $f(t) \geq 0$  for all  $t$  is only possible if  $\|Ax\| = 0$ , i.e.,  $Ax = 0$

hence there exists a nonzero  $x$  with  $Ax = 0$

## Exercises

- show that if  $A \in \mathbf{R}^{n \times n}$  is positive semidefinite, then

$$B^T A B$$

is positive semidefinite for any  $B \in \mathbf{R}^{n \times m}$

- show that if  $A \in \mathbf{R}^{n \times n}$  is positive definite, then

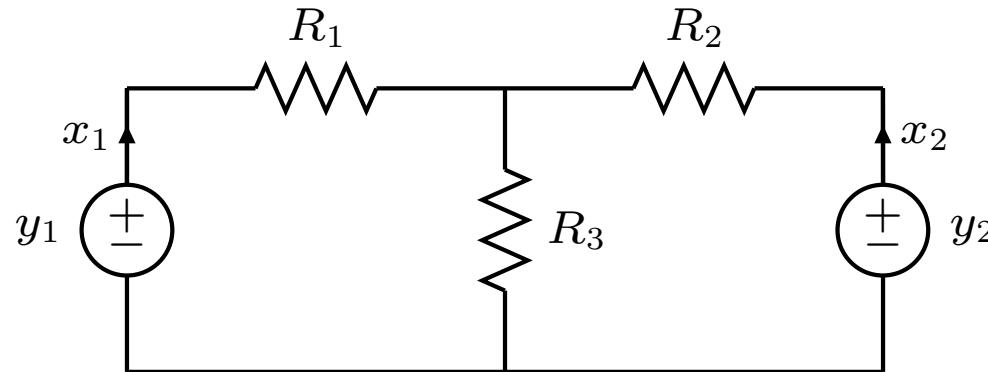
$$B^T A B$$

is positive definite for any  $B \in \mathbf{R}^{n \times m}$  with linearly independent columns

# Outline

- positive definite matrices
- **examples**
- Cholesky factorization
- complex positive definite matrices
- kernel methods
- least squares classification

## Exercise: resistor circuit



$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} R_1 + R_3 & R_3 \\ R_3 & R_2 + R_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

show that

$$A = \begin{bmatrix} R_1 + R_3 & R_3 \\ R_3 & R_2 + R_3 \end{bmatrix}$$

is positive definite if  $R_1, R_2, R_3$  are positive

# Solution

## Solution from physics

- $x^T Ax = y^T x$  is the power delivered by sources, dissipated by resistors
- power dissipated by the resistors is positive unless both currents are zero

## Algebraic solution

$$\begin{aligned}x^T Ax &= (R_1 + R_3)x_1^2 + 2R_3x_1x_2 + (R_2 + R_3)x_2^2 \\&= R_1x_1^2 + R_2x_2^2 + R_3(x_1 + x_2)^2 \\&\geq 0\end{aligned}$$

and  $x^T Ax = 0$  only if  $x_1 = x_2 = 0$

## Gram matrix

recall the definition of *Gram matrix* of a matrix  $B$  (page 4-21):

$$A = B^T B$$

- every Gram matrix is positive semidefinite

$$x^T A x = x^T B^T B x = \|Bx\|^2 \geq 0 \quad \forall x$$

- a Gram matrix is positive definite if

$$x^T A x = x^T B^T B x = \|Bx\|^2 > 0 \quad \forall x \neq 0$$

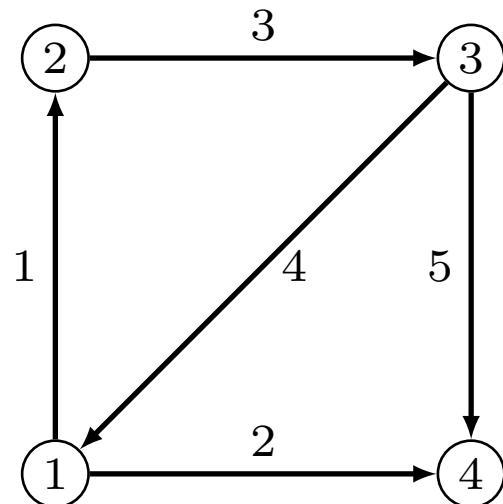
in other words,  $B$  has linearly independent columns

# Graph Laplacian

recall definition of node-arc incidence matrix of a directed graph (p. 3-29)

$$B_{ij} = \begin{cases} 1 & \text{if arc } j \text{ enters node } i \\ -1 & \text{if arc } j \text{ leaves node } i \\ 0 & \text{otherwise} \end{cases}$$

assume there are no self-loops and at most one arc between any two nodes



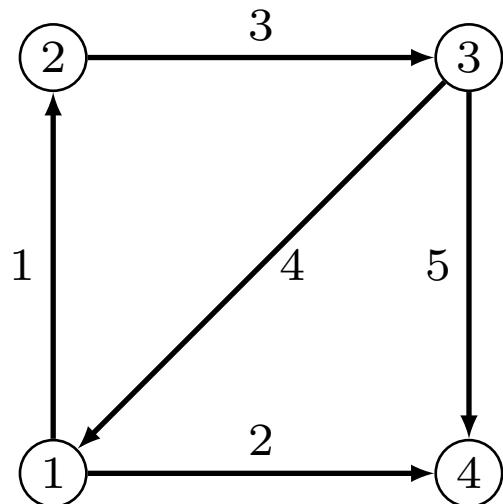
$$B = \begin{bmatrix} -1 & -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

# Graph Laplacian

the positive semidefinite matrix  $A = BB^T$  is called the *Laplacian*

$$A_{ij} = \begin{cases} \text{degree of node } i & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and there is an arc } i \rightarrow j \text{ or } j \rightarrow i \\ 0 & \text{otherwise} \end{cases}$$

the degree of a node is the number of arcs incident to it



$$A = BB^T = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

## Laplacian function

recall the interpretation of matrix-vector multiplication with  $B^T$  (p. 3-31)

- if  $y$  is vector of node potentials, then  $B^T y$  contains potential differences:

$$(B^T y)_j = y_k - y_l \quad \text{if arc } j \text{ goes from node } l \text{ to } k$$

- $y^T A y = y^T B B^T y$  is the sum of squared potential differences

$$y^T A y = \|B^T y\|^2 = \sum_{\text{arcs } i \rightarrow j} (y_j - y_i)^2$$

**Example:** for the graph on the previous page

$$y^T A y = (y_2 - y_1)^2 + (y_4 - y_1)^2 + (y_3 - y_2)^2 + (y_1 - y_3)^2 + (y_4 - y_3)^2$$

# Variance and covariance of random variables

let  $a = (a_1, a_2, \dots, a_n)$  be a random  $n$ -vector, with

$$\mu_i = \mathbf{E} a_i, \quad s_{ij} = \mathbf{E} ((a_i - \mu_i)(a_j - \mu_j))$$

( $\mathbf{E}$  denotes expectation)

- $\mu_i$  is the *mean* or *expected value* of  $a_i$
- $s_{ii}$  is the *variance* and  $\sqrt{s_{ii}}$  is the *standard deviation* of  $a_i$
- $s_{ij}$ , for  $i \neq j$ , is the *covariance* of  $a_i$  and  $a_j$

**Note:**

these terms have a different meaning for (non-random) vectors (see p.2-9)

# Covariance matrix

covariance matrix (or variance-covariance matrix) has  $i, j$  element  $s_{ij}$ :

$$\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{bmatrix} = \mathbf{E} \left( \begin{bmatrix} a_1 - \mu_1 \\ a_2 - \mu_2 \\ \vdots \\ a_n - \mu_n \end{bmatrix} \begin{bmatrix} a_1 - \mu_1 \\ a_2 - \mu_2 \\ \vdots \\ a_n - \mu_n \end{bmatrix}^T \right)$$
$$= \mathbf{E} ((a - \mu)(a - \mu)^T)$$

- on the right-hand side, expectation of a matrix applies element-wise
- $\mu$  is the vector of means:

$$\mu = (\mu_1, \mu_2, \dots, \mu_n) = (\mathbf{E} a_1, \mathbf{E} a_2, \dots, \mathbf{E} a_n)$$

# Positive semidefiniteness

every covariance matrix is positive semidefinite: for any  $x$ ,

$$\begin{aligned} & \left[ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right]^T \left[ \begin{array}{cccc} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{array} \right] \left[ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right] \\ &= \mathbf{E} \left( \left[ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right]^T \left[ \begin{array}{c} a_1 - \mu_1 \\ a_2 - \mu_2 \\ \vdots \\ a_n - \mu_n \end{array} \right] \left[ \begin{array}{c} a_1 - \mu_1 \\ a_2 - \mu_2 \\ \vdots \\ a_n - \mu_n \end{array} \right]^T \left[ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right] \right) \\ &= \mathbf{E} (x_1(a_1 - \mu_1) + x_2(a_2 - \mu_2) + \cdots + x_n(a_n - \mu_n))^2 \\ &\geq 0 \end{aligned}$$

# Outline

- positive definite matrices
- examples
- **Cholesky factorization**
- complex positive definite matrices
- kernel methods
- least squares classification

## Cholesky factorization

every positive definite matrix  $A \in \mathbf{R}^{n \times n}$  can be factored as

$$A = R^T R$$

where  $R$  is upper triangular with positive diagonal elements

- complexity of computing  $R$  is  $(1/3)n^3$  flops
- $R$  is called the *Cholesky factor* of  $A$
- can be interpreted as ‘square root’ of a positive define matrix
- gives a practical method for testing positive definiteness

# Cholesky factorization algorithm

$$\begin{bmatrix} A_{11} & A_{1,2:n} \\ A_{2:n,1} & A_{2:n,2:n} \end{bmatrix} = \begin{bmatrix} R_{11} & 0 \\ R_{1,2:n}^T & R_{2:n,2:n}^T \end{bmatrix} \begin{bmatrix} R_{11} & R_{1,2:n} \\ 0 & R_{2:n,2:n} \end{bmatrix}$$
$$= \begin{bmatrix} R_{11}^2 & R_{11}R_{1,2:n} \\ R_{11}R_{1,2:n}^T & R_{1,2:n}^T R_{1,2:n} + R_{2:n,2:n}^T R_{2:n,2:n} \end{bmatrix}$$

1. compute first row of  $R$ :

$$R_{11} = \sqrt{A_{11}}, \quad R_{1,2:n} = \frac{1}{R_{11}} A_{1,2:n}$$

2. compute  $2,2$  block  $R_{2:n,2:n}$  from

$$A_{2:n,2:n} - R_{1,2:n}^T R_{1,2:n} = R_{2:n,2:n}^T R_{2:n,2:n}$$

this is a Cholesky factorization of order  $n - 1$

## Discussion

the algorithm works for positive definite  $A$  of size  $n \times n$

- step 1: if  $A$  is positive definite then  $A_{11} > 0$
- step 2: if  $A$  is positive definite, then

$$A_{2:n,2:n} - R_{1,2:n}^T R_{1,2:n} = A_{2:n,2:n} - \frac{1}{A_{11}} A_{2:n,1} A_{2:n,1}^T$$

is positive definite (see page 11-5)

- hence the algorithm works for  $n = m$  if it works for  $n = m - 1$
- it obviously works for  $n = 1$ ; therefore it works for all  $n$

## Example

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} R_{11} & 0 & 0 \\ R_{12} & R_{22} & 0 \\ R_{13} & R_{23} & R_{33} \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}$$

- first row of  $R$

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & R_{22} & 0 \\ -1 & R_{23} & R_{33} \end{bmatrix} \begin{bmatrix} 5 & 3 & -1 \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix}$$

- second row of  $R$

$$\begin{bmatrix} 18 & 0 \\ 0 & 11 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \end{bmatrix} \begin{bmatrix} 3 & -1 \end{bmatrix} = \begin{bmatrix} R_{22} & 0 \\ R_{23} & R_{33} \end{bmatrix} \begin{bmatrix} R_{22} & R_{23} \\ 0 & R_{33} \end{bmatrix}$$

$$\begin{bmatrix} 9 & 3 \\ 3 & 10 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 1 & R_{33} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 0 & R_{33} \end{bmatrix}$$

- third column of  $R$

$$10 - 1 = R_{33}^2, \quad R_{33} = 3$$

## Conclusion

$$\begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 0 \\ 3 & 3 & 0 \\ -1 & 1 & 3 \end{bmatrix} \begin{bmatrix} 5 & 3 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$

# Solving equations with positive definite $A$

solve  $Ax = b$  with  $A$  a positive definite  $n \times n$  matrix

## Algorithm

- factor  $A$  as  $A = R^T R$
- solve  $R^T R x = b$ 
  - solve  $R^T y = b$  by forward substitution
  - solve  $R x = y$  by back substitution

**Complexity:**  $(1/3)n^3 + 2n^2 \approx (1/3)n^3$  flops

- factorization:  $(1/3)n^3$
- forward and backward substitution:  $2n^2$

## Cholesky factorization of Gram matrix

- suppose  $B$  is an  $m \times n$  matrix with linearly independent columns
- the Gram matrix  $A = B^T B$  is positive definite (p. 4-21)

two methods for computing the Cholesky factor of  $A$ , given  $B$

1. compute  $A = B^T B$ , then Cholesky factorization of  $A$

$$A = R^T R$$

2. compute QR factorization  $B = QR$ ; since

$$A = B^T B = R^T Q^T Q R = R^T R$$

the matrix  $R$  is the Cholesky factor of  $A$

## Example

$$B = \begin{bmatrix} 3 & -6 \\ 4 & -8 \\ 0 & 1 \end{bmatrix}, \quad A = B^T B = \begin{bmatrix} 25 & -50 \\ -50 & 101 \end{bmatrix}$$

1. Cholesky factorization:

$$A = \begin{bmatrix} 5 & 0 \\ -10 & 1 \end{bmatrix} \begin{bmatrix} 5 & -10 \\ 0 & 1 \end{bmatrix}$$

2. QR factorization

$$B = \begin{bmatrix} 3 & -6 \\ 4 & -8 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3/5 & 0 \\ 4/5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & -10 \\ 0 & 1 \end{bmatrix}$$

## Comparison of the two methods

**Numerical stability:** QR factorization method is more stable

- see the example on page 8-17
- QR method computes  $R$  without ‘squaring’  $B$  (*i.e.*, forming  $B^T B$ )
- this is important when the columns of  $B$  are ‘almost’ linearly dependent

## Complexity

- method 1: cost of symmetric product  $B^T B$  plus Cholesky factorization

$$mn^2 + (1/3)n^3 \text{ flops}$$

- method 2:  $2mn^2$  flops for QR factorization
- method 1 is faster but only by a factor of at most two (if  $m \gg n$ )

# Sparse positive definite matrices

## Cholesky factorization of dense matrices

- $(1/3)n^3$  flops
- on a standard computer: a few seconds or less, for  $n$  up to several 1000

## Cholesky factorization of sparse matrices

- if  $A$  is very sparse,  $R$  is often (but not always) sparse
- if  $R$  is sparse, the cost of the factorization is much less than  $(1/3)n^3$
- exact cost depends on  $n$ , number of nonzero elements, sparsity pattern
- very large sets of equations ( $n \sim 10^6$ ) are solved by exploiting sparsity

## Sparse Cholesky factorization

if  $A$  is sparse and positive definite, it is usually factored as

$$A = PR^T R P^T$$

$P$  permutation matrix;  $R$  upper triangular with positive diagonal elements

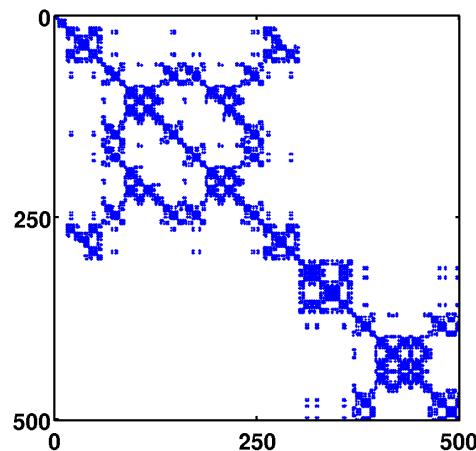
**Interpretation:** we permute the rows and columns of  $A$  and factor

$$P^T A P = R^T R$$

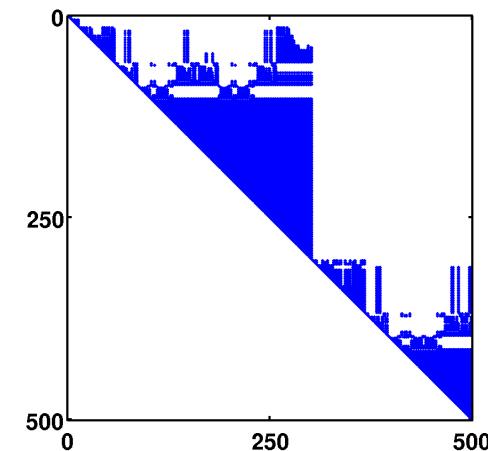
- choice of  $P$  greatly affects the sparsity  $R$
- many heuristic methods (that we don't cover) for choosing  $P$

# Example

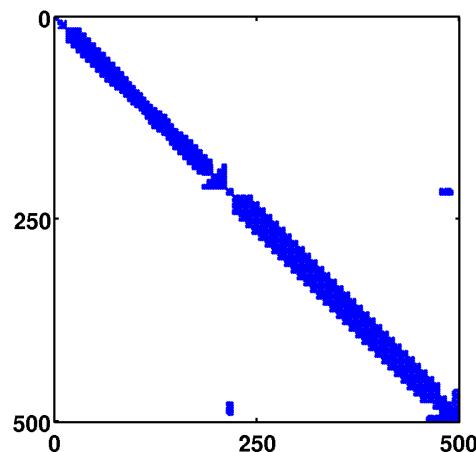
sparsity pattern of  $A$



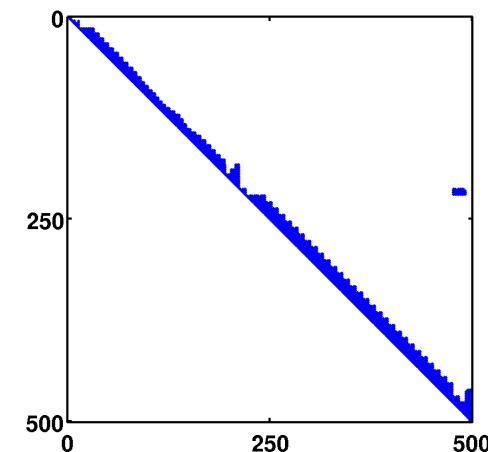
Cholesky factor of  $A$



pattern of  $P^T AP$



Cholesky factor of  $P^T AP$



# Solving sparse positive definite equations

solve  $Ax = b$  with  $A$  a sparse positive definite matrix

## Algorithm

1. compute sparse Cholesky factorization  $A = PR^T R P^T$
2. permute right-hand side:  $c := P^T b$
3. solve  $R^T y = c$  by forward substitution
4. solve  $Rz = y$  by back substitution
5. permute solution:  $x := Pz$

# Outline

- positive definite matrices
- examples
- Cholesky factorization
- **complex positive definite matrices**
- kernel methods
- least squares classification

## Quadratic form

suppose  $A$  is  $n \times n$  and Hermitian ( $A_{ij} = \bar{A}_{ji}$ )

$$\begin{aligned} x^H Ax &= \sum_{i=1}^n \sum_{j=1}^n A_{ij} \bar{x}_i x_j \\ &= \sum_{i=1}^n A_{ii} |x_i|^2 + \sum_{i>j} (A_{ij} \bar{x}_i x_j + \bar{A}_{ij} x_i \bar{x}_j) \\ &= \sum_{i=1}^n A_{ii} |x_i|^2 + 2 \operatorname{Re} \sum_{i>j} A_{ij} \bar{x}_i x_j \end{aligned}$$

note that  $x^H Ax$  is real for all  $x \in \mathbf{C}^n$

## Complex positive definite matrices

- a Hermitian  $n \times n$  matrix  $A$  is positive semidefinite if

$$x^H A x \geq 0 \quad \text{for all } x \in \mathbf{C}^n$$

- a Hermitian  $n \times n$  matrix  $A$  is positive definite if

$$x^H A x > 0 \quad \text{for all nonzero } x \in \mathbf{C}^n$$

### Cholesky factorization

every positive definite matrix  $A \in \mathbf{C}^{n \times n}$  can be factored as

$$A = R^H R$$

where  $R$  is upper triangular with positive real diagonal elements

# Outline

- positive definite matrices
- examples
- Cholesky factorization
- complex positive definite matrices
- **kernel methods**
- least squares classification

## Regularized least squares model fitting

we revisit the problem of fitting a linearly parameterized model (p. 9-4)

$$\begin{aligned}\hat{f}(z) &= \hat{\theta}_1 f_1(z) + \hat{\theta}_2 f_2(z) + \cdots + \hat{\theta}_p f_p(z) \\ &= \hat{\theta}^T F(z)\end{aligned}$$

$F(z) = (f_1(z), \dots, f_p(z))$  is a  $p$ -vector of basis functions  $f_i(z)$

**Regularized least squares model fitting:** choose for  $\hat{\theta}$  the solution of

$$\text{minimize } \sum_{i=1}^N (\theta^T F(x_i) - y_i)^2 + \lambda \sum_{i=1}^p \theta_i^2$$

- $(x_1, y_1), \dots, (x_N, y_n)$  are  $N$  data points
- regularization helps avoid over-fitting (and handle problems with  $N < p$ )

## Regularized least squares problem in matrix notation

$$\text{minimize } \|A\theta - y\|^2 + \lambda\|\theta\|^2$$

$A$  has size  $N \times p$  (# data points  $\times$  # basis functions)

$$A = \begin{bmatrix} F(x_1)^T \\ F(x_2)^T \\ \vdots \\ F(x_N)^T \end{bmatrix} = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_p(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_p(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_N) & f_2(x_N) & \cdots & f_p(x_N) \end{bmatrix}$$

- we discuss methods for problems with  $N \ll p$  ( $A$  is very wide)
- equivalent ‘stacked’ least squares problem (p. 9-12) has size  $(p + N) \times p$
- QR factorization method may be too expensive when  $N \ll p$

## Solution of regularized LS problem

from the normal equations:

$$\hat{\theta} = (A^T A + \lambda I)^{-1} A^T y = A^T (A A^T + \lambda I)^{-1} y$$

- second expression follows from the property

$$(A^T A + \lambda I)^{-1} A^T = A^T (A A^T + \lambda I)^{-1}$$

this is easily proved, by writing it as  $A^T (A A^T + \lambda I) = (A^T A + \lambda I) A^T$

- from the second expression for  $\hat{\theta}$  and the definition of  $A$ ,

$$\hat{f}(z) = \hat{\theta}^T F(z) = w^T A F(z) = \sum_{i=1}^N w_i F(x_i)^T F(z)$$

where  $w = (A A^T + \lambda I)^{-1} y$

# Algorithm

1. compute the  $N \times N$  matrix  $Q = AA^T$ , which has elements

$$Q_{ij} = F(x_i)^T F(x_j), \quad i, j = 1, \dots, N$$

2. use a Cholesky factorization to solve the equation

$$(Q + \lambda I)w = y$$

- $\hat{\theta} = A^T w$  is not needed;  $w$  is sufficient to evaluate the function  $\hat{f}$ :

$$\hat{f}(z) = \sum_{i=1}^N w_i F(x_i)^T F(z)$$

- complexity:  $(1/3)N^3$  flops plus cost of computing  $Q$

## Example: multivariate polynomials

$\hat{f}(z)$  is polynomial of degree  $d$  (or less) in  $n$  variables  $z = (z_1, \dots, z_n)$

- $\hat{f}(z)$  is a linear combination of all possible *monomials*

$$z_1^{k_1} z_2^{k_2} \cdots z_n^{k_n}$$

where  $k_1, \dots, k_n$  are nonnegative integers with  $k_1 + k_2 + \cdots + k_n \leq d$

- number of different monomials is

$$\binom{n+d}{n} = \frac{(n+d)!}{n! d!}$$

**Example:** for  $n = 2, d = 3$  there are 10 monomials

$$1, \quad z_1, \quad z_2, \quad z_1^2, \quad z_1 z_2, \quad z_2^2, \quad z_1^3, \quad z_1^2 z_2, \quad z_1 z_2^2, \quad z_2^3$$

## Multinomial formula

$$(z_0 + z_1 + \cdots + z_n)^d = \sum_{k_0 + \cdots + k_n = d} \frac{(d+1)!}{k_0! k_1! \cdots k_n!} z_0^{k_0} z_1^{k_1} \cdots z_n^{k_n}$$

sum is over all nonnegative integers  $k_0, k_1, \dots, k_n$  with sum  $d$

- setting  $z_0 = 1$  gives

$$(1 + z_1 + z_2 + \cdots + z_n)^d = \sum_{k_1 + \cdots + k_n \leq d} c_{k_1 k_2 \cdots k_n} z_1^{k_1} z_2^{k_2} \cdots z_n^{k_n}$$

- sum includes all monomials of degree  $d$  or less with variables  $z_1, \dots, z_n$
- coefficient  $c_{k_1 k_2 \cdots k_n}$  is defined as

$$c_{k_1 k_2 \cdots k_n} = \frac{(d+1)!}{k_0! k_1! k_2! \cdots k_n!} \quad \text{with} \quad k_0 = d - k_1 - \cdots - k_n$$

## Vector of monomials

write polynomial of degree  $d$  or less, with variables  $z \in \mathbf{R}^n$ , as

$$f(z) = \theta^T F(z)$$

- $F(z)$  is vector of basis functions

$$\sqrt{c_{k_1 \dots k_n}} z_1^{k_1} z_2^{k_2} \dots z_n^{k_n} \quad \text{for all } k_1 + k_2 + \dots + k_n \leq d$$

- length of  $F(z)$  is  $p = (n+d)!/(n!d!)$
- multinomial formula gives simple formula for inner products  $F(u)^T F(v)$ :

$$\begin{aligned} F(u)^T F(v) &= \sum_{k_1+\dots+k_n \leq d} c_{k_1 k_2 \dots k_n} (u_1^{k_1} \dots u_n^{k_n})(v_1^{k_1} \dots v_n^{k_n}) \\ &= (1 + u_1 v_1 + \dots + u_n v_n)^d \end{aligned}$$

- only  $2n+1$  flops needed for inner product of length  $p = (n+d)!/(n!d!)$

## Example

vector of monomials of degree  $d = 3$  or less in  $n = 2$  variables

$$F(u)^T F(v) = \begin{bmatrix} 1 \\ \sqrt{3}u_1 \\ \sqrt{3}u_2 \\ \sqrt{3}u_1^2 \\ \sqrt{6}u_1u_2 \\ \sqrt{3}u_2^2 \\ u_1^3 \\ \sqrt{3}u_1^2u_2 \\ \sqrt{3}u_1u_2^2 \\ u_2^3 \end{bmatrix}^T \begin{bmatrix} 1 \\ \sqrt{3}v_1 \\ \sqrt{3}v_2 \\ \sqrt{3}v_1^2 \\ \sqrt{6}v_1v_2 \\ \sqrt{3}v_2^2 \\ v_1^3 \\ \sqrt{3}v_1^2v_2 \\ \sqrt{3}v_1v_2^2 \\ v_2^3 \end{bmatrix}$$
$$= (1 + u_1v_1 + u_2v_2)^3$$

## Least squares fitting of multivariate polynomials

to fit polynomial of degree  $d$  or less to  $N$  data points  $(x_i, y_i)$ , with  $x_i \in \mathbf{R}^n$

**Algorithm** (page 11-35)

1. compute the  $N \times N$  matrix  $Q$  with elements

$$Q_{ij} = K(x_i, x_j) \quad \text{where } K(u, v) = (1 + u^T v)^d$$

2. use a Cholesky factorization to solve the equation  $(Q + \lambda I)w = y$

the fitted polynomial is

$$\hat{f}(z) = \sum_{i=1}^N w_i K(x_i, z) = \sum_{i=1}^N w_i (1 + x_i^T z)^d$$

algorithm has complexity  $nN^2 + (1/3)N^3$  flops

# Kernel methods

**Kernel function:** a generalized inner product  $K(u, v)$

- $K(u, v)$  is inner product of vectors of basis functions  $F(u)$  and  $F(v)$
- $F(u)$  may be infinite-dimensional
- kernel methods work with  $K(u, v)$  directly, do not require  $F(u)$

## Examples

- the polynomial kernel function  $K(u, v) = (1 + u^T v)^d$
- the *Gaussian Radial Basis Function* kernel

$$K(u, v) = \exp\left(-\frac{\|u - v\|^2}{2\sigma^2}\right)$$

- kernels exist for computing with graphs, texts, strings of symbols, . . .

# Outline

- positive definite matrices
- examples
- Cholesky factorization
- complex positive definite matrices
- kernel methods
- **least squares classification**

## Example: handwritten digit classification

- MNIST data set used in homework 1
- $28 \times 28$  images of handwritten digits ( $n = 28^2 = 784$  pixels)
- data set contains 60000 training examples; 10000 test examples
- we use least squares to compute ten binary classifiers
- each classifier distinguishes one digit from the rest
- we then combine the binary classifiers to get a multiclass classifier
- we will use 10000 of the 60000 training examples ( $\approx 1000$  per digit)

## Binary (two-way) classification

this is a data fitting problem (page 9-2) with two values of the outcome  $y$

$$y \approx f(x), \quad f : \mathbf{R}^n \rightarrow \{-1, +1\}$$

- outcome represents two categories (true/false, spam/not spam, . . . )
- the model  $\hat{f} : \mathbf{R}^n \rightarrow \{-1, +1\}$  is called a binary or Boolean *classifier*

we use a model of the form

$$\begin{aligned}\hat{f}(x) &= \text{sign} \left( \hat{\theta}_1 f_1(x) + \hat{\theta}_2 f_2(x) + \cdots + \hat{\theta}_p f_p(x) \right) \\ &= \begin{cases} 1 & \text{if } \hat{\theta}_1 f_1(x) + \cdots + \hat{\theta}_p f_p(x) \geq 0 \\ -1 & \text{if } \hat{\theta}_1 f_1(x) + \cdots + \hat{\theta}_p f_p(x) < 0 \end{cases}\end{aligned}$$

functions  $f_i(x)$  are basis functions;  $\hat{\theta}_i$  are model parameters

## Least squares binary classifier

compute model parameters  $\hat{\theta}$  by solving a regularized least squares problem

$$\text{minimize} \quad \sum_{i=1}^N (\theta^T F(x_i) - y_i)^2 + \lambda \sum_{i=1}^p \theta_i^2$$

- $F(z) = (f_1(z), \dots, f_p(z))$  is vector of basis functions
- $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$  are given data points with  $y_i \in \{-1, 1\}$

## Least squares classifier (using algorithm on page 11-35)

$$\hat{f}(z) = \text{sign} \left( \sum_{i=1}^N w_i K(x_i, z) \right)$$

- $w$  is the solution of  $(Q + \lambda I)w = y$
- $Q$  has elements  $Q_{ij} = F(x_i)^T F(x_j) = K(x_i, x_j)$

# Multiclass classifier

can be viewed as data fitting problem with  $K$  values of the outcome  $y$

$$y \approx f(x), \quad f : \mathbf{R}^n \rightarrow \{1, 2, \dots, K\}$$

## Via binary classification

- compute  $K$  binary classifiers

$$\hat{f}^{(k)}(z) = \text{sign} \left( \sum_{i=1}^N w_i^{(k)} K(x_i, z) \right)$$

- $\hat{f}^{(k)}$  classifies class  $k$  (outcome +1) versus the rest (outcome -1)
- define multiclass classifier as

$$\hat{f}(x) = \operatorname{argmax}_{k=1, \dots, K} \left( \sum_{i=1}^N w_i^{(k)} K(x_i, z) \right)$$

## Handwritten digit classification

- data set contains  $N = 10000$  vectors  $x_i$  of length  $n = 784$
- we use the polynomial kernel with degree  $d = 3$

$$K(u, v) = (1 + u^T v)^3$$

hence  $F(x)$  has length  $p = (n + d)!/(n! d!) = 80,931,145$

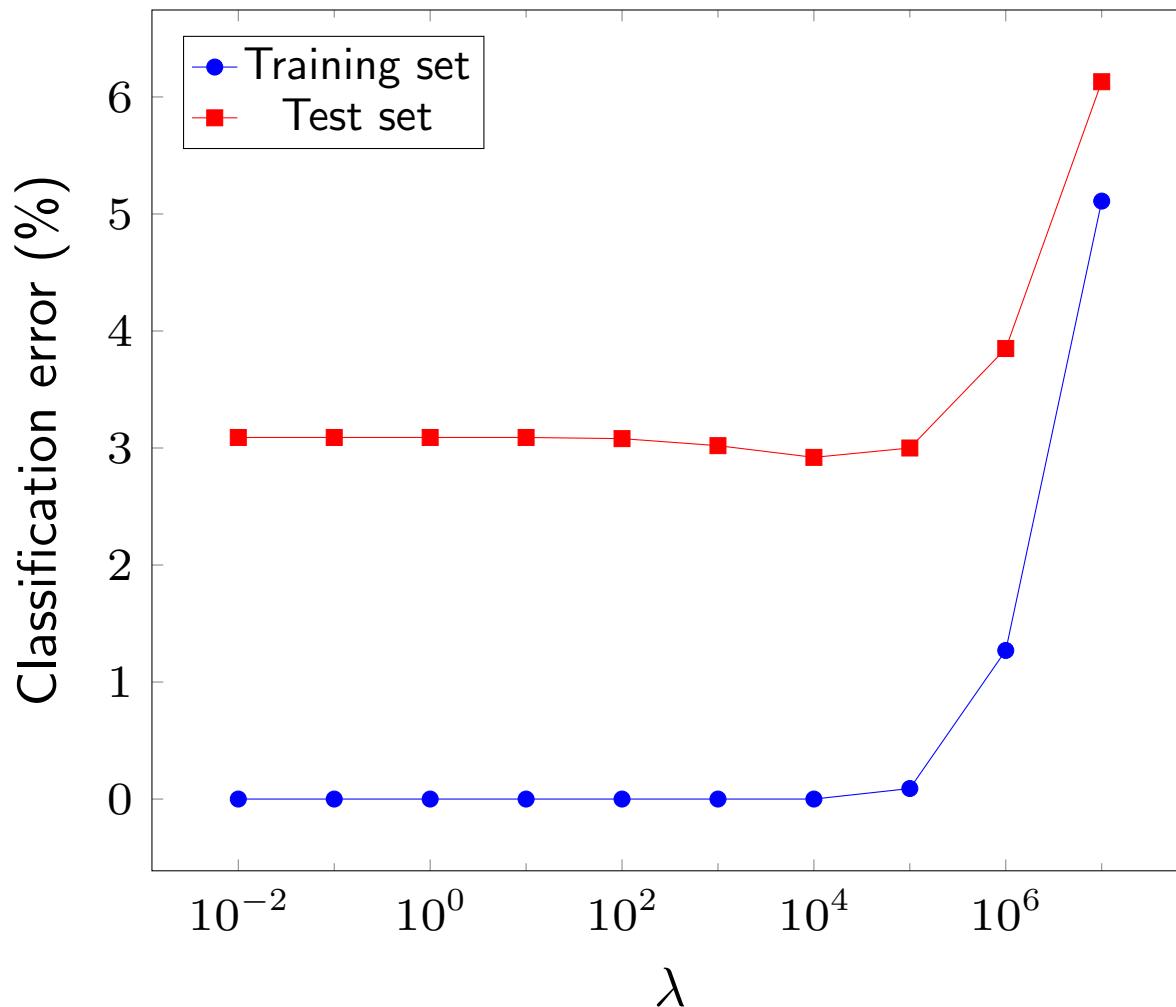
- to compute the classifier for digit  $k$  versus the rest, solve linear equation

$$(Q + \lambda I)w = y$$

with  $y_i = 1$  if  $x_i$  is an example of digit  $k$ , and  $y_i = -1$  otherwise

- only one Cholesky factorization is needed to compute the 10 classifiers

# Classification error



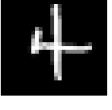
percentage of misclassified digits versus  $\lambda$

# Confusion matrix

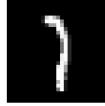
| Digit | Predicted digit |      |      |      |     |     |     |      |     |     | Total |
|-------|-----------------|------|------|------|-----|-----|-----|------|-----|-----|-------|
|       | 0               | 1    | 2    | 3    | 4   | 5   | 6   | 7    | 8   | 9   |       |
| 0     | 965             | 1    | 0    | 0    | 0   | 1   | 8   | 2    | 3   | 0   | 980   |
| 1     | 0               | 1127 | 2    | 1    | 1   | 0   | 2   | 1    | 1   | 0   | 1135  |
| 2     | 6               | 2    | 988  | 4    | 1   | 1   | 5   | 16   | 8   | 1   | 1032  |
| 3     | 0               | 0    | 7    | 973  | 0   | 12  | 0   | 8    | 6   | 4   | 1010  |
| 4     | 1               | 3    | 0    | 0    | 957 | 0   | 3   | 1    | 3   | 14  | 982   |
| 5     | 3               | 0    | 0    | 5    | 0   | 874 | 5   | 2    | 2   | 1   | 892   |
| 6     | 9               | 4    | 0    | 0    | 5   | 2   | 937 | 0    | 1   | 0   | 958   |
| 7     | 0               | 13   | 13   | 1    | 5   | 0   | 0   | 987  | 2   | 7   | 1028  |
| 8     | 3               | 1    | 3    | 11   | 4   | 4   | 3   | 5    | 934 | 6   | 974   |
| 9     | 3               | 4    | 2    | 7    | 13  | 3   | 1   | 6    | 4   | 966 | 1009  |
| All   | 990             | 1155 | 1015 | 1002 | 986 | 897 | 964 | 1028 | 964 | 999 | 10000 |

- multiclass classifier ( $\lambda = 10^4$ ) on 10000 test examples
- 292 digits are misclassified (2.9% error)

# Examples of misclassified digits

| Digit | Predicted digit   |   |   |  |   |   |   |   |   |   |
|-------|---|---|---|--|---|---|---|---|---|---|
|       | 0   | 1   | 2   | 3  | 4   | 5   | 6   | 7   | 8   | 9   |
| 0     |    |   |   |  |  |    |    |    |   |   |
| 1     |   |    |    |  |   |    |    |    |   |   |
| 2     |    |    |   |   |   |    |    |    |    |    |
| 3     |   |   |  |  |   |  |   |  |  |  |
| 4     |  |  |   |  |   |  |  |  |  |   |

# Examples of misclassified digits

| Digit | Predicted digit   |   |   |   |  |   |   |   |   |   |
|-------|---|---|---|---|--|---|---|---|---|---|
|       | 0   | 1   | 2   | 3   | 4  | 5   | 6   | 7   | 8   | 9   |
| 5     |    |   |   |    |  |   |    |    |  |    |
| 6     |    |    |   |   |    |    |   |   |  |   |
| 7     |   |    |    |    |    |   |   |   |  |    |
| 8     |  |  |  |  |  |  |  |  |   |  |
| 9     |  |  |  |  |  |  |  |  |   |  |

# 12. Nonlinear least squares

- definition and examples
- Gauss-Newton method
- Levenberg-Marquardt method

# Nonlinear least squares

$$\text{minimize} \quad \sum_{i=1}^m f_i(x)^2 = \|f(x)\|^2$$

- $f_1(x), \dots, f_m(x)$  are differentiable functions of vector variable  $x$
- $f$  is a function from  $\mathbf{R}^n$  to  $\mathbf{R}^m$  with components  $f_i(x)$ :

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix}$$

- problem reduces to (linear) least squares if  $f(x) = Ax - b$

# Location from range measurements

- vector  $x$  represents unknown location in 2-D or 3-D
- we estimate  $x$  by measuring distances to known points  $a_1, \dots, a_m$ :

$$\rho_i = \|x - a_i\| + v_i, \quad i = 1, \dots, m$$

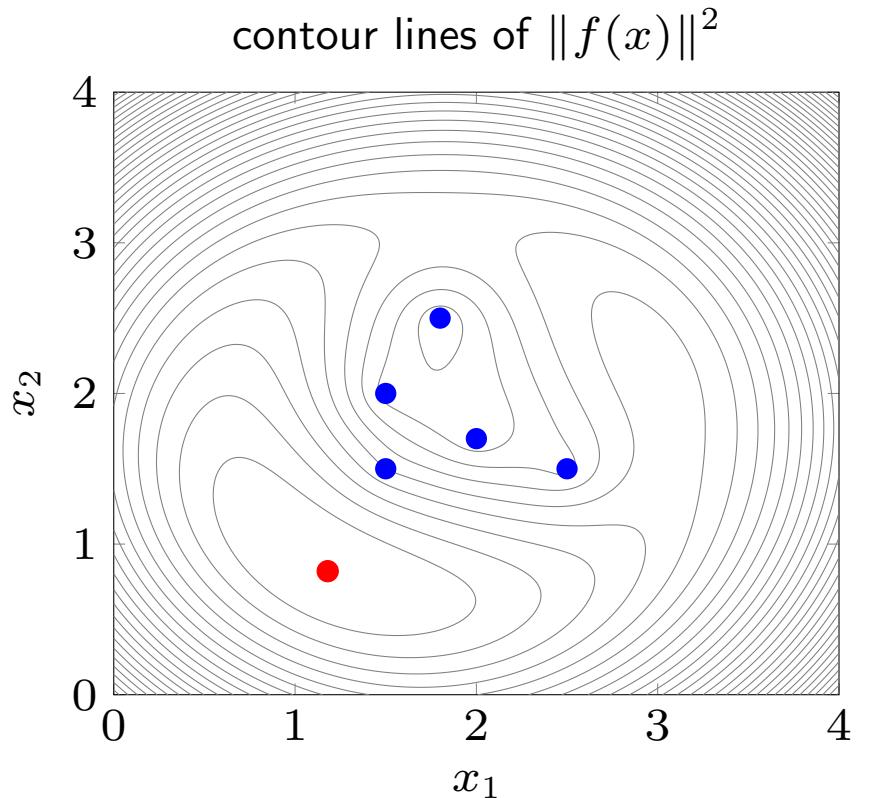
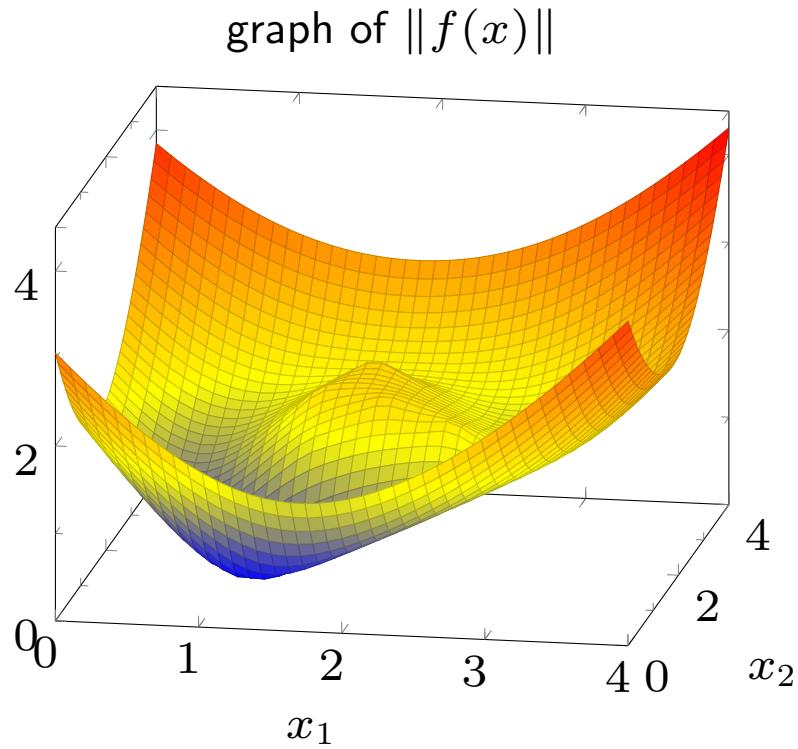
- $v_i$  is measurement error

**Nonlinear least squares estimate:** compute estimate  $\hat{x}$  by minimizing

$$\sum_{i=1}^m (\|x - a_i\| - \rho_i)^2$$

this is a nonlinear least squares problem with  $f_i(x) = \|x - a_i\| - \rho_i$

# Example



- correct position is  $(1, 1)$ ; the five points  $a_i$  are marked with blue dots
- red square marks (global) minimum at  $(1.18, 0.82)$

## Location from multiple camera views

- an object at location  $x$  (in 3-D) is viewed by  $m$  cameras
- the image of the object in the 2-D image plane of camera  $i$  is at location

$$y_i = \frac{1}{c_i^T x + d_i} (A_i x + b_i) + v_i$$

- $v_i$  is measurement error
- the parameters  $A_i, b_i, c_i, d_i$  describe camera  $i$  and its position ( $c_i^T x + d_i > 0$  for objects in front of the camera)
- goal is to determine 3-vector  $x$  from the  $m$  observations  $y_1, \dots, y_m$

**Nonlinear least squares estimate:** compute estimate  $\hat{x}$  by minimizing

$$\sum_{i=1}^m \left( \frac{1}{c_i^T x + d_i} (A_i x + b_i) - y_i \right)^2$$

# Model fitting

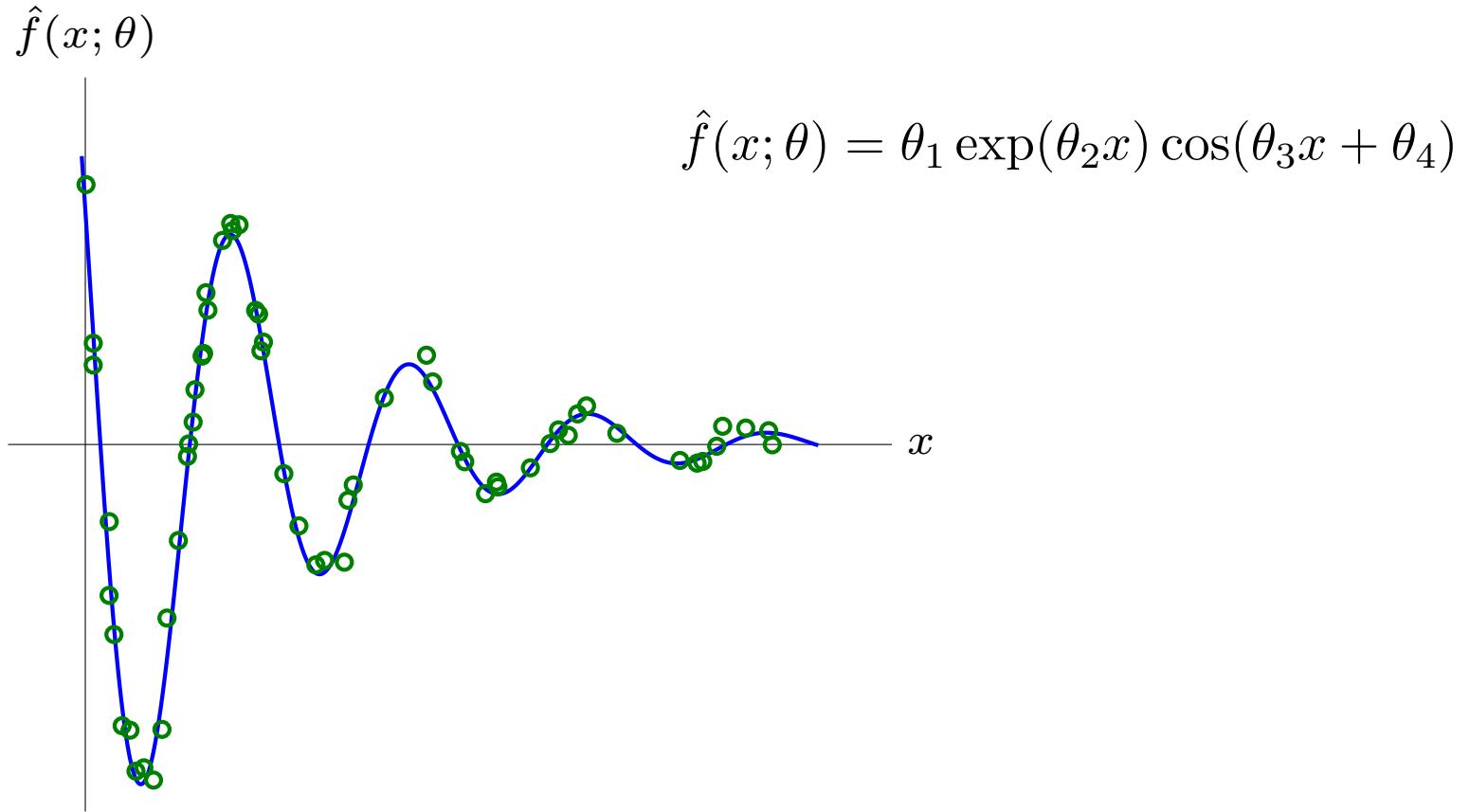
$$\text{minimize} \quad \sum_{i=1}^N (\hat{f}(x_i; \theta) - y_i)^2$$

- model  $\hat{f}(x; \theta)$  is parameterized by parameters  $\theta_1, \dots, \theta_p$
- $(x_1, y_1), \dots, (x_N, y_N)$  are data points
- the minimization is over the model parameters  $\theta$
- on page 9-4 we considered models that are linear in  $\theta$ :

$$\hat{f}(x; \theta) = \theta_1 f_1(x) + \dots + \theta_p f_p(x)$$

here we allow  $\hat{f}(x; \theta)$  to be a nonlinear function of  $\theta$

## Example



a nonlinear least squares problem with four variables  $\theta_1, \theta_2, \theta_3, \theta_4$ :

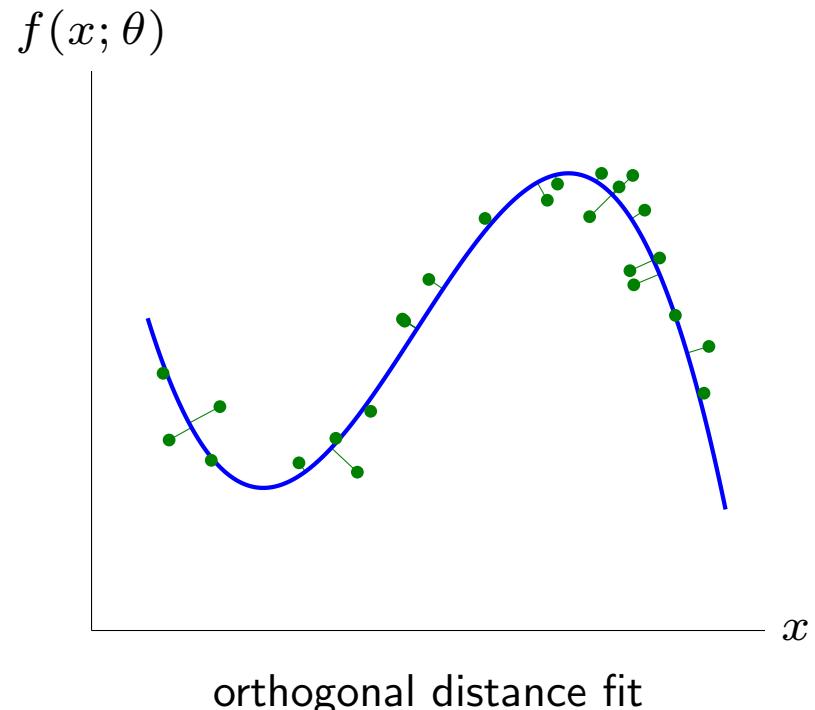
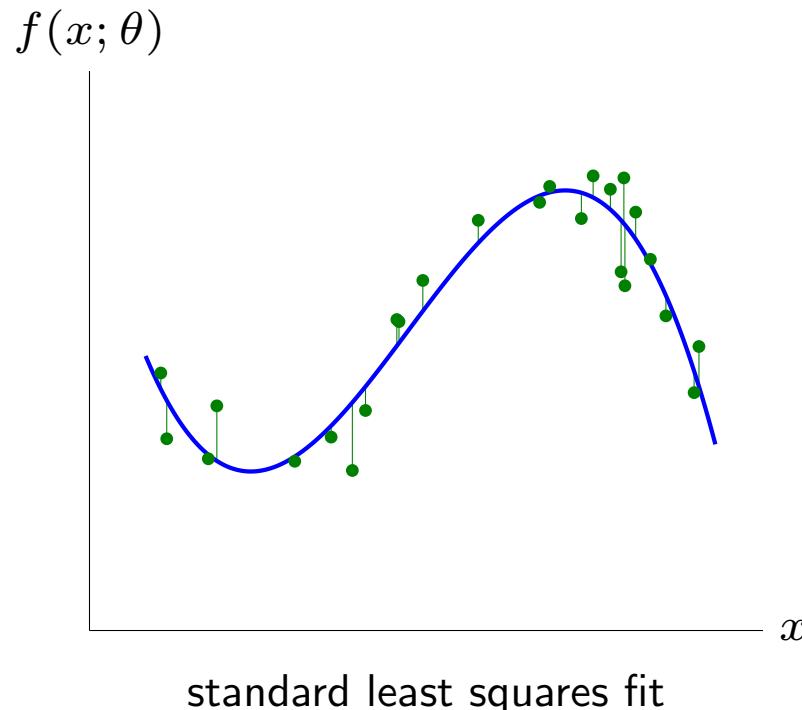
$$\text{minimize} \quad \sum_{i=1}^N (\theta_1 e^{\theta_2 x_i} \cos(\theta_3 x_i + \theta_4) - y_i)^2$$

# Orthogonal distance regression

minimize the mean square distance of data points to graph of  $\hat{f}(x; \theta)$

**Example:** orthogonal distance regression with cubic polynomial

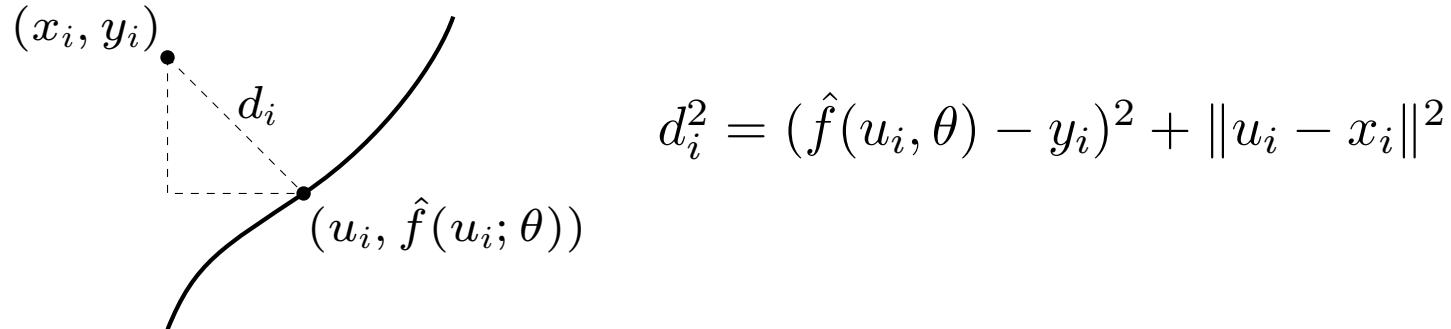
$$\hat{f}(x; \theta) = \theta_1 + \theta_2 x + \theta_3 x^2 + \theta_4 x^3$$



# Nonlinear least squares formulation

$$\text{minimize} \quad \sum_{i=1}^N \left( (\hat{f}(u_i; \theta) - y_i)^2 + \|u_i - x_i\|^2 \right)$$

- optimization variables are model parameters  $\theta$  and  $N$  points  $u_i$
- $i$ th term is squared distance of point  $(x_i, y_i)$  to point  $(u_i, \hat{f}(u_i, \theta))$



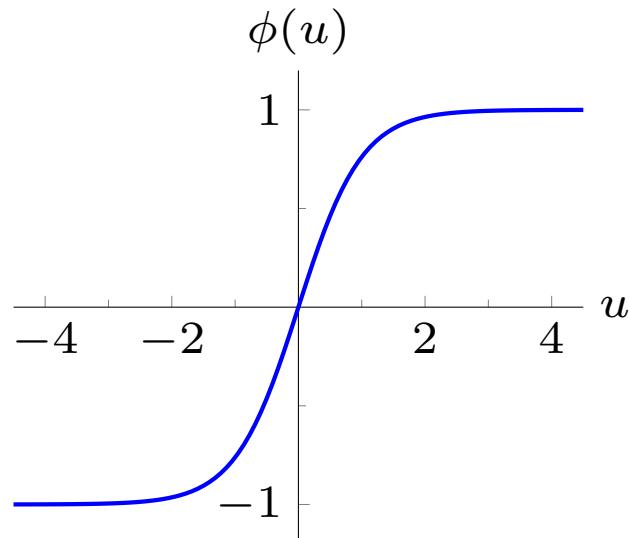
- minimizing over  $u_i$  gives squared distance of  $(x_i, y_i)$  to graph
- minimizing over  $u$  and  $\theta$  minimizes mean squared distance

# Binary classification

$$\hat{f}(x; \theta) = \text{sign}(\theta_1 f_1(x) + \theta_2 f_2(x) + \cdots + \theta_p f_p(x))$$

- in lecture 11 we computed  $\theta$  by solving a linear least squares problem
- better results are obtained by solving a nonlinear least squares problem

$$\text{minimize } \sum_{i=1}^N (\phi(\theta_1 f_1(x_i) + \cdots + \theta_p f_p(x_i)) - y_i)^2 + \lambda \sum_{i=1}^p \theta_i^2$$



- $(x_i, y_i)$  are data points
- $y_i \in \{-1, 1\}$  is class label
- $\phi(u)$  is the sigmoidal function

$$\phi(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

# Outline

- definition and examples
- **Gauss-Newton method**
- Levenberg-Marquardt method

# Gradient

**Gradient** of differentiable function  $g : \mathbf{R}^n \rightarrow \mathbf{R}$  at  $\hat{x} \in \mathbf{R}^n$  is

$$\nabla g(\hat{x}) = \left( \frac{\partial g}{\partial x_1}(\hat{x}), \frac{\partial g}{\partial x_2}(\hat{x}), \dots, \frac{\partial g}{\partial x_n}(\hat{x}) \right)$$

**Affine approximation** (linearization) of  $g$  around  $\hat{x}$  is

$$\begin{aligned}\hat{g}(x) &= g(\hat{x}) + \frac{\partial g}{\partial x_1}(\hat{x})(x_1 - \hat{x}_1) + \dots + \frac{\partial g}{\partial x_n}(\hat{x})(x_n - \hat{x}_n) \\ &= g(\hat{x}) + \nabla g(\hat{x})^T(x - \hat{x})\end{aligned}$$

(see page 1-31)

# Derivative matrix

**Derivative matrix** of differentiable function  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  at  $\hat{x} \in \mathbf{R}^n$ :

$$Df(\hat{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\hat{x}) & \frac{\partial f_1}{\partial x_2}(\hat{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\hat{x}) \\ \frac{\partial f_2}{\partial x_1}(\hat{x}) & \frac{\partial f_2}{\partial x_2}(\hat{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\hat{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(\hat{x}) & \frac{\partial f_m}{\partial x_2}(\hat{x}) & \cdots & \frac{\partial f_m}{\partial x_n}(\hat{x}) \end{bmatrix} = \begin{bmatrix} \nabla f_1(\hat{x})^T \\ \nabla f_2(\hat{x})^T \\ \vdots \\ \nabla f_m(\hat{x})^T \end{bmatrix}$$

**Affine approximation** (linearization) of  $f$  around  $\hat{x}$  is

$$\hat{f}(x) = f(\hat{x}) + Df(\hat{x})(x - \hat{x})$$

(see page 3-36)

## Gradient of nonlinear least squares cost

$$g(x) = \|f(x)\|^2 = \sum_{i=1}^m f_i(x)^2$$

- first derivative of  $g$  with respect to  $x_j$ :

$$\frac{\partial g}{\partial x_j}(\hat{x}) = 2 \sum_{i=1}^m f_i(\hat{x}) \frac{\partial f_i}{\partial x_j}(\hat{x})$$

- gradient of  $g$  at  $\hat{x}$ :

$$\nabla g(\hat{x}) = \begin{bmatrix} \frac{\partial g}{\partial x_1}(\hat{x}) \\ \vdots \\ \frac{\partial g}{\partial x_n}(\hat{x}) \end{bmatrix} = 2 \sum_{i=1}^m f_i(\hat{x}) \nabla f_i(\hat{x}) = 2 Df(\hat{x})^T f(\hat{x})$$

- necessary condition for optimality of  $\hat{x}$  is that  $\nabla g(\hat{x}) = 0$

## Gauss-Newton method

$$\text{minimize } g(x) = \|f(x)\|^2 = \sum_{i=1}^m f_i(x)^2$$

start at some initial guess  $x^{(1)}$ , and repeat for  $k = 1, 2, \dots$ :

- linearize  $f$  around  $x^{(k)}$ :

$$\hat{f}(x) = f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})$$

- substitute affine approximation  $\hat{f}$  for  $f$  in least squares problem:

$$\text{minimize } \|\hat{f}(x)\|^2$$

- define  $x^{(k+1)}$  as the solution of this (linear) least squares problem

## Gauss-Newton update

least squares problem solved in iteration  $k$  (with  $A = Df(x^{(k)})$ ):

$$\text{minimize} \quad \|f(x^{(k)}) + A(x - x^{(k)})\|^2$$

- if  $A$  has linearly independent columns, solution is given by

$$x^{(k+1)} = x^{(k)} - (A^T A)^{-1} A^T f(x^{(k)})$$

- Gauss-Newton step  $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$  is

$$\begin{aligned}\Delta x^{(k)} &= -(A^T A)^{-1} A^T f(x^{(k)}) \\ &= -\frac{1}{2}(A^T A)^{-1} \nabla g(x^{(k)})\end{aligned}$$

(using the expression for  $\nabla g$  on page 12-13)

## Predicted cost reduction in iteration $k$

- predicted cost function at  $x^{(k+1)}$ , based on approximation  $\hat{f}$ :

$$\begin{aligned}\|\hat{f}(x^{(k+1)})\|^2 &= \|f(x^{(k)}) + A\Delta x^{(k)}\|^2 \\ &= \|f(x^{(k)})\|^2 + 2f(x^{(k)})^T A\Delta x^{(k)} + \|A\Delta x^{(k)}\|^2 \\ &= \|f(x^{(k)})\|^2 - \|A\Delta x^{(k)}\|^2\end{aligned}$$

- if columns of  $Df(x^{(k)})$  are linearly independent and  $\Delta x^{(k)} \neq 0$ ,

$$\|\hat{f}(x^{(k+1)})\|^2 < \|f(x^{(k)})\|^2$$

- however, since  $\hat{f}$  is only a local approximation of  $f$ , it is possible that

$$\|f(x^{(k+1)})\|^2 > \|f(x^{(k)})\|^2$$

# Outline

- definition and examples
- Gauss-Newton method
- **Levenberg-Marquardt method**

# Levenberg-Marquardt method

addresses two difficulties in Gauss-Newton method:

- how to update  $x^{(k)}$  when columns of  $Df(x^{(k)})$  are linearly dependent
- what to do when the Gauss-Newton update does not reduce  $\|f(x)\|^2$

## Levenberg-Marquardt method:

compute  $x^{(k+1)}$  by solving a *regularized* least squares problem

$$\text{minimize} \quad \|f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})\|^2 + \lambda^{(k)}\|x - x^{(k)}\|^2$$

- with  $\lambda^{(k)} > 0$ , always has a unique solution (no condition on  $Df(x^{(k)})$ )
- parameter  $\lambda^{(k)} > 0$  controls size of update

## Levenberg-Marquardt update

least squares problem solved in iteration  $k$  (with  $A = Df(x^{(k)})$ ,  $\lambda = \lambda^{(k)}$ ):

$$\text{minimize} \quad \|f(x^{(k)}) + A(x - x^{(k)})\|^2 + \lambda \|x - x^{(k)}\|^2$$

- solution is given by

$$x^{(k+1)} = x^{(k)} - (A^T A + \lambda I)^{-1} A^T f(x^{(k)})$$

- Levenberg-Marquardt step  $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$  is

$$\begin{aligned}\Delta x^{(k)} &= -(A^T A + \lambda I)^{-1} A^T f(x^{(k)}) \\ &= -\frac{1}{2}(A^T A + \lambda I)^{-1} \nabla g(x^{(k)})\end{aligned}$$

- for  $\lambda = 0$  this is the Gauss-Newton step (if defined); for large  $\lambda$ ,

$$\Delta x^{(k)} \approx -\frac{1}{2\lambda} \nabla g(x^{(k)})$$

## Cost reduction

- predicted cost function at  $x^{(k+1)}$ , based on local approximation  $\hat{f}$ :

$$\begin{aligned}\|\hat{f}(x^{(k+1)})\|^2 &= \|f(x^{(k)}) + A\Delta x^{(k)}\|^2 \\ &= \|f(x^{(k)})\|^2 + 2f(x^{(k)})^T A\Delta x^{(k)} + \|A\Delta x^{(k)}\|^2 \\ &= \|f(x^{(k)})\|^2 - \|A\Delta x^{(k)}\|^2 - 2\lambda^{(k)}\|\Delta x^{(k)}\|^2\end{aligned}$$

- for large  $\lambda$ , we can use affine approximation to estimate actual cost:

$$\begin{aligned}g(x^{(k+1)}) &\approx g(x^{(k)}) + \nabla g(x^{(k)})^T \Delta x^{(k)} \\ \|f(x^{(k+1)})\|^2 &\approx \|f(x^{(k)})\|^2 + \nabla g(x^{(k)})^T \Delta x^{(k)} \\ &\approx \|f(x^{(k)})\|^2 - \frac{1}{2\lambda} \|\nabla g(x^{(k)})\|^2\end{aligned}$$

(using the expression on page 12-18)

# Regularization parameter

several strategies for adapting  $\lambda$  are possible; the simplest example:

- at iteration  $k$ , compute the solution  $\hat{x}$  of

$$\text{minimize } \|f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})\|^2 + \lambda^{(k)} \|x - x^{(k)}\|^2$$

- if  $\|f(\hat{x})\|^2 < \|f(x^{(k)})\|^2$ , take  $x^{(k+1)} = \hat{x}$  and decrease  $\lambda$
- otherwise, do not update  $x$  (take  $x^{(k+1)} = x^{(k)}$ ), but increase  $\lambda$

## Some variations

- compare actual cost reduction with predicted cost reduction
- solve a least squares problem with ‘trust region’

$$\begin{aligned} \text{minimize } & \|f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})\|^2 \\ \text{subject to } & \|x - x^{(k)}\|^2 \leq \gamma \end{aligned}$$

## Summary: Levenberg-Marquardt method

choose  $x^{(1)}$  and  $\lambda^{(1)}$  and repeat for  $k = 1, 2, \dots$ :

1. evaluate  $f(x^{(k)})$  and  $A = Df(x^{(k)})$
2. compute solution of regularized least squares problem:

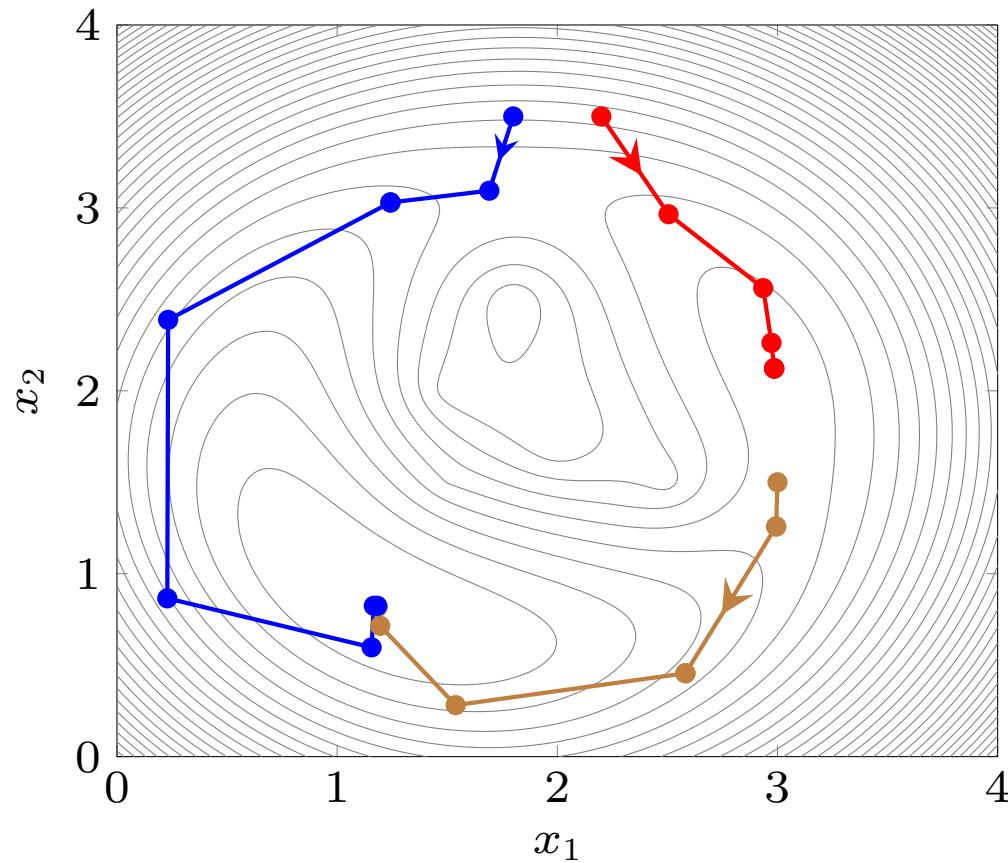
$$\hat{x} = x^{(k)} - (A^T A + \lambda^{(k)} I)^{-1} A^T f(x^{(k)})$$

3. define  $x^{(k+1)}$  and  $\lambda^{(k+1)}$  as follows:

$$\begin{cases} x^{(k+1)} = \hat{x} \text{ and } \lambda^{(k+1)} = \beta_1 \lambda^{(k)} & \text{if } \|f(\hat{x})\|^2 < \|f(x^{(k)})\|^2 \\ x^{(k+1)} = x^{(k)} \text{ and } \lambda^{(k+1)} = \beta_2 \lambda^{(k)} & \text{otherwise} \end{cases}$$

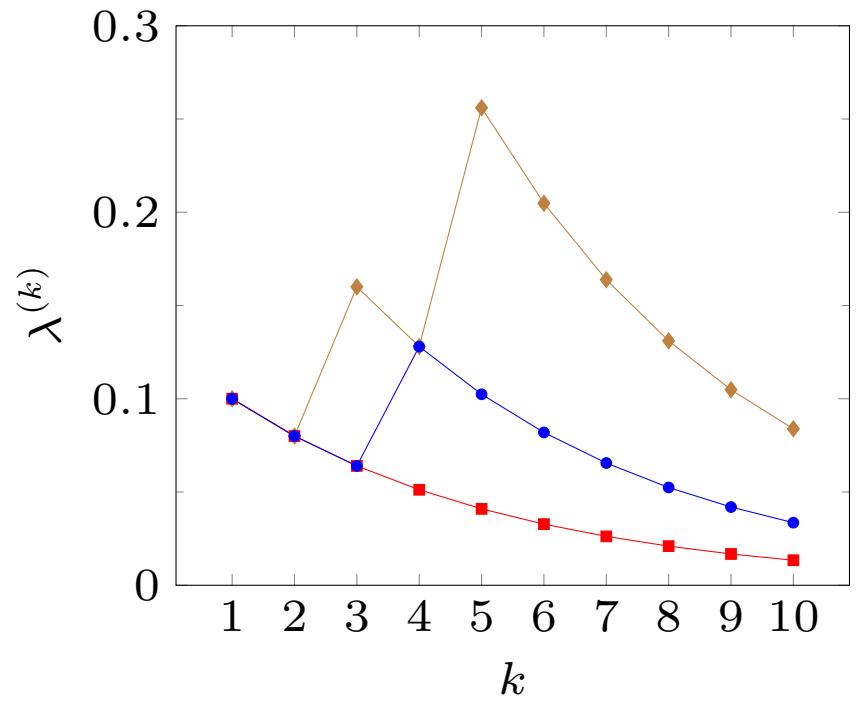
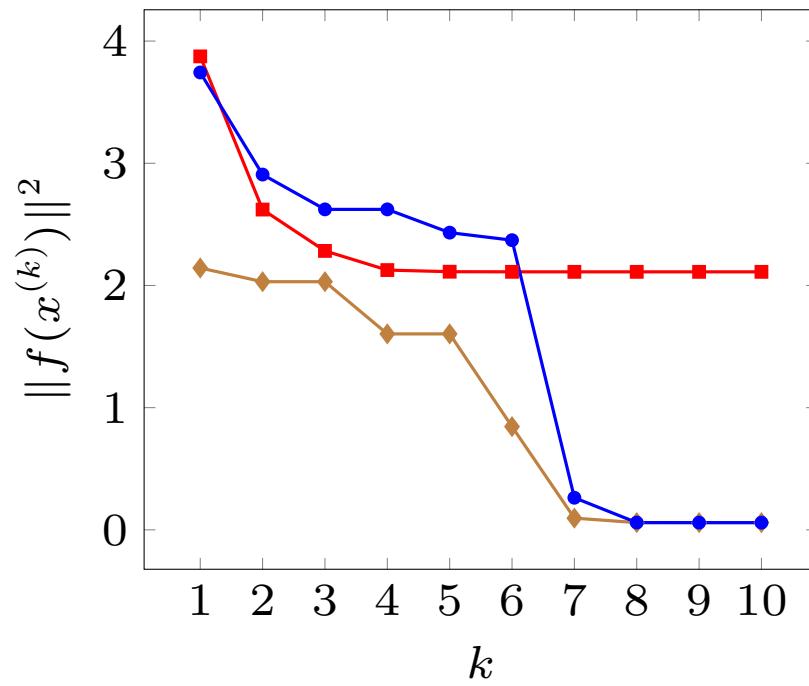
- $\beta_1, \beta_2$  are constants with  $0 < \beta_1 < 1 < \beta_2$
- in step 2,  $\hat{x}$  can be computed using a QR factorization
- terminate if  $\nabla g(x^{(k)}) = 2A^T f(x^{(k)})$  is sufficiently small

## Location from range measurements



- iterates from three starting points, with  $\lambda^{(1)} = 0.1$ ,  $\beta_1 = 0.8$ ,  $\beta_2 = 2$
- algorithm started at  $(1.8, 3.5)$  and  $(3.0, 1.5)$  finds minimum  $(1.18, 0.82)$
- started at  $(2.2, 3.5)$  converges to non-optimal point

# Cost function and regularization parameter



cost function and  $\lambda$  for the three starting points on previous page

# 13. Nonlinear equations

- Newton method for nonlinear equations
- damped Newton method for unconstrained minimization
- Newton method for nonlinear least squares

## Set of nonlinear equations

$n$  nonlinear equations in  $n$  variables  $x_1, x_2, \dots, x_n$ :

$$f_1(x_1, \dots, x_n) = 0$$

$$f_2(x_1, \dots, x_n) = 0$$

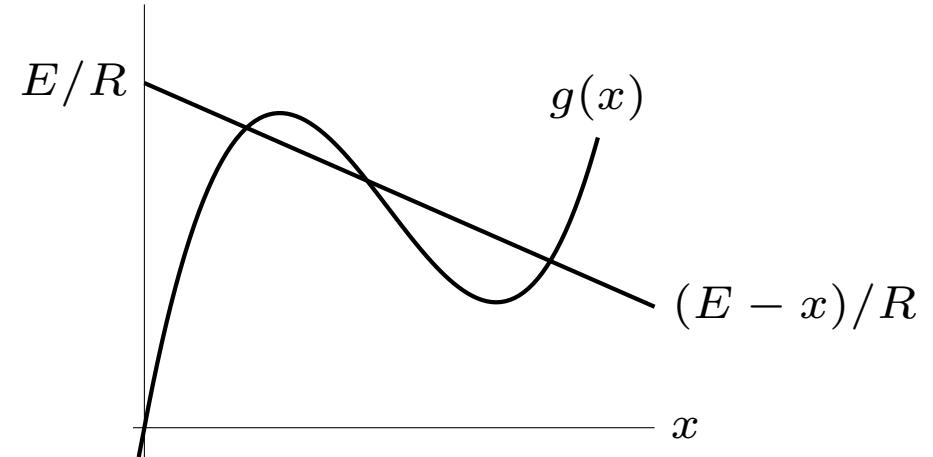
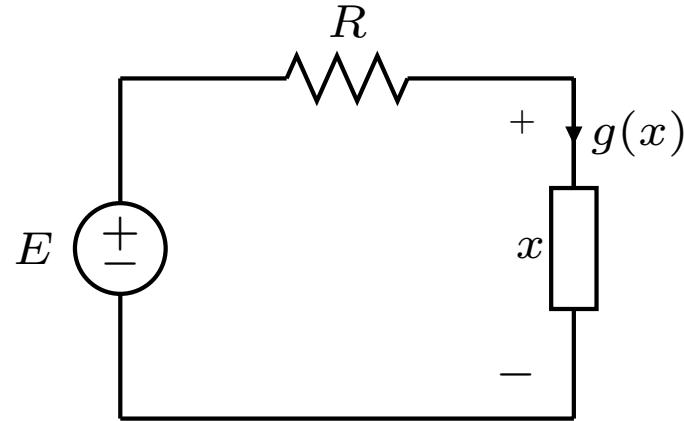
⋮

$$f_n(x_1, \dots, x_n) = 0$$

in vector notation:  $f(x) = 0$  with

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad f(x) = \begin{bmatrix} f_1(x_1, \dots, x_n) \\ f_2(x_1, \dots, x_n) \\ \vdots \\ f_n(x_1, \dots, x_n) \end{bmatrix}$$

## Example: nonlinear resistive circuit



$$g(x) - \frac{E - x}{R} = 0$$

a nonlinear equation in the variable  $x$ , with three solutions

# Newton method

suppose  $f : \mathbf{R}^n \rightarrow \mathbf{R}^n$  is differentiable

**Algorithm:** choose  $x^{(1)}$  and repeat for  $k = 1, 2, \dots$

$$x^{(k+1)} = x^{(k)} - Df(x^{(k)})^{-1} f(x^{(k)})$$

- each iteration requires one evaluation of  $f(x)$  and  $Df(x)$
- each iteration requires factorization of  $n \times n$  matrix  $Df(x)$
- we assume  $Df(x)$  is nonsingular

# Interpretation

$$x^+ = x - Df(x)^{-1}f(x) \quad (x = x^{(k)}, x^+ = x^{(k+1)})$$

- linearize  $f$  (*i.e.*, make affine approximation) around current iterate  $x$

$$\hat{f}(y) = f(x) + Df(x)(y - x)$$

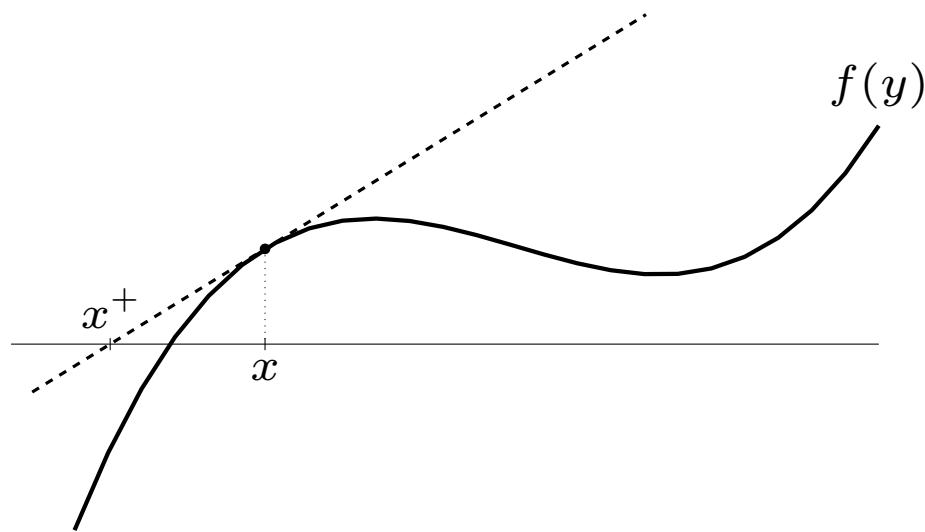
- solve linearized equation  $\hat{f}(y) = 0$

$$y = x - Df(x)^{-1}f(x)$$

- take  $y$  as new iterate  $x^+$

# One variable

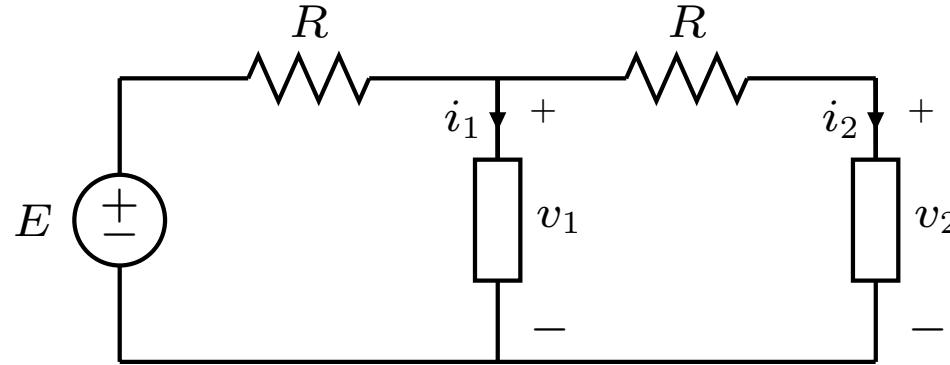
$$\hat{f}(y) = f(x) + f'(x)(y - x)$$



- affine approximation of  $f$  around  $x$  is  $\hat{f}(y) = f(x) + f'(x)(y - x)$
- solve the linearized equation  $\hat{f}(y) = 0$  and take the solution  $y$  as  $x^+$ :

$$x^+ = x - \frac{f(x)}{f'(x)}$$

## Example: nonlinear resistive circuit



- nonlinear resistors with  $i$ - $v$  characteristics  $i_1 = g_1(v_1)$ ,  $i_2 = g_2(v_2)$
- two nonlinear equations in two variables  $v_1$ ,  $v_2$

$$f_1(v_1, v_2) = g_1(v_1) + \frac{v_1 - E}{R} + \frac{v_1 - v_2}{R} = 0$$
$$f_2(v_1, v_2) = g_2(v_2) + \frac{v_2 - v_1}{R} = 0$$

- derivative matrix

$$Df(v) = \begin{bmatrix} g'_1(v_1) + 2/R & -1/R \\ -1/R & g'_2(v_2) + 1/R \end{bmatrix}$$

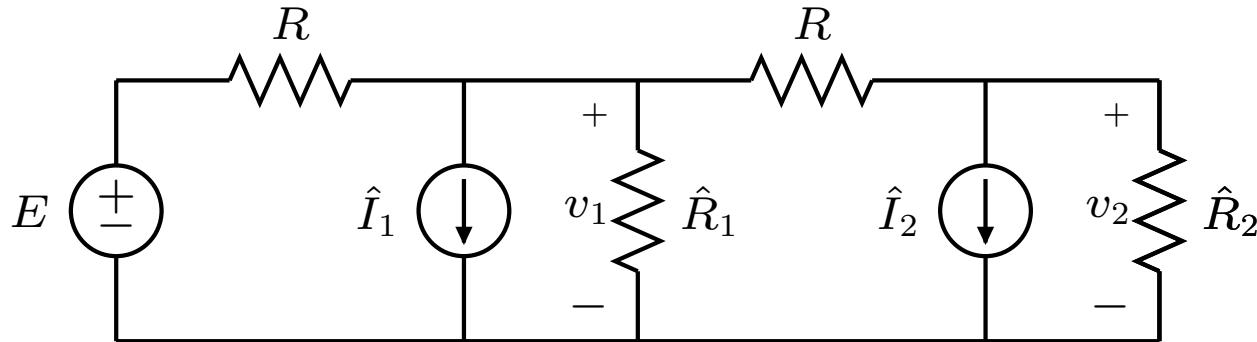
# Linearized equations

- linearized equations around  $\hat{v}_1, \hat{v}_2$ :

$$g_1(\hat{v}_1) - g'_1(\hat{v}_1)\hat{v}_1 + g'_1(\hat{v}_1)v_1 + \frac{v_1 - E}{R} + \frac{v_1 - v_2}{R} = 0$$

$$g_2(\hat{v}_2) - g'_2(\hat{v}_2)\hat{v}_2 + g'_2(\hat{v}_2)v_2 + \frac{v_2 - v_1}{R} = 0$$

- describes a linear resistive circuit



$$\hat{I}_1 = g_1(\hat{v}_1) - g'_1(\hat{v}_1)\hat{v}_1, \quad \hat{R}_1 = 1/g'_1(\hat{v}_1)$$

$$\hat{I}_2 = g_2(\hat{v}_2) - g'_2(\hat{v}_2)\hat{v}_2, \quad \hat{R}_2 = 1/g'_2(\hat{v}_2)$$

## Relation to Gauss-Newton method

recall Gauss-Newton method for nonlinear least squares problem

$$\text{minimize } \|f(x)\|^2$$

where  $f$  is a differentiable function from  $\mathbf{R}^n$  to  $\mathbf{R}^m$

- Gauss-Newton update

$$x^+ = x - (Df(x)^T Df(x))^{-1} Df(x)^T f(x)$$

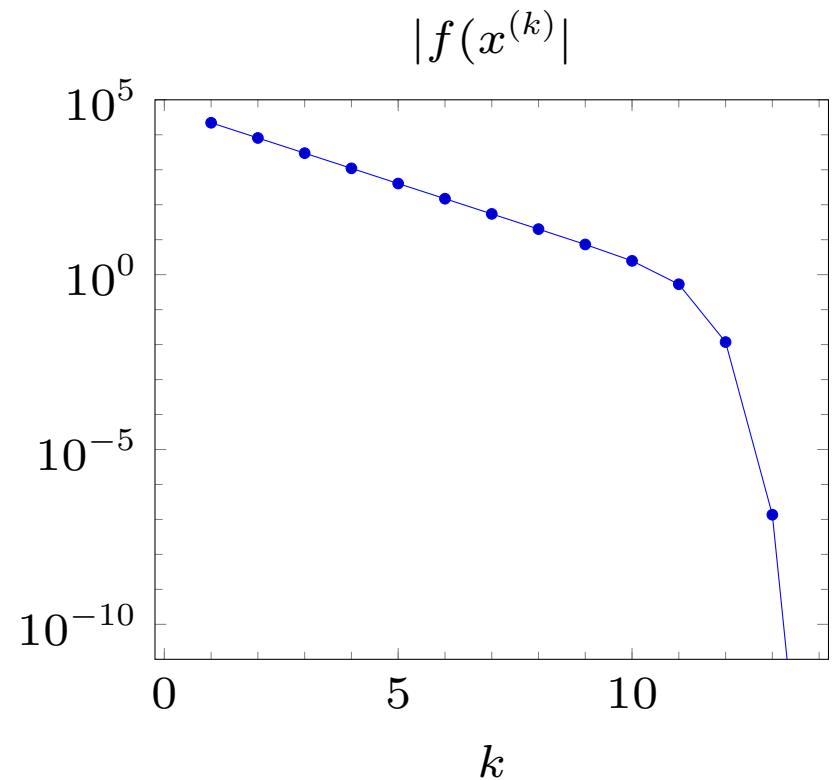
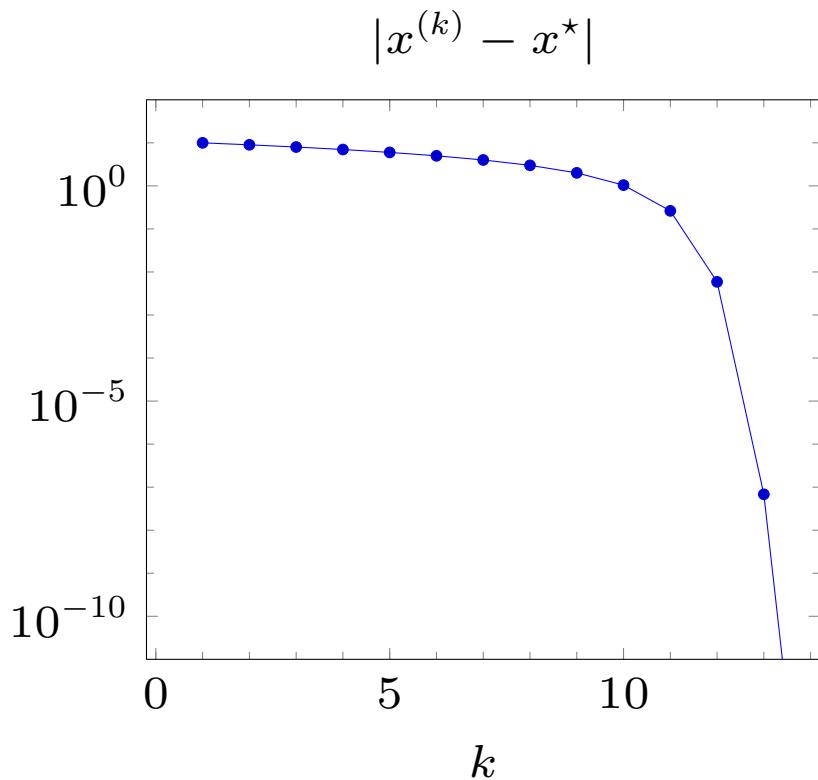
- if  $m = n$ , then  $Df(x)$  is square and this is the Newton update

$$x^+ = x - Df(x)^{-1} f(x)$$

# Example 1

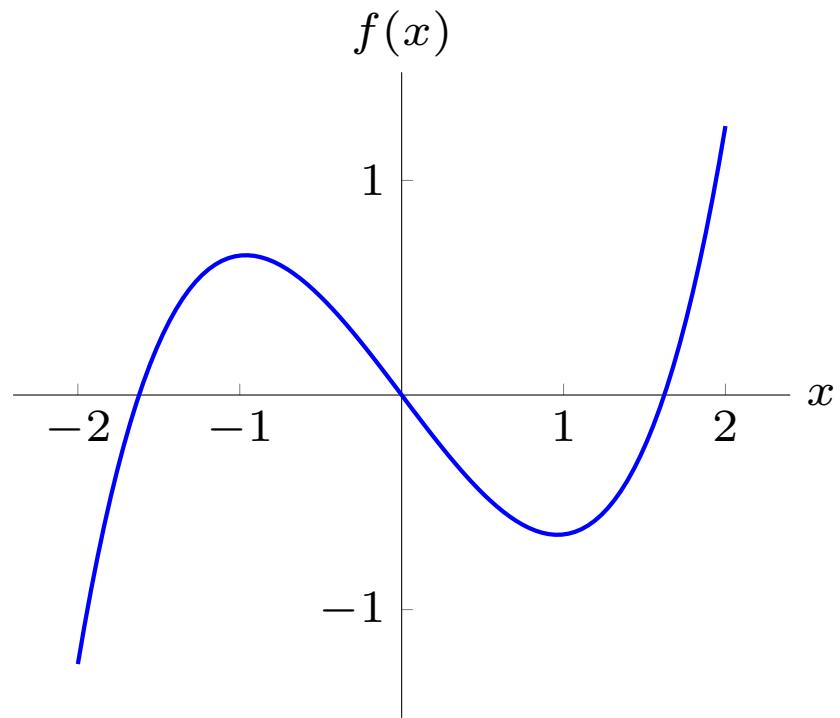
Newton method applied to

$$f(x) = e^x - e^{-x}, \quad x^{(1)} = 10$$



## Example 2

$$f(x) = e^x - e^{-x} - 3x$$

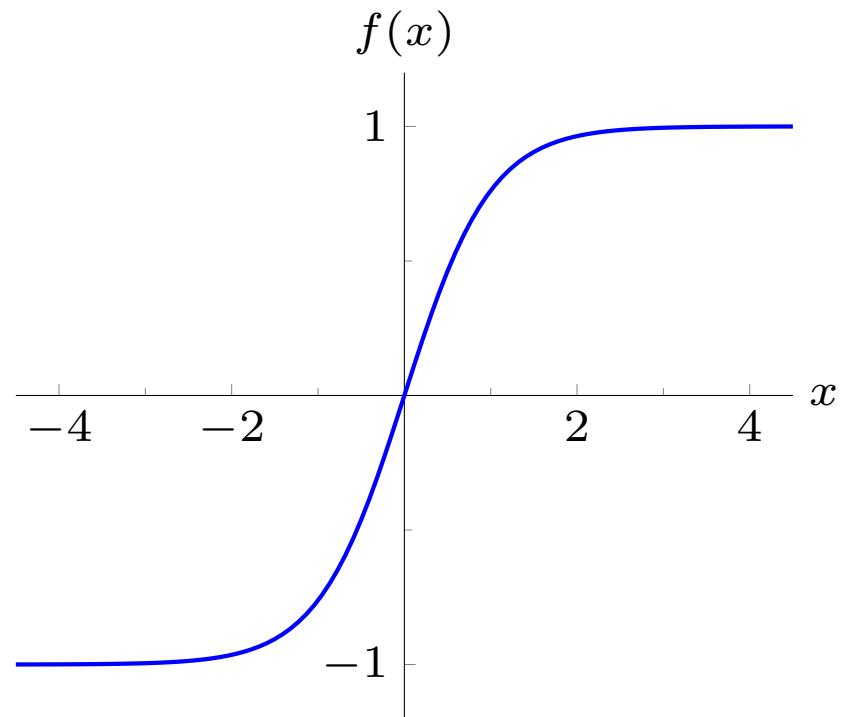


- starting point  $x^{(1)} = -1$ : converges to  $x^* = -1.62$
- starting point  $x^{(1)} = -0.8$ : converges to  $x^* = 1.62$
- starting point  $x^{(1)} = -0.7$ : converges to  $x^* = 0$

converges to a different solution depending on the starting point

## Example 3

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

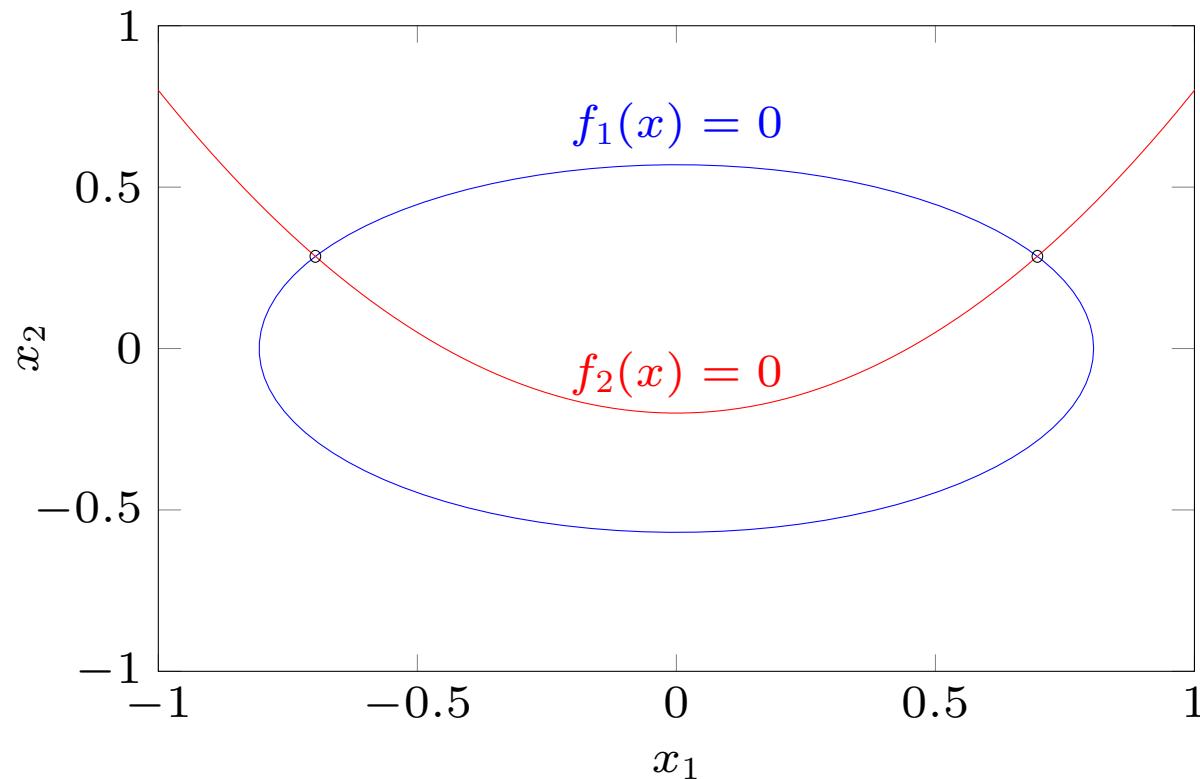


- starting point  $x^{(1)} = 0.9$ : converges very rapidly to  $x^* = 0$
- starting point  $x^{(1)} = 1.1$ : does not converge

## Example 4

$$f_1(x_1, x_2) = \log(x_1^2 + 2x_2^2 + 1) - 0.5 = 0$$

$$f_2(x_1, x_2) = x_2 - x_1^2 + 0.2 = 0$$



two equations in two variables; two solutions  $(0.70, 0.29), (-0.70, 0.29)$

## Example 4

### Newton iteration

- evaluate  $g = f(x)$  and

$$H = Df(x) = \begin{bmatrix} 2x_1/(x_1^2 + 2x_2^2 + 1) & 4x_2/(x_1^2 + 2x_2^2 + 1) \\ -2x_1 & 1 \end{bmatrix}$$

- solve  $Hv = -g$  (two linear equations in two variables)
- update  $x := x + v$

### Results

- $x^{(1)} = (1, 1)$ : converges to  $x^* = (0.70, 0.29)$  in about 4 iterations
- $x^{(1)} = (-1, 1)$ : converges to  $x^* = (-0.70, 0.29)$  in about 4 iterations
- $x^{(1)} = (1, -1)$  or  $x^{(0)} = (-1, -1)$ : does not converge

# Observations

- Newton's method works very well if started near a solution
- may not work at all otherwise
- can converge to different solutions depending on the starting point
- does not necessarily find the solution closest to the starting point

## Convergence of Newton's method

if  $f(x^*) = 0$  and  $Df(x^*)$  is nonsingular, and  $x^{(1)}$  is close to  $x^*$ , then

$$x^{(k)} \rightarrow x^*, \quad \|x^{(k+1)} - x^*\| \leq c \|x^{(k)} - x^*\|^2$$

for some  $c > 0$

- this is called quadratic convergence
- explains fast convergence when started near solution
- in practice, we don't know what  $c$  is, or how close  $x^{(1)}$  has to be

# Outline

- Newton's method for sets of nonlinear equations
- **damped Newton for unconstrained minimization**
- Newton method for nonlinear least squares

# Unconstrained minimization problem

$$\text{minimize } g(x_1, x_2, \dots, x_n)$$

$g$  is a function from  $\mathbf{R}^n$  to  $\mathbf{R}$

- $x = (x_1, x_2, \dots, x_n)$  is  $n$ -vector of optimization *variables*
- $g(x)$  is the *cost function* or *objective function*
- to solve a maximization problem (*i.e.*, maximize  $g(x)$ ), minimize  $-g(x)$
- we will assume that  $g$  is twice differentiable

# Local and global optimum

- $x^*$  is an *optimal point* (or a *minimum*) if

$$g(x^*) \leq g(x) \quad \text{for all } x$$

also called *globally* optimal

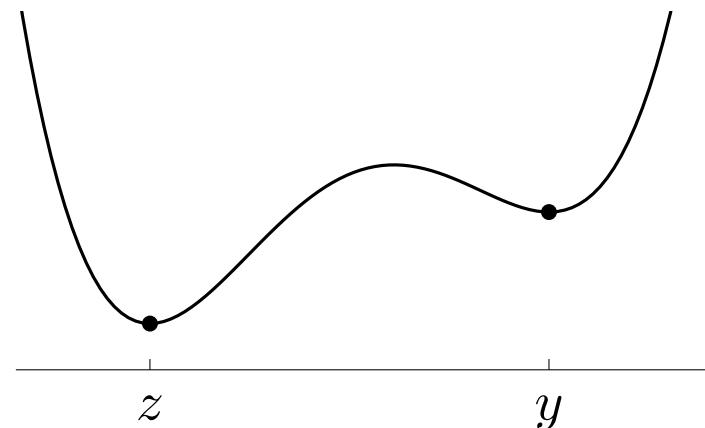
- $x^*$  is a *locally optimal point* (*local minimum*) if for some  $R > 0$

$$g(x^*) \leq g(x) \quad \text{for all } x \text{ with } \|x - x^*\| \leq R$$

## Example

$y$  is locally optimal

$z$  is (globally) optimal



## Gradient and Hessian

**Gradient** of  $g : \mathbf{R}^n \rightarrow \mathbf{R}$  at  $z \in \mathbf{R}^n$  is the  $n$ -vector

$$\nabla g(z) = \left( \frac{\partial g}{\partial x_1}(z), \frac{\partial g}{\partial x_2}(z), \dots, \frac{\partial g}{\partial x_n}(z) \right)$$

**Hessian** of  $g$  at  $z$ : a symmetric  $n \times n$  matrix  $\nabla^2 g(z)$  with elements

$$\nabla^2 g(z)_{ij} = \frac{\partial^2 g}{\partial x_i \partial x_j}(z)$$

this is also the derivative matrix  $Df(z)$  of  $f(x) = \nabla g(x)$  at  $z$

**Quadratic (second order) approximation** of  $g$  around  $z$ :

$$g_q(x) = g(z) + \nabla g(z)^T(x - z) + \frac{1}{2}(x - z)^T \nabla^2 g(z)(x - z)$$

## Examples

**Affine function:**  $g(x) = a^T x + b$

$$\nabla g(x) = a, \quad \nabla^2 g(x) = 0$$

**Quadratic function:**  $g(x) = x^T P x + q^T x + r$  with  $P$  symmetric

$$\nabla g(x) = 2Px + q, \quad \nabla^2 g(x) = 2P$$

**Least squares cost:**  $g(x) = \|Ax - b\|^2 = x^T A^T Ax - 2b^T Ax + b^T b$

$$\nabla g(x) = 2A^T Ax - 2A^T b, \quad \nabla^2 g(x) = 2A^T A$$

# Properties

**Linear combination:** if  $g(x) = \alpha_1 g_1(x) + \alpha_2 g_2(x)$ , then

$$\nabla g(x) = \alpha_1 \nabla g_1(x) + \alpha_2 \nabla g_2(x)$$

$$\nabla^2 g(x) = \alpha_1 \nabla^2 g_1(x) + \alpha_2 \nabla^2 g_2(x)$$

**Composition with affine mapping:** if  $g(x) = h(Cx + d)$ , then

$$\nabla g(x) = C^T \nabla h(Cx + d)$$

$$\nabla^2 g(x) = C^T \nabla^2 h(Cx + d) C$$

## Example

$$g(x_1, x_2) = e^{x_1+x_2-1} + e^{x_1-x_2-1} + e^{-x_1-1}$$

## Gradient

$$\nabla g(x) = \begin{bmatrix} e^{x_1+x_2-1} + e^{x_1-x_2-1} - e^{-x_1-1} \\ e^{x_1+x_2-1} - e^{x_1-x_2-1} \end{bmatrix}$$

## Hessian

$$\nabla^2 g(x) = \begin{bmatrix} e^{x_1+x_2-1} + e^{x_1-x_2-1} + e^{-x_1-1} & e^{x_1+x_2-1} - e^{x_1-x_2-1} \\ e^{x_1+x_2-1} - e^{x_1-x_2-1} & e^{x_1+x_2-1} + e^{x_1-x_2-1} \end{bmatrix}$$

## Gradient and Hessian via composition property

express  $g$  as  $g(x) = h(Cx + d)$  with  $h(y_1, y_2, y_3) = e^{y_1} + e^{y_2} + e^{y_3}$  and

$$C = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 0 \end{bmatrix}, \quad d = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

**Gradient:**  $\nabla g(x) = C^T \nabla h(Cx + d)$

$$\nabla g(x) = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} e^{x_1+x_2-1} \\ e^{x_1-x_2-1} \\ e^{-x_1-1} \end{bmatrix}$$

**Hessian:**  $\nabla^2 g(x) = C^T \nabla h^2(Cx + d)C$

$$\nabla^2 g(x) = \begin{bmatrix} 1 & 1 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} e^{x_1+x_2-1} & 0 & 0 \\ 0 & e^{x_1-x_2-1} & 0 \\ 0 & 0 & e^{-x_1-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & 0 \end{bmatrix}$$

# Optimality conditions for twice differentiable $g$

**Necessary condition:** if  $x^*$  is locally optimal, then

$$\nabla g(x^*) = 0 \quad \text{and} \quad \nabla^2 g(x^*) \text{ is positive semidefinite}$$

**Sufficient condition:** if  $x^*$  satisfies

$$\nabla g(x^*) = 0 \quad \text{and} \quad \nabla^2 g(x^*) \text{ is positive definite}$$

then  $x^*$  is locally optimal

## Necessary and sufficient condition for convex functions

- $g$  is called *convex* if  $\nabla^2 g(x)$  is positive semidefinite everywhere
- if  $g$  is convex then  $x^*$  is optimal if and only if  $\nabla g(x^*) = 0$

## Examples ( $n = 1$ )

- $g(x) = \log(e^x + e^{-x})$

$$g'(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad g''(x) = \frac{4}{(e^x + e^{-x})^2}$$

$g''(x) \geq 0$  everywhere;  $x^* = 0$  is the unique optimal point

- $g(x) = x^4$

$$g'(x) = 4x^3, \quad g''(x) = 12x^2$$

$g''(x) \geq 0$  everywhere;  $x^* = 0$  is the unique optimal point

- $g(x) = x^3$

$$g'(x) = 3x^2, \quad g''(x) = 6x$$

$g'(0) = 0, g''(0) = 0$  but  $x = 0$  is not locally optimal

## Examples

- $g(x) = x^T Px + q^T x + r$  ( $P$  is positive definite)

$$\nabla g(x) = 2Px + q, \quad \nabla^2 g(x) = 2P$$

$\nabla^2 g(x)$  is positive definite everywhere, hence the unique optimal point is

$$x^\star = -(1/2)P^{-1}q$$

- $g(x) = \|Ax - b\|^2$  ( $A$  is a matrix with linearly independent columns)

$$\nabla g(x) = 2A^T Ax - 2A^T b, \quad \nabla^2 g(x) = 2A^T A$$

$\nabla^2 g(x)$  is positive definite everywhere, hence the unique optimal point is

$$x^\star = (A^T A)^{-1} A^T b$$

## Examples

example of page 13-22: we can express  $\nabla^2 g(x)$  as

$$\nabla^2 g(x) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} e^{x_1+x_2-1} & 0 & 0 \\ 0 & e^{x_1-x_2-1} & 0 \\ 0 & 0 & e^{-x_1-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{bmatrix}$$

this shows that  $\nabla^2 g(x)$  is positive definite for all  $x$

therefore  $x^*$  is optimal if and only if

$$\nabla g(x^*) = \begin{bmatrix} e^{x_1^*+x_2^*-1} + e^{x_1^*-x_2^*-1} - e^{-x_1^*-1} \\ e^{x_1^*+x_2^*-1} - e^{x_1^*-x_2^*-1} \end{bmatrix} = 0$$

two nonlinear equations in two variables

## Newton's method for minimizing a convex function

if  $\nabla^2 g(x)$  is positive definite everywhere, we can minimize  $g(x)$  by solving

$$\nabla g(x) = 0$$

**Algorithm:** choose  $x^{(1)}$  and repeat for  $k = 1, 1, 2, \dots$

$$x^{(k+1)} = x^{(k)} - \nabla^2 g(x^{(k)})^{-1} \nabla g(x^{(k)})$$

- $v = -\nabla^2 g(x)^{-1} \nabla g(x)$  is called the *Newton step* at  $x$
- converges if started sufficiently close to the solution
- Newton step computed by a Cholesky factorization of the Hessian

# Interpretations of Newton step

## Affine approximation of gradient

- affine approximation of  $f(y) = \nabla g(y)$  around  $x$  is

$$\hat{f}(y) = \nabla g(x) + \nabla^2 g(x)(y - x)$$

- Newton update  $x + v$  is solution of linear equation  $\hat{f}(y) = 0$

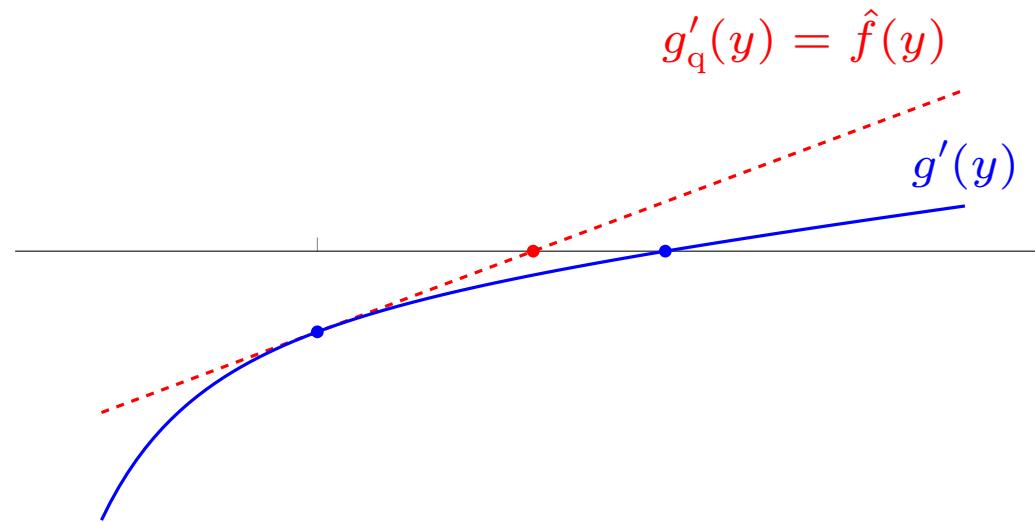
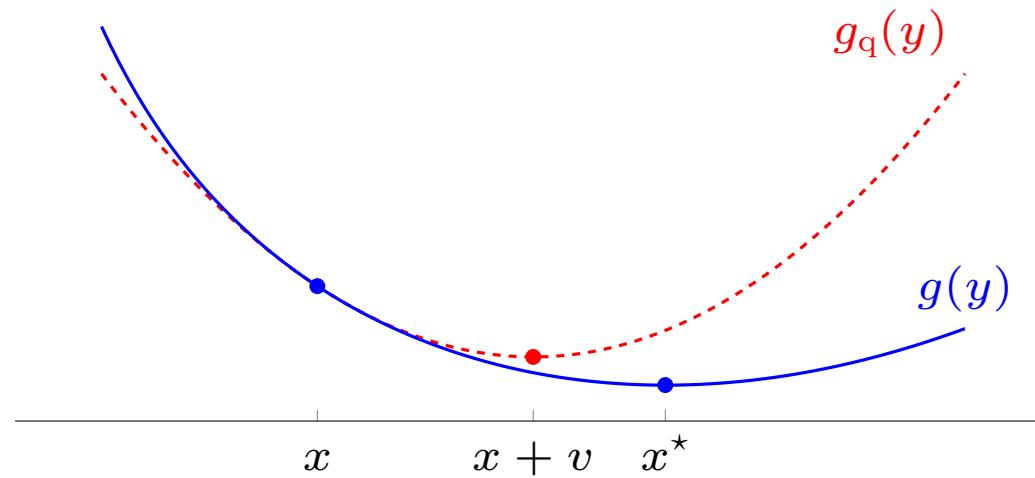
## Quadratic approximation of function

- quadratic approximation of  $g(y)$  around  $x$  is

$$g_q(y) = g(x) + \nabla g(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 g(x)(y - x)$$

- Newton update  $x + v$  is minimizer of  $g_q$  (solution of  $\nabla g_q(y) = 0$ )

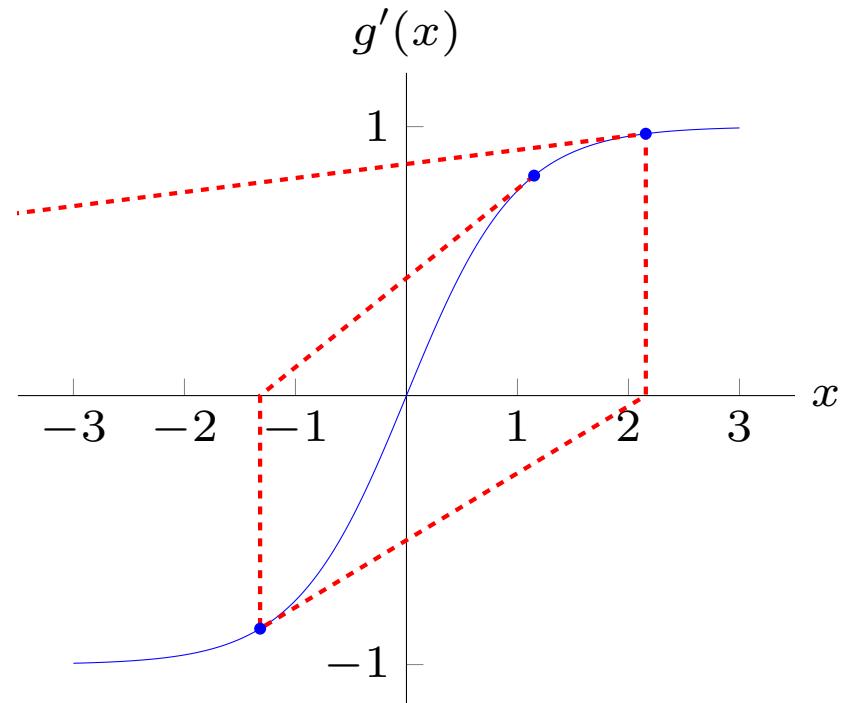
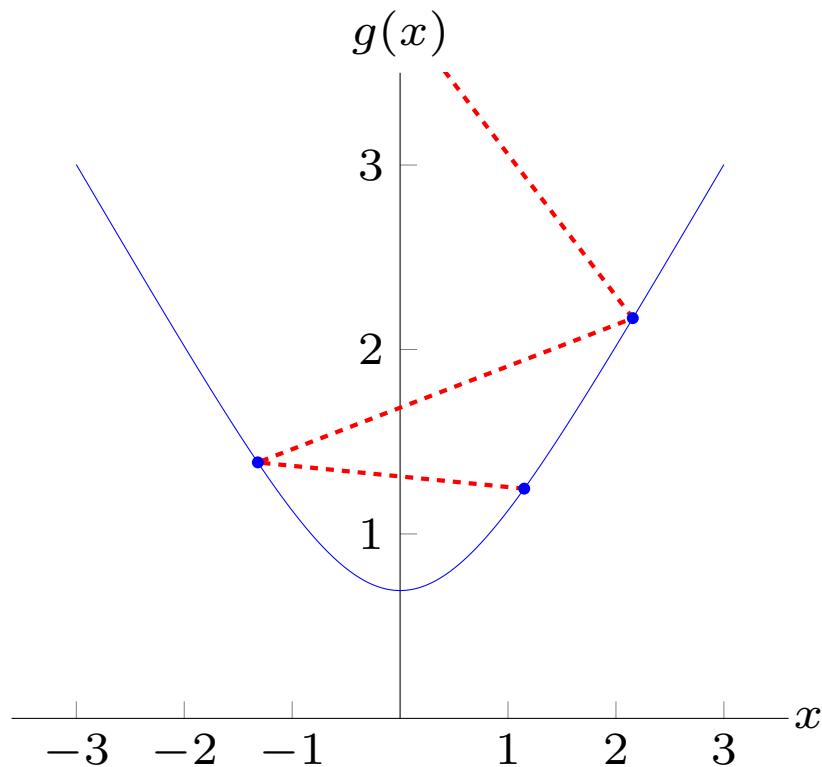
## Example ( $n = 1$ )



$$g_q(y) = g(x) + g'(x)(y - x) + \frac{g''(x)}{2}(y - x)^2$$

# Example

$$g(x) = \log(e^x + e^{-x}), \quad g'(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad g''(x) = \frac{4}{(e^x + e^{-x})^2}$$



does not converge when started at  $x^{(1)} = 1.15$

## Damped Newton method

use update  $x^+ = x + tv$  and choose  $t$  so that  $g(x + tv) < g(x)$

**Algorithm:** choose  $x^{(1)}$  and repeat for  $k = 1, 2, \dots$

1. compute Newton step  $v = -\nabla^2 g(x^{(k)})^{-1} \nabla g(x^{(k)})$
2. find largest  $t$  in  $\{1, 0.5, 0.5^2, 0.5^3, \dots\}$  that satisfies

$$g(x^{(k)} + tv) \leq g(x^{(k)}) + \alpha t \nabla g(x^{(k)})^T v$$

and take  $x^{(k+1)} = x^{(k)} + tv$

- $\alpha$  is an algorithm parameter (small and positive, e.g.,  $\alpha = 0.01$ )
- $t$  is called the *step size*; step 2 in algorithm is called *line search*

## Interpretation of line search

to determine a suitable step size, consider the function  $h : \mathbf{R} \rightarrow \mathbf{R}$

$$h(t) = g(x + tv)$$

$x = x^{(k)}$  is the current iterate;  $v$  is the Newton step at  $x$

- $h'(0) = \nabla g(x)^T v$  is the *directional derivative* of  $g$  at  $x$  in direction  $v$
- affine approximation of  $h$  at  $t = 0$  is

$$\hat{h}(t) = h(0) + h'(0)t = g(x) + t\nabla g(x)^T v$$

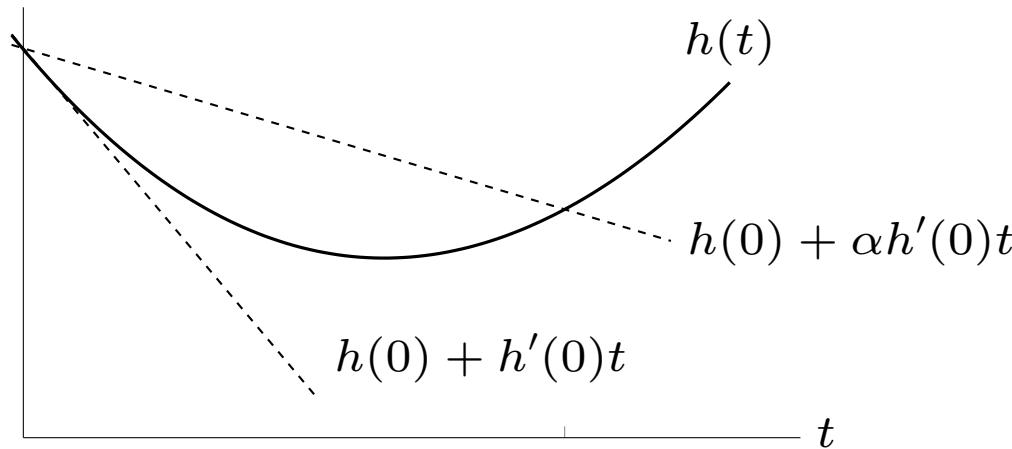
- line search accepts step size  $t$  if  $g(x + tv) \leq g(x) + \alpha t \nabla g(x)^T v$ ; i.e.,

$$h(t) - h(0) \leq \alpha(\hat{h}(t) - h(0))$$

decrease  $h(t) - h(0)$  is at least  $\alpha$  times what is expected based on  $\hat{h}$

## Interpretation of line search

start with  $t = 1$ ; divide  $t$  by 2 until  $h(t) \leq h(0) + \alpha h'(0)t$

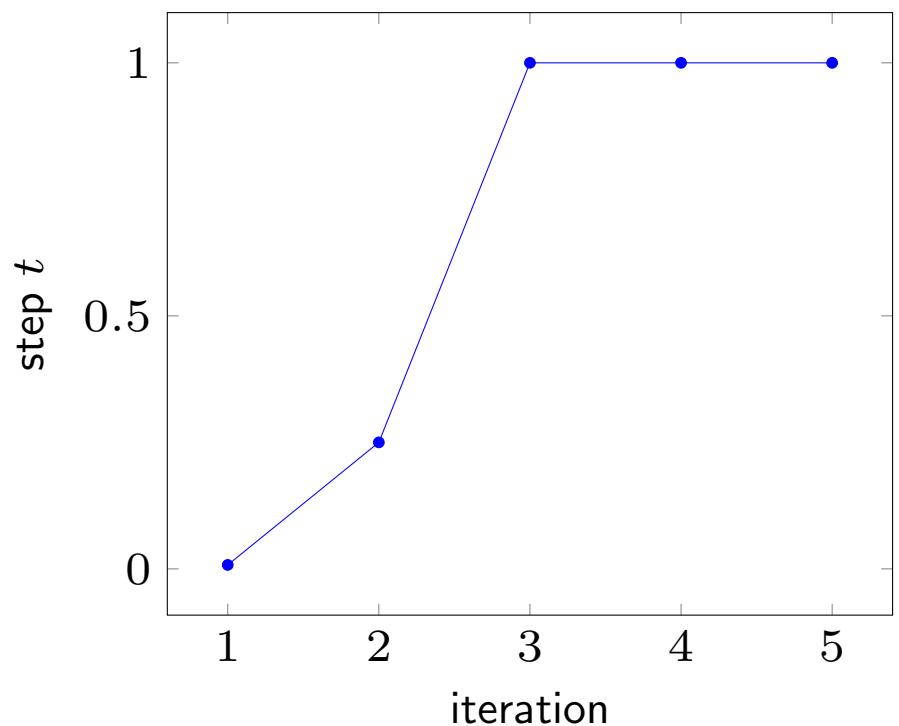
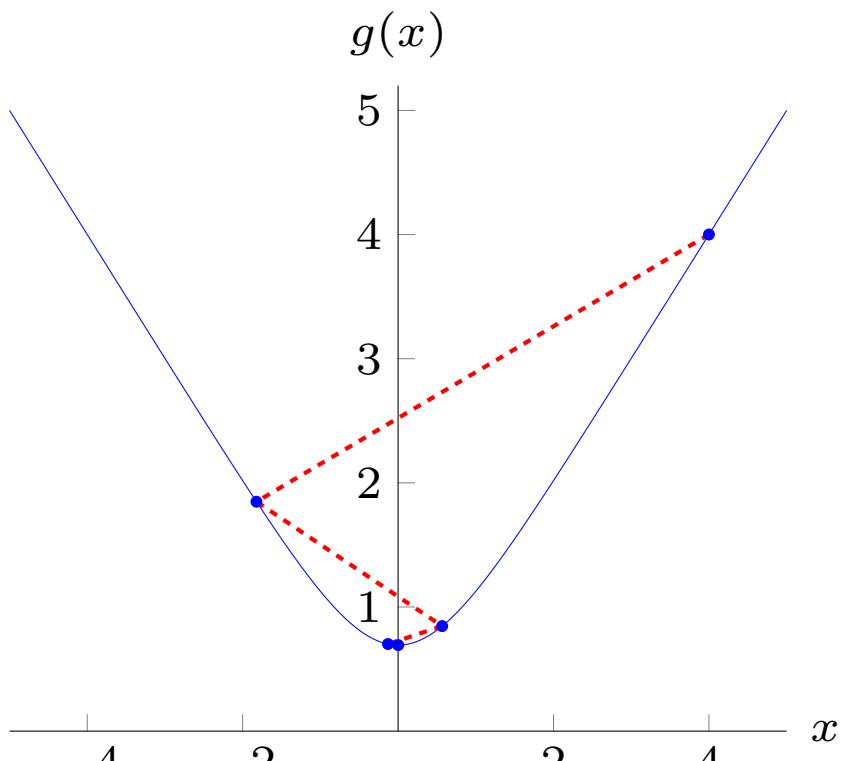


- works if  $h'(0) = \nabla g(x)^T v < 0$  ( $v$  is a **descent direction**)
- if  $\nabla^2 g(x)$  is positive definite, the Newton step is a descent direction

$$h'(0) = \nabla g(x)^T v = v^T \nabla^2 g(x) v < 0$$

# Example

$$g(x) = \log(e^x + e^{-x}), \quad x^{(0)} = 4$$



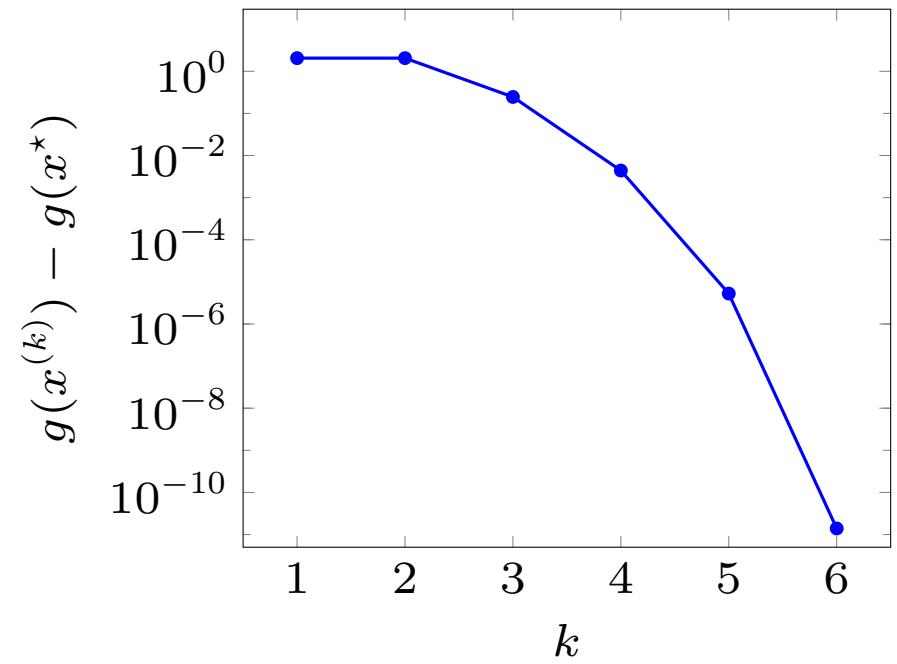
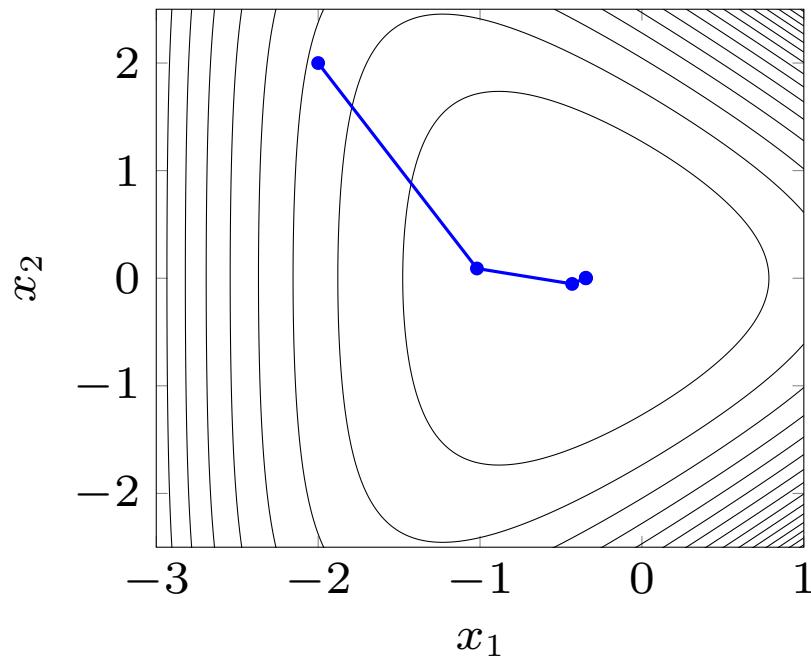
close to the solution: very fast convergence, no backtracking steps

# Example

example of page 13-22

$$g(x_1, x_2) = e^{x_1+x_2-1} + e^{x_1-x_2-1} + e^{-x_1-1}$$

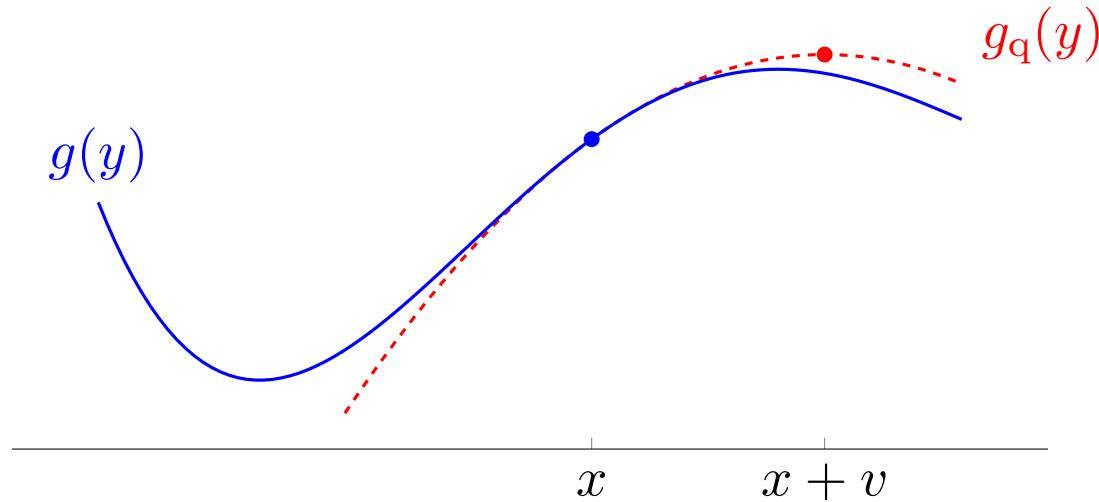
damped Newton method started at  $x = (-2, 2)$



## Newton method for nonconvex functions

if  $\nabla^2 g(x)$  is not positive definite, it is possible that Newton step  $v$  satisfies

$$\nabla g(x)^T v = -\nabla g(x)^T \nabla^2 g(x)^{-1} \nabla g(x) > 0$$



- if Newton step is not descent direction, replace it with descent direction
- simplest choice is  $v = -\nabla g(x)$ ; practical methods make other choices

# Outline

- Newton's method for sets of nonlinear equations
- damped Newton for unconstrained minimization
- **Newton method for nonlinear least squares**

## Hessian of nonlinear least squares cost

$$g(x) = \|f(x)\|^2 = \sum_{i=1}^m f_i(x)^2$$

- gradient (from page 12-13):

$$\nabla g(x) = 2 \sum_{i=1}^m f_i(x) \nabla f_i(x) = 2Df(x)^T f(x)$$

- second derivatives:

$$\frac{\partial^2 g}{\partial x_j \partial x_k}(x) = 2 \sum_{i=1}^m \left( \frac{\partial f_i}{\partial x_j}(x) \frac{\partial f_i}{\partial x_k}(x) + f_i(x) \frac{\partial^2 f_i}{\partial x_j \partial x_k}(x) \right)$$

- Hessian

$$\nabla^2 g(x) = 2Df(x)^T Df(x) + 2 \sum_{i=1}^m f_i(x) \nabla^2 f_i(x))$$

# Newton and Gauss-Newton steps

**(Undamped) Newton step:**

$$\begin{aligned}v_{\text{nt}} &= -\nabla^2 g(x)^{-1} \nabla g(x) \\&= \left( Df(x)^T Df(x) + \sum_{i=1}^m f_i(x) \nabla^2 f_i(x) \right)^{-1} Df(x)^T f(x)\end{aligned}$$

**Gauss-Newton step** (from pages 12-15):

$$v_{\text{gn}} = -\left(Df(x)^T Df(x)\right)^{-1} Df(x)^T f(x)$$

- can be written as  $v_{\text{gn}} = -H_{\text{gn}}^{-1} \nabla g(x)$  where  $H_{\text{gn}} = 2Df(x)^T Df(x)$
- $H_{\text{gn}}$  is the Hessian without the term  $\sum_i f_i(x) \nabla^2 f_i(x)$

# Comparison

## Newton step

- requires second derivatives of  $f$
- not always a descent direction ( $\nabla^2 g(x)$  not necessarily positive definite)
- fast convergence near local minimum

## Gauss-Newton step

- does not require second derivatives
- a descent direction (if columns of  $Df(x)$  are linearly independent):

$$\nabla g(x)^T v_{\text{gn}} = -2v_{\text{gn}}^T Df(x)^T Df(x) v_{\text{gn}} < 0 \quad \text{if } v_{\text{gn}} \neq 0$$

- local convergence to  $x^*$  is similar to Newton method if

$$\sum_{i=1}^m f_i(x^*) \nabla^2 f_i(x^*)$$

is small (e.g.,  $f(x^*)$  is small, or  $f$  is nearly affine around  $x^*$ )

# 14. Problem condition

- condition of a mathematical problem
- matrix norm
- condition number

# Sources of error in numerical computation

**Example:** evaluate a function  $f : \mathbf{R} \rightarrow \mathbf{R}$  at a given  $x$

sources of error in the result:

- $x$  is not exactly known
  - measurement errors
  - errors in previous computations

→ how sensitive is  $f(x)$  to errors in  $x$ ?
- the algorithm for computing  $f(x)$  is not exact
  - discretization (*e.g.*, algorithm uses a table to look up function values)
  - truncation (*e.g.*, function is evaluated by truncating a Taylor series)
  - rounding error during the computation

→ how large is the error introduced by the algorithm?

# **Condition (conditioning) of a problem**

describes sensitivity of the solution with respect to errors in the data

- **well-conditioned problem:**

small changes in the data produce small changes in the solution

- **ill-conditioned (badly conditioned) problem:**

small changes in the data can produce large changes in the solution

a rigorous definition depends on what ‘large error’ means

- absolute or relative error, which norm is used, . . .
- the informal definition is sufficient for our purposes

## Example: function evaluation

here the problem is: given  $x$ , evaluate  $y = f(x)$

- if  $x$  is changed to  $x + \Delta x$ , solution changes to

$$y + \Delta y = f(x + \Delta x)$$

- condition with respect to absolute error in  $x$  and  $y$

$$|\Delta y| \approx |f'(x)| |\Delta x|$$

problem is ill-conditioned w.r.t. absolute error if  $|f'(x)|$  is very large

- condition with respect to relative errors in  $x$  and  $y$

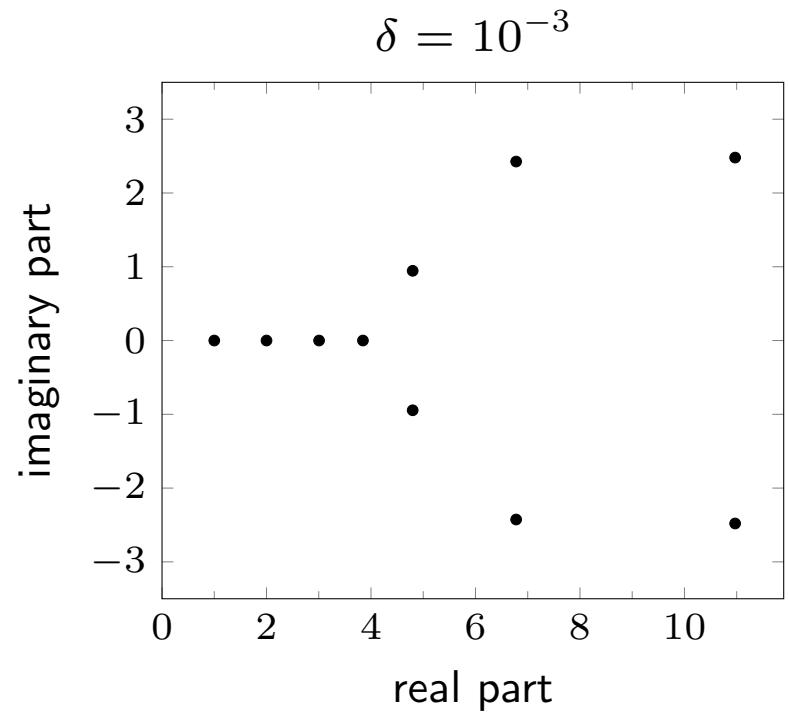
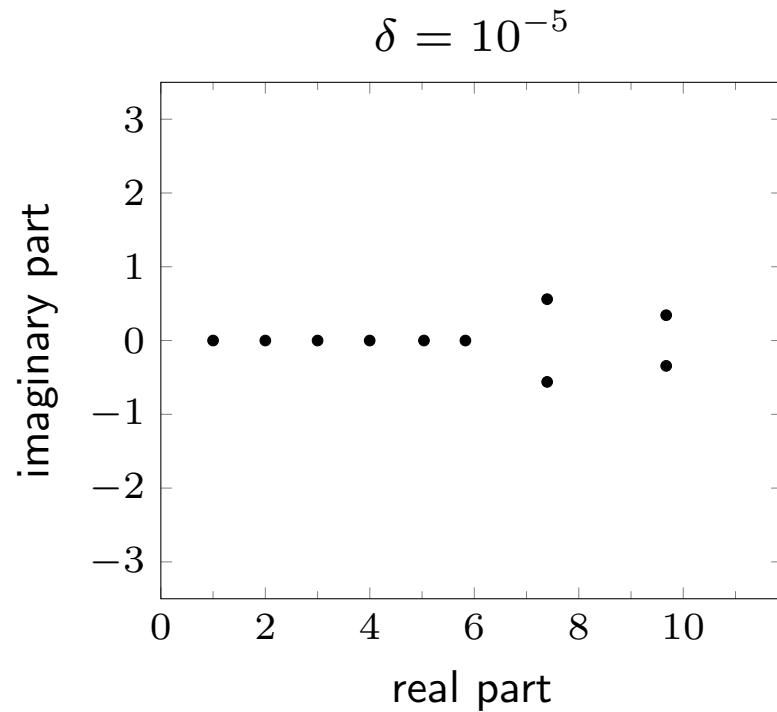
$$\frac{|\Delta y|}{|y|} \approx \frac{|f'(x)| |x|}{|f(x)|} \frac{|\Delta x|}{|x|}$$

ill-conditioned w.r.t relative error if  $|f'(x)| |x| / |f(x)|$  is very large

# Roots of a polynomial

$$p(x) = (x - 1)(x - 2) \cdots (x - 10) + \delta \cdot x^{10}$$

roots of  $p$  computed by MATLAB for two values of  $\delta$



roots are very sensitive to errors in the coefficients

# Condition of a set of linear equations

- assume  $A$  is nonsingular and  $Ax = b$
- if we change  $b$  to  $b + \Delta b$ , the new solution is  $x + \Delta x$  with

$$A(x + \Delta x) = b + \Delta b$$

- the change in  $x$  is

$$\Delta x = A^{-1} \Delta b$$

## Condition

- the equations are *well-conditioned* if small  $\Delta b$  results in small  $\Delta x$
- the equations are *ill-conditioned* if small  $\Delta b$  can result in large  $\Delta x$

## Example of ill-conditioned equations

$$A = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 + 10^{-10} & 1 - 10^{-10} \end{bmatrix}, \quad A^{-1} = \begin{bmatrix} 1 - 10^{10} & 10^{10} \\ 1 + 10^{10} & -10^{10} \end{bmatrix}$$

- solution for  $b = (1, 1)$  is  $x = (1, 1)$
- change in  $x$  if we change  $b$  to  $b + \Delta b$ :

$$\Delta x = A^{-1} \Delta b = \begin{bmatrix} \Delta b_1 - 10^{10}(\Delta b_1 - \Delta b_2) \\ \Delta b_1 + 10^{10}(\Delta b_1 - \Delta b_2) \end{bmatrix}$$

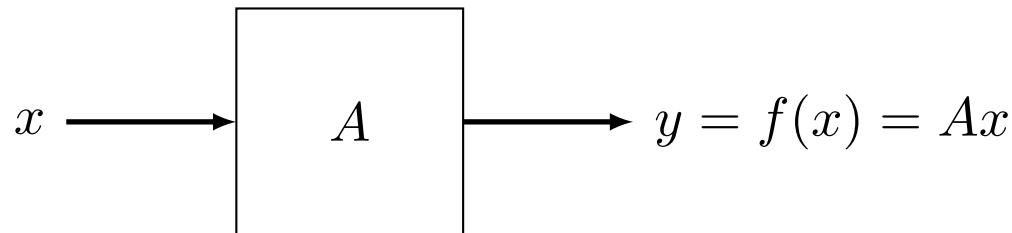
small  $\Delta b$  can lead to extremely large  $\Delta x$

# Outline

- condition of a mathematical problem
- **matrix norm**
- condition number

# Norm of a matrix

$m \times n$  matrix  $A$  defines a linear function  $f(x) = Ax$



- $\|Ax\|/\|x\|$  gives the *amplification factor* or *gain* in the direction  $x$
- we define the *norm* of  $A$  as the maximum gain over all directions:

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

this norm is also called the *spectral norm* or *2-norm*

- in MATLAB: `norm(A)`

# Computing the norm of a matrix

**Simple matrices:** sometimes it is easy to maximize  $\|Ax\|/\|x\|$

- zero matrix:  $\|0\| = 0$
- identity matrix:  $\|I\| = 1$
- diagonal matrix:

$$A = \begin{bmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{nn} \end{bmatrix}, \quad \|A\| = \max_{i=1,\dots,n} |A_{ii}|$$

- matrix with orthonormal columns:  $\|A\| = 1$

**General matrices:**  $\|A\|$  must be computed by numerical algorithms

## Other matrix norms

many other definitions of matrix norms exist, e.g., the *Frobenius* norm

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n A_{ij}^2}$$

- has a simple explicit expression
- in MATLAB: use `norm(A, 'fro')`

in this course:  $\|A\|$  stands for the norm on defined on page 14-8

- no simple explicit expression (except for special  $A$ )
- readily computed numerically

## Properties of the matrix norm

- *nonnegative definiteness*:  $\|A\| \geq 0$  for all  $A$ ;  $\|A\| = 0$  iff  $A = 0$
- *homogeneity*:  $\|\alpha A\| = |\alpha| \|A\|$
- *triangle inequality*:  $\|A + B\| \leq \|A\| + \|B\|$
- $\|Ax\| \leq \|A\| \|x\|$  if the product  $Ax$  exists
- $\|AB\| \leq \|A\| \|B\|$  if the product  $AB$  exists
- if  $A$  is nonsingular:  $\|A\| \|A^{-1}\| \geq 1$
- if  $A$  is nonsingular:  $1/\|A^{-1}\| = \min_{x \neq 0} (\|Ax\|/\|x\|)$
- $\|A^T\| = \|A\|$

# Outline

- condition of a mathematical problem
- matrix norm
- **condition number**

## Bound on absolute error

suppose  $A$  is nonsingular

$$x = A^{-1}b, \quad \Delta x = A^{-1}\Delta b$$

### Upper bound on $\|\Delta x\|$

$$\|\Delta x\| \leq \|A^{-1}\| \|\Delta b\|$$

(follows from property 4 on page 14-11)

- small  $\|A^{-1}\|$  means that  $\|\Delta x\|$  is small when  $\|\Delta b\|$  is small
- large  $\|A^{-1}\|$  means that  $\|\Delta x\|$  can be large, even when  $\|\Delta b\|$  is small
- for every  $A$ , there exists  $\Delta b$  such that  $\|\Delta x\| = \|A^{-1}\| \|\Delta b\|$  (no proof)

## Bound on relative error

suppose in addition that  $b \neq 0$ ; hence  $x \neq 0$

**Upper bound** on  $\|\Delta x\|/\|x\|$

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|\Delta b\|}{\|b\|} \quad (1)$$

(follows from  $\|\Delta x\| \leq \|A^{-1}\| \|\Delta b\|$  and  $\|b\| \leq \|A\| \|x\|$ )

- $\|A\| \|A^{-1}\|$  small means  $\|\Delta x\|/\|x\|$  is small when  $\|\Delta b\|/\|b\|$  is small
- $\|A\| \|A^{-1}\|$  large means  $\|\Delta x\|/\|x\|$  can be much larger than  $\|\Delta b\|/\|b\|$
- for every  $A$ , there exist  $b, \Delta b$  such that equality holds in (1) (no proof)

# Condition number

**Definition:** the condition number of a nonsingular matrix  $A$  is

$$\kappa(A) = \|A\| \|A^{-1}\|$$

## Properties

- $\kappa(A) \geq 1$  for all  $A$  (last property on page 14-11)
- $A$  is a *well-conditioned* matrix if  $\kappa(A)$  is small (close to 1):  
the relative error in  $x$  is not much larger than the relative error in  $b$
- $A$  is *badly conditioned* or *ill-conditioned* if  $\kappa(A)$  is large:  
the relative error in  $x$  can be much larger than the relative error in  $b$

# Example

- $A$  is blurring matrix, nonsingular with condition number  $\approx 10^9$
- we apply  $A$  to image  $x$



blurred image

$$y_1 = Ax$$



blurred and noisy image

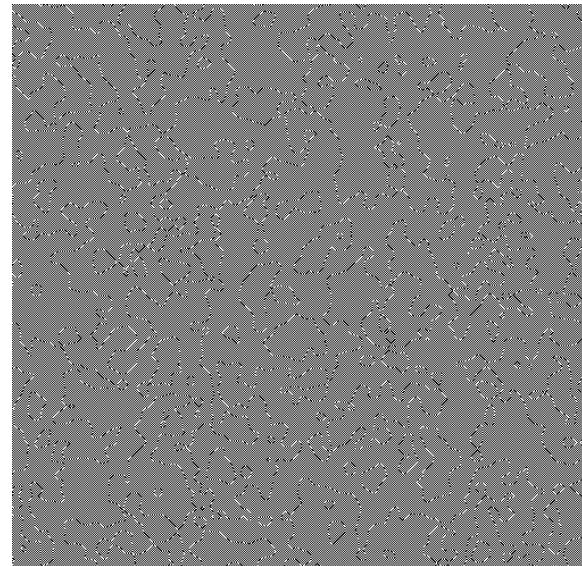
$$y_2 = Ax + \text{small noise}$$

# Example

we solve  $Ax = y$  for the two blurred images



$A^{-1}y_1$



$A^{-1}y_2$

- illustrates ill conditioning of  $A$
- explains need for regularization in deblurring algorithms

# 15. Algorithm stability

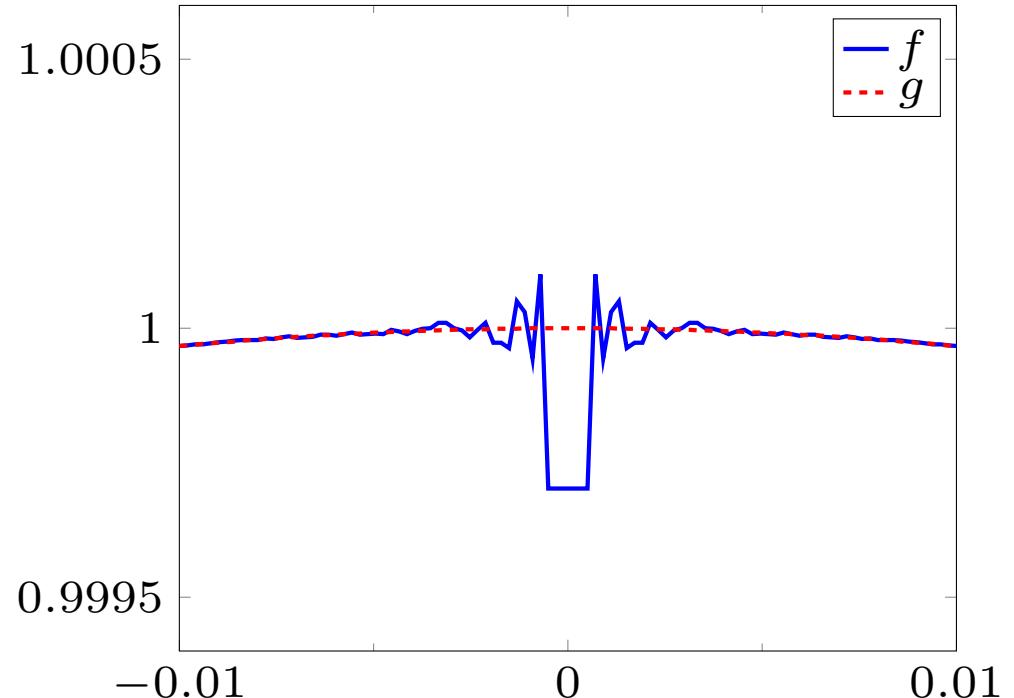
- cancellation
- numerical stability

# Example

two expressions for the same function

$$f(x) = \frac{1 - (\cos x)^2}{x^2}$$

$$g(x) = \frac{(\sin x)^2}{x^2}$$



- results of  $\cos x$  and  $\sin x$  were rounded to 10 significant digits
- other calculations are exact

## Evaluation of $f$

evaluate  $f(x)$  at  $x = 5 \cdot 10^{-5}$

- calculate  $\cos x$  and round result to 10 digits

$$\begin{aligned}\cos x &= 0.99999999875000\dots \\ &\rightsquigarrow 0.9999999988\end{aligned}$$

- evaluate  $f(x) = (1 - \cos(x)^2)/x^2$  using rounded value of  $\cos x$

$$\frac{1 - (0.9999999988)^2}{(5 \cdot 10^{-5})^2} = 0.9599\dots$$

has only one correct significant digit (correct value is 0.9999\dots)

## Evaluation of $g$

evaluate  $g(x)$  at  $x = 5 \cdot 10^{-5}$

- calculate  $\sin x$  and round result to 10 digits

$$\begin{aligned}\sin x &= 0.499999999791667\dots \cdot 10^{-5} \\ &\rightsquigarrow 0.4999999998 \cdot 10^{-5}\end{aligned}$$

- evaluate  $f(x) = \sin(x)^2/x^2$  using rounded value of  $\cos x$

$$\frac{(\sin x)^2}{x^2} \approx \frac{(0.4999999998 \cdot 10^{-5})^2}{(5 \cdot 10^{-5})^2} = 0.9999\dots$$

has about ten correct significant digits

**Conclusion:**  $f$  and  $g$  are equivalent mathematically, but not numerically

# Cancellation

$$\hat{a} = a(1 + \Delta a), \quad \hat{b} = b(1 + \Delta b)$$

- $a, b$ : exact data;  $\hat{a}, \hat{b}$ : approximations;  $\Delta a, \Delta b$ : unknown relative errors
- relative error in  $\hat{x} = \hat{a} - \hat{b} = (a - b) + (a\Delta a - b\Delta b)$  is

$$\frac{|\hat{x} - x|}{|x|} = \frac{|a\Delta a - b\Delta b|}{|a - b|}$$

if  $a \simeq b$ , small  $\Delta a$  and  $\Delta b$  can lead to very large relative errors in  $x$

this is called **cancellation**; cancellation occurs when:

- we subtract two numbers that are almost equal
- one or both numbers are subject to error

## Example

cancellation occurs in the example when we evaluate the numerator of

$$f(x) = \frac{1 - (\cos x)^2}{x^2}$$

- $1 \simeq (\cos x)^2$  when  $x$  is small
- there is a rounding error in  $\cos x$

# Numerical stability

refers to the accuracy of an algorithm in the presence of rounding errors

- an algorithm is *unstable* if rounding errors cause large errors in the result
- rigorous definition depends on what ‘accurate’ and ‘large error’ mean
- instability is often, but not always, caused by cancellation

## Examples from earlier lectures

- solving linear equations by LU factorization without pivoting
- Cholesky factorization method for least-squares

# Roots of a quadratic equation

$$ax^2 + bx + c = 0 \quad (a \neq 0)$$

**Algorithm 1:** use the formulas

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

unstable if  $b^2 \gg |4ac|$

- if  $b^2 \gg |4ac|$  and  $b \leq 0$ , cancellation occurs in  $x_2$  ( $-b \simeq \sqrt{b^2 - 4ac}$ )
- if  $b^2 \gg |4ac|$  and  $b \geq 0$ , cancellation occurs in  $x_1$  ( $b \simeq \sqrt{b^2 - 4ac}$ )
- in both cases  $b$  may be exact, but the squareroot introduces small errors

## Roots of a quadratic equation

$$ax^2 + bx + c = 0 \quad (a \neq 0)$$

**Algorithm 2:** use fact that roots  $x_1, x_2$  satisfy  $x_1x_2 = c/a$

- if  $b \leq 0$ , calculate

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{c}{ax_1}$$

- if  $b > 0$ , calculate

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \quad x_1 = \frac{c}{ax_2}$$

no cancellation when  $b^2 \gg |4ac|$

## Exercises

- suppose  $\text{chop}(x, n)$  rounds  $x$  to  $n$  decimal digits
- for example  $\text{chop}(\pi, 4)$  returns 3.14200000000000

**Exercise 1:** cancellation occurs in  $(1 - \cos x)/\sin x$  when  $x \approx 0$

```
>> x = 1e-2;  
>> (1 - chop(cos(x), 4)) / chop(sin(x), 4)
```

ans =

0

(exact value is about 0.005)

give a stable alternative method

## Exercise 2: evaluate

$$\sum_{k=1}^{3000} k^{-2} = 1.6446$$

rounding all intermediate results to 4 digits

```
>> sum = 0;  
>> for k = 1:3000  
    sum = chop(sum + 1/k^2, 4);  
end  
>> sum  
  
sum =  
  
1.6240
```

- result has only two correct digits
- not caused by cancellation (there are no subtractions)

explain and propose a better method

**Exercise 3:** the number  $e = 2.7182818\dots$  can be defined as

$$e = \lim_{n \rightarrow \infty} (1 + 1/n)^n$$

this suggests an algorithm for calculating  $e$ : take a large  $n$  and evaluate

$$\hat{e} = (1 + 1/n)^n$$

results:

| $n$       | $\hat{e}$   | # correct digits |
|-----------|-------------|------------------|
| $10^4$    | 2.718145926 | 4                |
| $10^8$    | 2.718281798 | 7                |
| $10^{12}$ | 2.718523496 | 4                |
| $10^{16}$ | 1.000000000 | 0                |

explain

**Exercise 4:** on page 2-10 we showed that for an  $n$ -vector  $x$ ,

$$\text{std}(x)^2 = \frac{1}{n} \|x - \text{avg}(x)\mathbf{1}\|^2 = \frac{1}{n} \left( \|x\|^2 - \frac{(\mathbf{1}^T x)^2}{n} \right)$$

we evaluate the second expression for  $n = 10$  and

$$x = (1002, 1000, 1003, 1001, 1002, 1002, 1001, 1004, 1002, 1001)$$

```
>> sum1 = 0.0;  sum2 = 0.0;
>> for i = 1:n
    sum1 = chop( sum1 + x(i), 6 );
    sum2 = chop( sum2 + x(i)^2, 6 );
>> end
>> s = chop( ( sum2 - sum1^2 / n ) / n, 6)

s =
-3.2400
```

a negative number! explain and suggest a better method

# 16. IEEE floating point numbers

- floating point numbers with base 10
- floating point numbers with base 2
- IEEE floating point standard
- machine precision
- rounding error

# Floating point numbers with base 10

$$x = \pm (.d_1 d_2 \dots d_n)_{10} \cdot 10^e$$

- $.d_1 d_2 \dots d_n$  is the *mantissa* ( $d_i$  integer,  $0 \leq d_i \leq 9$ ,  $d_1 \neq 0$  if  $x \neq 0$ )
- $n$  is the *mantissa length* (or *precision*)
- $e$  is the *exponent* ( $e_{\min} \leq e \leq e_{\max}$ )

**Interpretation:**  $x = \pm(d_1 10^{-1} + d_2 10^{-2} + \dots + d_n 10^{-n}) \cdot 10^e$

**Example** (with  $n = 6$ ):

$$\begin{aligned} 12.625 &= +(.126250)_{10} \cdot 10^2 \\ &= +(1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 6 \cdot 10^{-3} + 2 \cdot 10^{-4} \\ &\quad + 5 \cdot 10^{-5} + 0 \cdot 10^{-6}) \cdot 10^2 \end{aligned}$$

used in pocket calculators

# Properties

- a finite set of numbers
- unevenly spaced: distance between floating point numbers varies
  - the smallest number greater than 1 is  $1 + 10^{-n+1}$
  - the smallest number greater than 10 is  $10 + 10^{-n+2}, \dots$
- largest positive number:

$$+(.999\dots 9)_{10} \cdot 10^{e_{\max}} = (1 - 10^{-n})10^{e_{\max}}$$

- smallest positive number:

$$x_{\min} = +(.100\dots 0)_{10} \cdot 10^{e_{\min}} = 10^{e_{\min}-1}$$

## Floating point numbers with base 2

$$x = \pm(.d_1 d_2 \dots d_n)_2 \cdot 2^e$$

- $.d_1 d_2 \dots d_n$  is the *mantissa* ( $d_i \in \{0, 1\}$ ,  $d_1 = 1$  if  $x \neq 0$ )
- $n$  is the *mantissa length* (or *precision*)
- $e$  is the *exponent* ( $e_{\min} \leq e \leq e_{\max}$ )

**Interpretation:**  $x = \pm(d_1 2^{-1} + d_2 2^{-2} + \dots + d_n 2^{-n}) \cdot 2^e$

**Example** (with  $n = 8$ ):

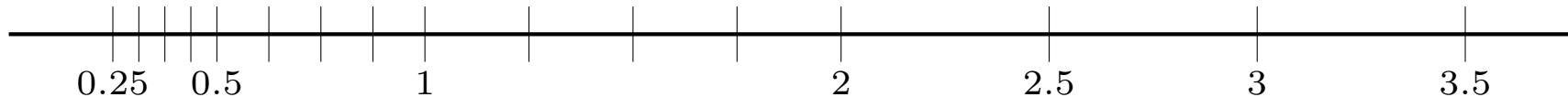
$$\begin{aligned} 12.625 &= +(.11001010)_2 \cdot 2^4 \\ &= +(1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5} \\ &\quad + 0 \cdot 2^{-6} + 1 \cdot 2^{-7} + 0 \cdot 2^{-8}) \cdot 2^4 \end{aligned}$$

used in almost all computers

## Small example

we enumerate all positive floating point numbers for

$$n = 3, \quad e_{\min} = -1, \quad e_{\max} = 2$$



$$+(.100)_2 \cdot 2^{-1} = 0.2500$$

$$+(.101)_2 \cdot 2^{-1} = 0.3125$$

$$+(.110)_2 \cdot 2^{-1} = 0.3750$$

$$+(.111)_2 \cdot 2^{-1} = 0.4375$$

$$+(.100)_2 \cdot 2^0 = 0.500$$

$$+(.101)_2 \cdot 2^0 = 0.625$$

$$+(.110)_2 \cdot 2^0 = 0.750$$

$$+(.111)_2 \cdot 2^0 = 0.875$$

$$+(.100)_2 \cdot 2^1 = 1.00$$

$$+(.101)_2 \cdot 2^1 = 1.25$$

$$+(.110)_2 \cdot 2^1 = 1.50$$

$$+(.111)_2 \cdot 2^1 = 1.75$$

$$+(.100)_2 \cdot 2^2 = 2.0$$

$$+(.101)_2 \cdot 2^2 = 2.5$$

$$+(.110)_2 \cdot 2^2 = 3.0$$

$$+(.111)_2 \cdot 2^2 = 3.5$$

# Properties

for the floating point number system on page 16-4

- a finite set of unevenly spaced numbers
- the largest positive number is

$$x_{\max} = +(.111 \dots 1)_2 \cdot 2^{e_{\max}} = (1 - 2^{-n})2^{e_{\max}}$$

- the smallest positive number is

$$x_{\min} = +(.100 \dots 0)_2 \cdot 2^{e_{\min}} = 2^{e_{\min}-1}$$

in practice, the number system includes ‘*subnormal*’ numbers

- unnormalized small numbers:  $d_1 = 0$ ,  $e = e_{\min}$
- includes the number 0

# **IEEE standard for binary arithmetic**

specifies two binary floating point number formats

## **IEEE standard single precision**

$$n = 24, \quad e_{\min} = -125, \quad e_{\max} = 128$$

requires 32 bits: 1 sign bit, 23 bits for mantissa, 8 bits for exponent

## **IEEE standard double precision**

$$n = 53, \quad e_{\min} = -1021, \quad e_{\max} = 1024$$

requires 64 bits: 1 sign bit, 52 bits for mantissa, 11 bits for exponent

used in almost all modern computers

# Machine precision

**Machine precision** (of the binary floating point number system on p.16-4)

$$\epsilon_M = 2^{-n}$$

$n$  is the mantissa length

**Example:** IEEE standard double precision

$$n = 53, \quad \epsilon_M = 2^{-53} \simeq 1.1102 \cdot 10^{-16}$$

## Interpretation

$1 + 2\epsilon_M$  is the smallest floating point number greater than 1:

$$(.10 \cdots 01)_2 \cdot 2^1 = 1 + 2^{1-n} = 1 + 2\epsilon_M$$

# Rounding

- a floating-point number system is a finite set of numbers
- all other numbers must be rounded
- we use the notation  $\text{fl}(x)$  for the floating-point representation of  $x$

## Rounding rules

- numbers are rounded to the nearest floating-point number
- ties are resolved by rounding to the number with least significant bit 0 ('round to nearest even')

**Example:** numbers  $x \in (1, 1 + 2\epsilon_M)$  are rounded to 1 or  $1 + 2\epsilon_M$

$$\text{fl}(x) = 1 \quad \text{for } 1 \leq x \leq 1 + \epsilon_M$$

$$\text{fl}(x) = 1 + 2\epsilon_M \quad \text{for } 1 + \epsilon_M < x \leq 1 + 2\epsilon_M$$

therefore numbers between  $1$  and  $1 + \epsilon_M$  are indistinguishable from  $1$

# Rounding error and machine precision

general bound on the rounding error:

$$\frac{|\text{fl}(x) - x|}{|x|} \leq \epsilon_M$$

- machine precision gives a bound on the relative error due to rounding
- number of correct (decimal) digits in  $\text{fl}(x)$  is roughly

$$-\log_{10} \epsilon_M$$

i.e., about 15 or 16 in IEEE double precision

- fundamental limit on accuracy of numerical computations

# Exercises

**Exercise 1:** explain the following results in MATLAB

```
>> (1 + 1e-16) - 1  
ans = 0
```

```
>> (1 + 2e-16) - 1  
ans = 2.2204e-16
```

```
>> (1 - 1e-16) - 1  
ans = -1.1102e-16
```

```
>> 1 + (1e-16 - 1)  
ans = 1.1102e-16
```

**Exercise 2:** run the following code in MATLAB and explain the result

```
x = 2;
for i=1:54
    x = sqrt(x);
end;
for i=1:54
    x = x^2;
end
```

**Exercise 3:** explain the following results ( $\log(1 + x)/x \approx 1$  for small  $x$ )

```
>> log(1 + 3e-16) / 3e-16
ans = 0.7401
```

```
>> log(1 + 3e-16) / ((1 + 3e-16) - 1)
ans = 1.0000
```

**Exercise 4:** the function  $f(x) = 1$  for  $x \in [10^{-16}, 10^{-15}]$ , evaluated as

$$((1 + x) - 1) / (1 + (x - 1))$$

