

Lecture slides for  
Introduction to Applied Linear Algebra:  
Vectors, Matrices, and Least Squares

Stephen Boyd   Lieven Vandenberghe

# 1. Vectors

# Outline

Notation

Examples

Addition and scalar multiplication

Inner product

Complexity

## Vectors

- ▶ a vector is an ordered list of numbers
- ▶ written as

$$\begin{bmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{bmatrix} \quad \text{or} \quad \begin{pmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{pmatrix}$$

or  $(-1.1, 0, 3.6, -7.2)$

- ▶ numbers in the list are the *elements* (*entries*, *coefficients*, *components*)
- ▶ number of elements is the *size* (*dimension*, *length*) of the vector
- ▶ vector above has dimension 4; its third entry is 3.6
- ▶ vector of size  $n$  is called an  $n$ -vector
- ▶ numbers are called *scalars*

## Vectors via symbols

- ▶ we'll use symbols to denote vectors, e.g.,  $a, X, p, \beta, E^{\text{aut}}$
- ▶ other conventions:  $\mathbf{g}, \vec{a}$
- ▶  $i$ th element of  $n$ -vector  $a$  is denoted  $a_i$
- ▶ if  $a$  is vector above,  $a_3 = 3.6$
- ▶ in  $a_i$ ,  $i$  is the *index*
- ▶ for an  $n$ -vector, indexes run from  $i = 1$  to  $i = n$
- ▶ *warning:* sometimes  $a_i$  refers to the  $i$ th vector in a list of vectors
- ▶ two vectors  $a$  and  $b$  of the same size are *equal* if  $a_i = b_i$  for all  $i$
- ▶ we overload  $=$  and write this as  $a = b$

## Block vectors

- ▶ suppose  $b$ ,  $c$ , and  $d$  are vectors with sizes  $m$ ,  $n$ ,  $p$
- ▶ the *stacked vector* or *concatenation* (of  $b$ ,  $c$ , and  $d$ ) is

$$a = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$$

- ▶ also called a *block vector*, with (block) entries  $b$ ,  $c$ ,  $d$
- ▶  $a$  has size  $m + n + p$

$$a = (b_1, b_2, \dots, b_m, c_1, c_2, \dots, c_n, d_1, d_2, \dots, d_p)$$

## Zero, ones, and unit vectors

- ▶  $n$ -vector with all entries 0 is denoted  $0_n$  or just 0
- ▶  $n$ -vector with all entries 1 is denoted  $\mathbf{1}_n$  or just  $\mathbf{1}$
- ▶ a *unit vector* has one entry 1 and all others 0
- ▶ denoted  $e_i$  where  $i$  is entry that is 1
- ▶ unit vectors of length 3:

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

# Sparsity

- ▶ a vector is *sparse* if many of its entries are 0
- ▶ can be stored and manipulated efficiently on a computer
- ▶  $\text{nnz}(x)$  is number of entries that are nonzero
- ▶ examples: zero vectors, unit vectors

# Outline

Notation

Examples

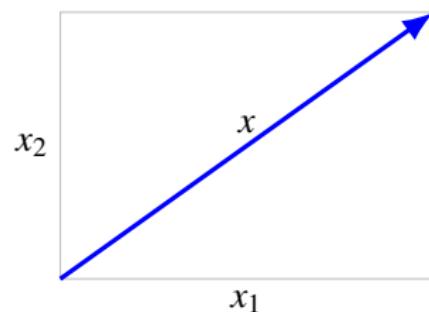
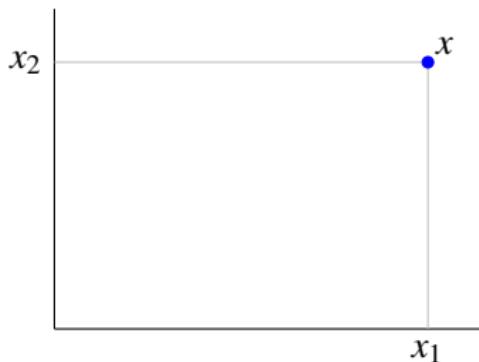
Addition and scalar multiplication

Inner product

Complexity

## Location or displacement in 2-D or 3-D

2-vector  $(x_1, x_2)$  can represent a location or a displacement in 2-D



## More examples

- ▶ color:  $(R, G, B)$
- ▶ quantities of  $n$  different commodities (or resources), e.g., bill of materials
- ▶ portfolio: entries give shares (or \$ value or fraction) held in each of  $n$  assets, with negative meaning short positions
- ▶ cash flow:  $x_i$  is payment in period  $i$  to us
- ▶ audio:  $x_i$  is the acoustic pressure at sample time  $i$   
(sample times are spaced  $1/44100$  seconds apart)
- ▶ features:  $x_i$  is the value of  $i$ th *feature* or *attribute* of an entity
- ▶ customer purchase:  $x_i$  is the total \$ purchase of product  $i$  by a customer over some period
- ▶ word count:  $x_i$  is the number of times word  $i$  appears in a document

## Word count vectors

- ▶ a short document:

**Word** count vectors are used **in** computer based **document** analysis. Each entry of the **word** count vector is the **number** of times the associated dictionary **word** appears **in** the **document**.

- ▶ a small dictionary (left) and word count vector (right)

word	3
in	2
number	1
horse	0
the	4
document	2

- ▶ dictionaries used in practice are much larger

# Outline

Notation

Examples

Addition and scalar multiplication

Inner product

Complexity

## Vector addition

- ▶  $n$ -vectors  $a$  and  $b$  can be added, with sum denoted  $a + b$
- ▶ to get sum, add corresponding entries:

$$\begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 9 \\ 3 \end{bmatrix}$$

- ▶ subtraction is similar

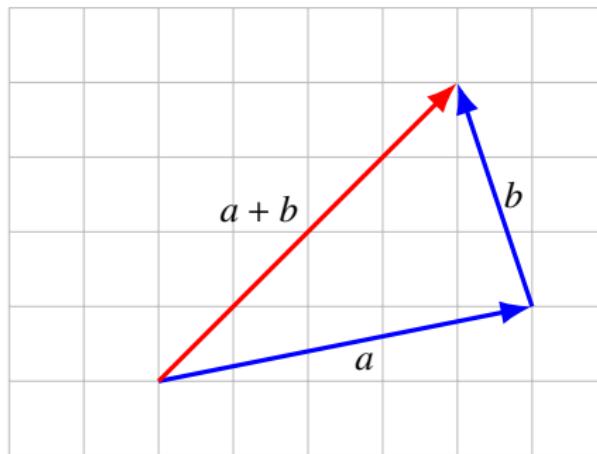
## Properties of vector addition

- ▶ *commutative*:  $a + b = b + a$
- ▶ *associative*:  $(a + b) + c = a + (b + c)$   
(so we can write both as  $a + b + c$ )
- ▶  $a + 0 = 0 + a = a$
- ▶  $a - a = 0$

these are easy and boring to verify

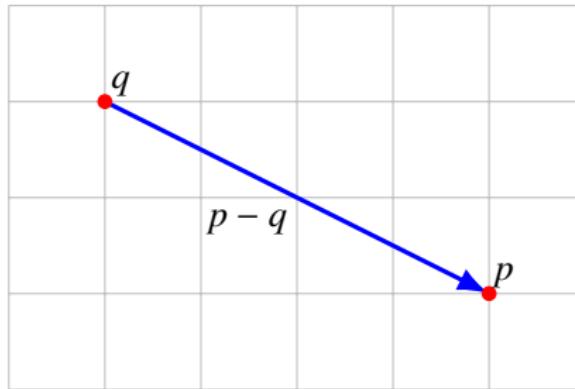
## Adding displacements

if 3-vectors  $a$  and  $b$  are displacements,  $a + b$  is the sum displacement



## Displacement from one point to another

displacement from point  $q$  to point  $p$  is  $p - q$



## Scalar-vector multiplication

- ▶ scalar  $\beta$  and  $n$ -vector  $a$  can be multiplied

$$\beta a = (\beta a_1, \dots, \beta a_n)$$

- ▶ also denoted  $a\beta$
- ▶ example:

$$(-2) \begin{bmatrix} 1 \\ 9 \\ 6 \end{bmatrix} = \begin{bmatrix} -2 \\ -18 \\ -12 \end{bmatrix}$$

## Properties of scalar-vector multiplication

- ▶ associative:  $(\beta\gamma)a = \beta(\gamma a)$
- ▶ left distributive:  $(\beta + \gamma)a = \beta a + \gamma a$
- ▶ right distributive:  $\beta(a + b) = \beta a + \beta b$

these equations look innocent, but be sure you understand them perfectly

## Linear combinations

- ▶ for vectors  $a_1, \dots, a_m$  and scalars  $\beta_1, \dots, \beta_m$ ,

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

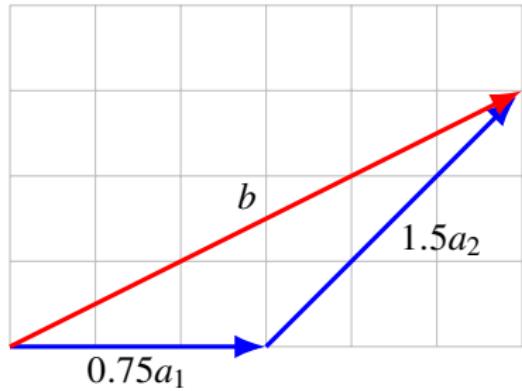
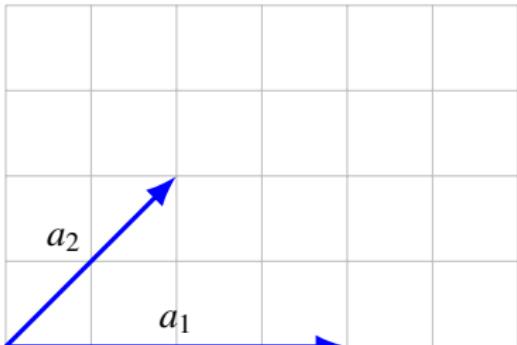
is a *linear combination* of the vectors

- ▶  $\beta_1, \dots, \beta_m$  are the *coefficients*
- ▶ a *very* important concept
- ▶ a simple identity: for any  $n$ -vector  $b$ ,

$$b = b_1 e_1 + \cdots + b_n e_n$$

## Example

two vectors  $a_1$  and  $a_2$ , and linear combination  $b = 0.75a_1 + 1.5a_2$



## Replicating a cash flow

- ▶  $c_1 = (1, -1.1, 0)$  is a \$1 loan from period 1 to 2 with 10% interest
- ▶  $c_2 = (0, 1, -1.1)$  is a \$1 loan from period 2 to 3 with 10% interest
- ▶ linear combination

$$d = c_1 + 1.1c_2 = (1, 0, -1.21)$$

is a two period loan with 10% compounded interest rate

- ▶ we have *replicated* a two period loan from two one period loans

# Outline

Notation

Examples

Addition and scalar multiplication

Inner product

Complexity

## Inner product

- ▶ *inner product* (or *dot product*) of  $n$ -vectors  $a$  and  $b$  is

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

- ▶ other notation used:  $\langle a, b \rangle$ ,  $\langle a | b \rangle$ ,  $(a, b)$ ,  $a \cdot b$
- ▶ example:

$$\begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix} = (-1)(1) + (2)(0) + (2)(-3) = -7$$

## Properties of inner product

- ▶  $a^T b = b^T a$
- ▶  $(\gamma a)^T b = \gamma(a^T b)$
- ▶  $(a + b)^T c = a^T c + b^T c$

can combine these to get, for example,

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

## General examples

- ▶  $e_i^T a = a_i$  (picks out  $i$ th entry)
- ▶  $\mathbf{1}^T a = a_1 + \cdots + a_n$  (sum of entries)
- ▶  $a^T a = a_1^2 + \cdots + a_n^2$  (sum of squares of entries)

## Examples

- ▶  $w$  is weight vector,  $f$  is feature vector;  $w^T f$  is score
- ▶  $p$  is vector of prices,  $q$  is vector of quantities;  $p^T q$  is total cost
- ▶  $c$  is cash flow,  $d$  is discount vector (with interest rate  $r$ ):

$$d = (1, 1/(1+r), \dots, 1/(1+r)^{n-1})$$

$d^T c$  is net present value (NPV) of cash flow

- ▶  $s$  gives portfolio holdings (in shares),  $p$  gives asset prices;  $p^T s$  is total portfolio value

# Outline

Notation

Examples

Addition and scalar multiplication

Inner product

Complexity

## Flop counts

- ▶ computers store (real) numbers in *floating-point format*
- ▶ basic arithmetic operations (addition, multiplication, ...) are called *floating point operations* or flops
- ▶ complexity of an algorithm or operation: total number of flops needed, as function of the input dimension(s)
- ▶ this can be *very grossly approximated*
- ▶ crude approximation of time to execute: computer speed/flops
- ▶ current computers are around 1Gflop/sec ( $10^9$  flops/sec)
- ▶ but this can vary by factor of 100

## Complexity of vector addition, inner product

- ▶  $x + y$  needs  $n$  additions, so:  $n$  flops
- ▶  $x^T y$  needs  $n$  multiplications,  $n - 1$  additions so:  $2n - 1$  flops
- ▶ we simplify this to  $2n$  (or even  $n$ ) flops for  $x^T y$
- ▶ and much less when  $x$  or  $y$  is sparse

## 2. Linear functions

# Outline

Linear and affine functions

Taylor approximation

Regression model

## Superposition and linear functions

- ▶  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  means  $f$  is a function mapping  $n$ -vectors to numbers
- ▶  $f$  satisfies the *superposition property* if

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

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

- ▶ be sure to parse this very carefully!
- ▶ a function that satisfies superposition is called *linear*

## The inner product function

- ▶ with  $a$  an  $n$ -vector, the function

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

is the *inner product function*

- ▶  $f(x)$  is a weighted sum of the entries of  $x$
- ▶ the inner product function is linear:

$$\begin{aligned} f(\alpha x + \beta y) &= a^T (\alpha x + \beta y) \\ &= a^T (\alpha x) + a^T (\beta y) \\ &= \alpha (a^T x) + \beta (a^T y) \\ &= \alpha f(x) + \beta f(y) \end{aligned}$$

## ...and all linear functions are inner products

- ▶ suppose  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  is linear
- ▶ then it can be expressed as  $f(x) = a^T x$  for some  $a$
- ▶ specifically:  $a_i = f(e_i)$
- ▶ follows from

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

## Affine functions

- ▶ a function that is linear plus a constant is called *affine*
- ▶ general form is  $f(x) = a^T x + b$ , with  $a$  an  $n$ -vector and  $b$  a scalar
- ▶ a function  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  is affine if and only if

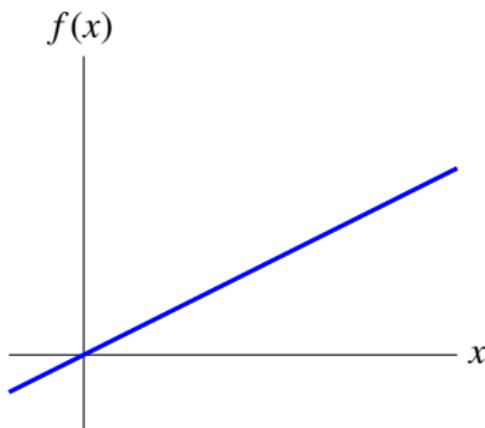
$$f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$$

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

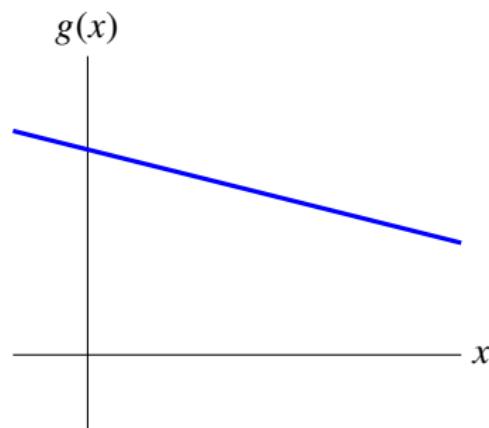
- ▶ sometimes (ignorant) people refer to affine functions as linear

## Linear versus affine functions

$f$  is linear



$g$  is affine, not linear



# Outline

Linear and affine functions

Taylor approximation

Regression model

## First-order Taylor approximation

- ▶ suppose  $f : \mathbf{R}^n \rightarrow \mathbf{R}$
- ▶ *first-order Taylor approximation* of  $f$ , near point  $z$ :

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

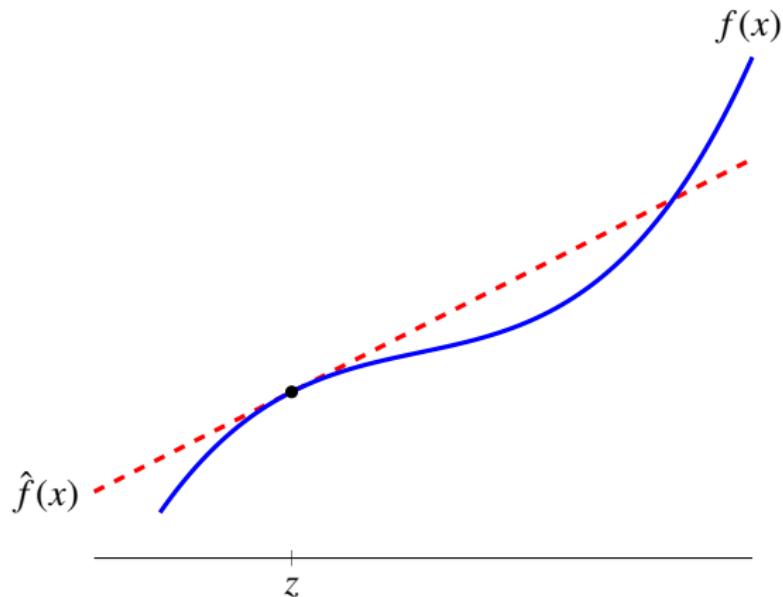
- ▶  $\hat{f}(x)$  is *very close* to  $f(x)$  when  $x_i$  are all near  $z_i$
- ▶  $\hat{f}$  is an affine function of  $x$
- ▶ can write using inner product as

$$\hat{f}(x) = f(z) + \nabla f(z)^T (x - z)$$

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

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

## Example



# Outline

Linear and affine functions

Taylor approximation

Regression model

## Regression model

- *regression model* is (the affine function of  $x$ )

$$\hat{y} = x^T \beta + v$$

- $x$  is a feature vector; its elements  $x_i$  are called *regressors*
- $n$ -vector  $\beta$  is the *weight vector*
- scalar  $v$  is the *offset*
- scalar  $\hat{y}$  is the *prediction*  
(of some actual outcome or *dependent variable*, denoted  $y$ )

## Example

- ▶  $y$  is selling price of house in \$1000 (in some location, over some period)
- ▶ regressor is

$$x = (\text{house area}, \# \text{ bedrooms})$$

(house area in 1000 sq.ft.)

- ▶ regression model weight vector and offset are

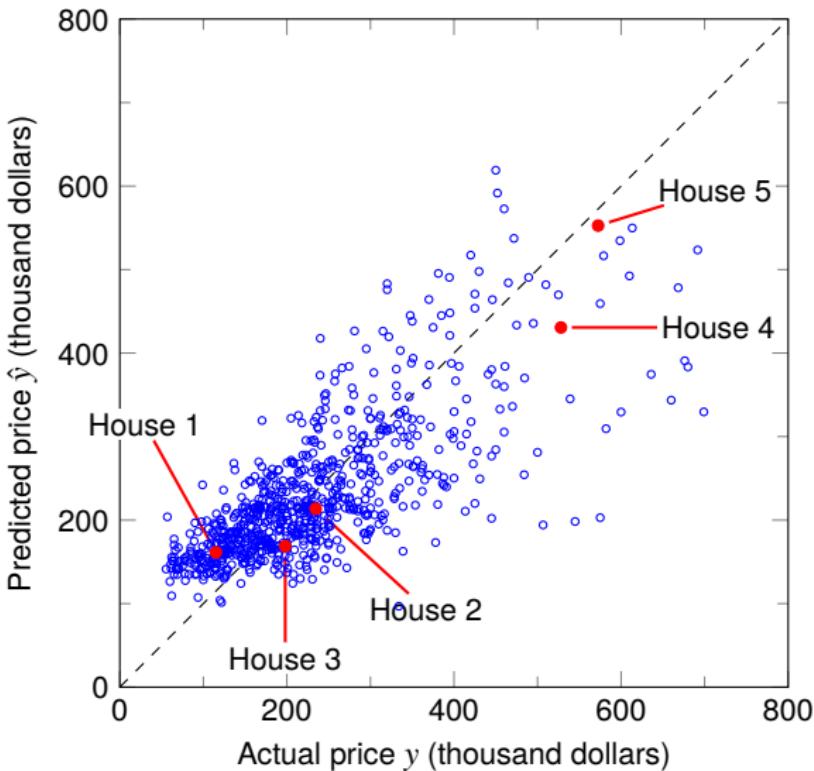
$$\beta = (148.73, -18.85), \quad v = 54.40$$

- ▶ we'll see later how to guess  $\beta$  and  $v$  from sales data

## Example

House	$x_1$ (area)	$x_2$ (beds)	$y$ (price)	$\hat{y}$ (prediction)
1	0.846	1	115.00	161.37
2	1.324	2	234.50	213.61
3	1.150	3	198.00	168.88
4	3.037	4	528.00	430.67
5	3.984	5	572.50	552.66

## Example



### 3. Norm and distance

# Outline

Norm

Distance

Standard deviation

Angle

## Norm

- ▶ the *Euclidean norm* (or just *norm*) of an  $n$ -vector  $x$  is

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{x^T x}$$

- ▶ used to measure the size of a vector
- ▶ reduces to absolute value for  $n = 1$

## Properties

for any  $n$ -vectors  $x$  and  $y$ , and any scalar  $\beta$

- ▶ *homogeneity*:  $\|\beta x\| = |\beta| \|x\|$
- ▶ *triangle inequality*:  $\|x + y\| \leq \|x\| + \|y\|$
- ▶ *nonnegativity*:  $\|x\| \geq 0$
- ▶ *definiteness*:  $\|x\| = 0$  only if  $x = 0$

easy to show except triangle inequality, which we show later

## RMS value

- ▶ mean-square value of  $n$ -vector  $x$  is

$$\frac{x_1^2 + \cdots + x_n^2}{n} = \frac{\|x\|^2}{n}$$

- ▶ root-mean-square value (RMS value) is

$$\text{rms}(x) = \sqrt{\frac{x_1^2 + \cdots + x_n^2}{n}} = \frac{\|x\|}{\sqrt{n}}$$

- ▶  $\text{rms}(x)$  gives ‘typical’ value of  $|x_i|$
- ▶ e.g.,  $\text{rms}(\mathbf{1}) = 1$  (independent of  $n$ )
- ▶ RMS value useful for comparing sizes of vectors of different lengths

## Norm of block vectors

- ▶ suppose  $a, b, c$  are vectors
- ▶  $\|(a, b, c)\|^2 = a^T a + b^T b + c^T c = \|a\|^2 + \|b\|^2 + \|c\|^2$
- ▶ so we have

$$\|(a, b, c)\| = \sqrt{\|a\|^2 + \|b\|^2 + \|c\|^2} = \|(\|a\|, \|b\|, \|c\|)\|$$

(parse RHS very carefully!)

- ▶ we'll use these ideas later

## Chebyshev inequality

- ▶ suppose that  $k$  of the numbers  $|x_1|, \dots, |x_n|$  are  $\geq a$
- ▶ then  $k$  of the numbers  $x_1^2, \dots, x_n^2$  are  $\geq a^2$
- ▶ so  $\|x\|^2 = x_1^2 + \dots + x_n^2 \geq ka^2$
- ▶ so we have  $k \leq \|x\|^2/a^2$
- ▶ number of  $x_i$  with  $|x_i| \geq a$  is no more than  $\|x\|^2/a^2$
- ▶ this is the *Chebyshev inequality*
- ▶ in terms of RMS value:

fraction of entries with  $|x_i| \geq a$  is no more than  $\left(\frac{\text{rms}(x)}{a}\right)^2$

- ▶ example: no more than 4% of entries can satisfy  $|x_i| \geq 5 \text{ rms}(x)$

# Outline

Norm

Distance

Standard deviation

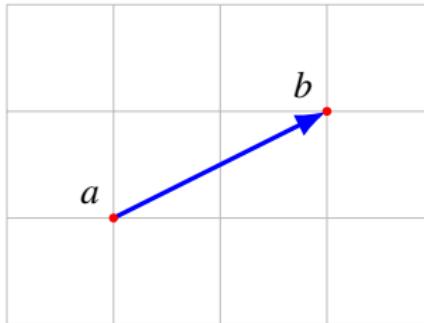
Angle

## Distance

- ▶ (Euclidean) *distance* between  $n$ -vectors  $a$  and  $b$  is

$$\mathbf{dist}(a, b) = \|a - b\|$$

- ▶ agrees with ordinary distance for  $n = 1, 2, 3$



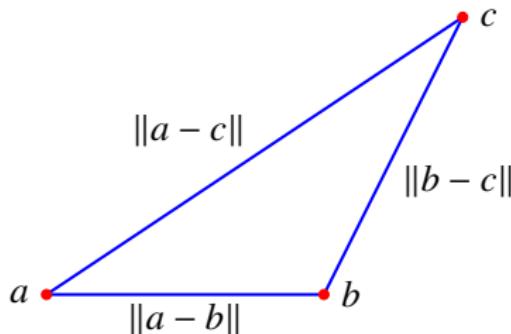
- ▶  $\text{rms}(a - b)$  is the *RMS deviation* between  $a$  and  $b$

## Triangle inequality

- ▶ triangle with vertices at positions  $a, b, c$
- ▶ edge lengths are  $\|a - b\|, \|b - c\|, \|a - c\|$
- ▶ by triangle inequality

$$\|a - c\| = \|(a - b) + (b - c)\| \leq \|a - b\| + \|b - c\|$$

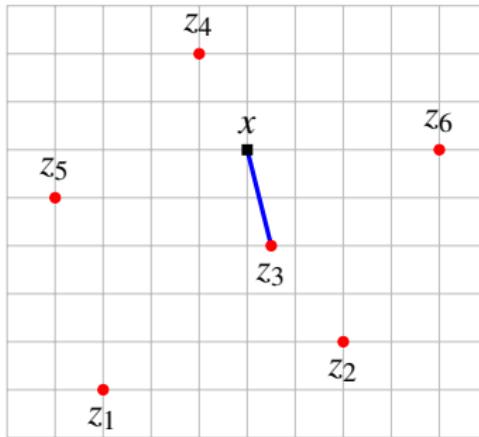
i.e., third edge length is no longer than sum of other two



## Feature distance and nearest neighbors

- ▶ if  $x$  and  $y$  are feature vectors for two entities,  $\|x - y\|$  is the *feature distance*
- ▶ if  $z_1, \dots, z_m$  is a list of vectors,  $z_j$  is the *nearest neighbor* of  $x$  if

$$\|x - z_j\| \leq \|x - z_i\|, \quad i = 1, \dots, m$$



- ▶ these simple ideas are very widely used

## Document dissimilarity

- ▶ 5 Wikipedia articles: ‘Veterans Day’, ‘Memorial Day’, ‘Academy Awards’, ‘Golden Globe Awards’, ‘Super Bowl’
- ▶ word count histograms, dictionary of 4423 words
- ▶ pairwise distances shown below

	Veterans Day	Memorial Day	Academy Awards	Golden Globe Awards	Super Bowl
Veterans Day	0	0.095	0.130	0.153	0.170
Memorial Day	0.095	0	0.122	0.147	0.164
Academy A.	0.130	0.122	0	0.108	0.164
Golden Globe A.	0.153	0.147	0.108	0	0.181
Super Bowl	0.170	0.164	0.164	0.181	0

# Outline

Norm

Distance

Standard deviation

Angle

## Standard deviation

- ▶ for  $n$ -vector  $x$ ,  $\text{avg}(x) = \mathbf{1}^T x / n$
- ▶ *de-meaned vector* is  $\tilde{x} = x - \text{avg}(x)\mathbf{1}$  (so  $\text{avg}(\tilde{x}) = 0$ )
- ▶ *standard deviation* of  $x$  is

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

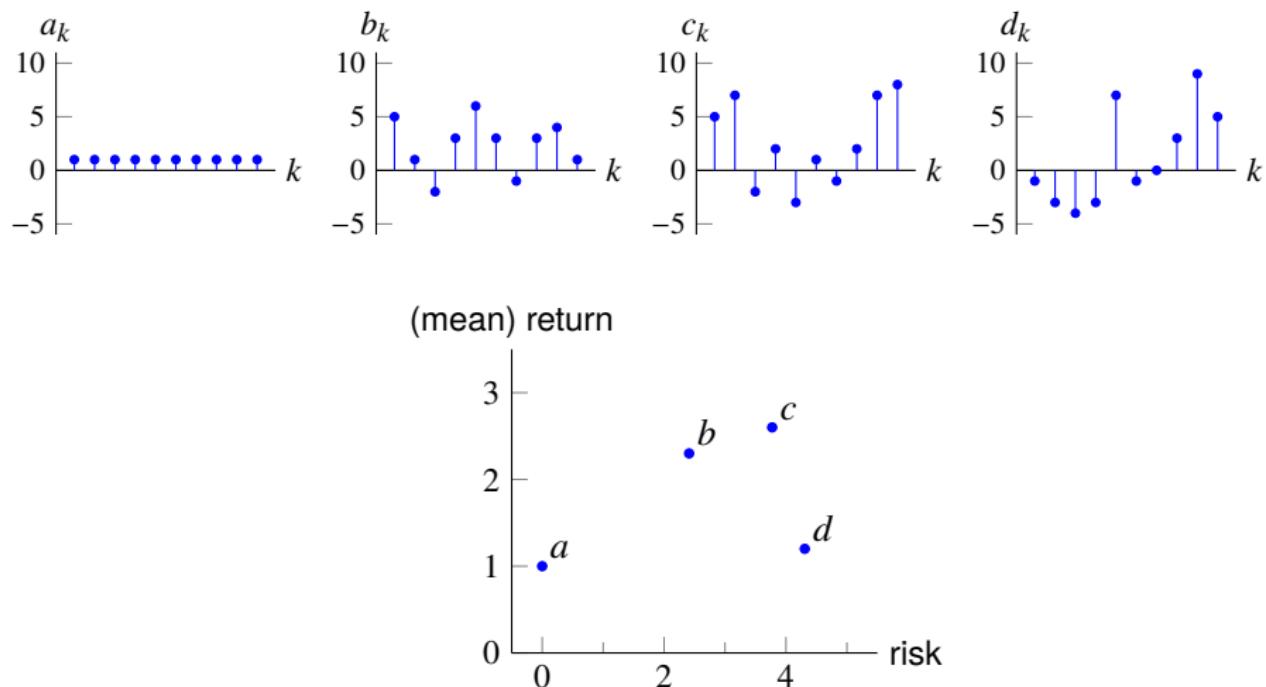
- ▶  $\text{std}(x)$  gives ‘typical’ amount  $x_i$  vary from  $\text{avg}(x)$
- ▶  $\text{std}(x) = 0$  only if  $x = \alpha \mathbf{1}$  for some  $\alpha$
- ▶ greek letters  $\mu, \sigma$  commonly used for mean, standard deviation
- ▶ a basic formula:

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

## Mean return and risk

- ▶  $x$  is time series of returns (say, in %) on some investment or asset over some period
- ▶ **avg**( $x$ ) is the mean return over the period, usually just called *return*
- ▶ **std**( $x$ ) measures how variable the return is over the period, and is called the *risk*
- ▶ multiple investments (with different return time series) are often compared in terms of return and risk
- ▶ often plotted on a *risk-return plot*

## Risk-return example



## Chebyshev inequality for standard deviation

- ▶  $x$  is an  $n$ -vector with mean  $\text{avg}(x)$ , standard deviation  $\text{std}(x)$
- ▶ rough idea: most entries of  $x$  are not too far from the mean
- ▶ by Chebyshev inequality, fraction of entries of  $x$  with

$$|x_i - \text{avg}(x)| \geq \alpha \text{ std}(x)$$

is no more than  $1/\alpha^2$  (for  $\alpha > 1$ )

- ▶ for return time series with mean 8% and standard deviation 3%, loss ( $x_i \leq 0$ ) can occur in no more than  $(3/8)^2 = 14.1\%$  of periods

# Outline

Norm

Distance

Standard deviation

Angle

## Cauchy–Schwarz inequality

- ▶ for two  $n$ -vectors  $a$  and  $b$ ,  $|a^T b| \leq \|a\| \|b\|$
- ▶ written out,

$$|a_1 b_1 + \cdots + a_n b_n| \leq (a_1^2 + \cdots + a_n^2)^{1/2} (b_1^2 + \cdots + b_n^2)^{1/2}$$

- ▶ now we can show triangle inequality:

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

## Derivation of Cauchy–Schwarz inequality

- ▶ it's clearly true if either  $a$  or  $b$  is 0
- ▶ so assume  $\alpha = \|a\|$  and  $\beta = \|b\|$  are nonzero
- ▶ we have

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

- ▶ divide by  $2\|a\| \|b\|$  to get  $a^T b \leq \|a\| \|b\|$
- ▶ apply to  $-a, b$  to get other half of Cauchy–Schwarz inequality

## Angle

- ▶ angle between two nonzero vectors  $a, b$  defined as

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

- ▶  $\angle(a, b)$  is the number in  $[0, \pi]$  that satisfies

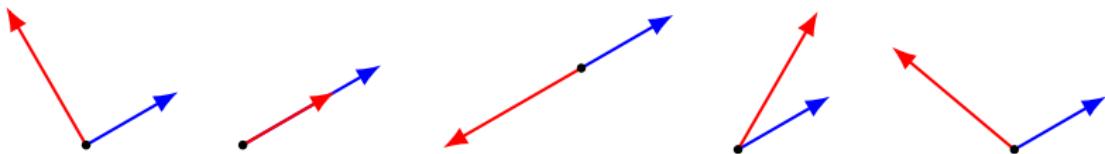
$$a^T b = \|a\| \|b\| \cos(\angle(a, b))$$

- ▶ coincides with ordinary angle between vectors in 2-D and 3-D

## Classification of angles

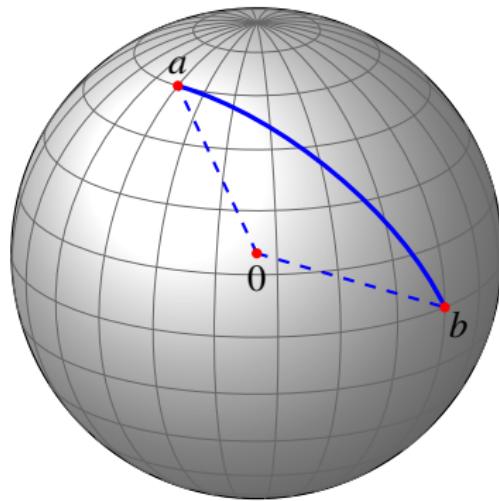
$$\theta = \angle(a, b)$$

- ▶  $\theta = \pi/2 = 90^\circ$ :  $a$  and  $b$  are *orthogonal*, written  $a \perp b$  ( $a^T b = 0$ )
- ▶  $\theta = 0$ :  $a$  and  $b$  are *aligned* ( $a^T b = \|a\| \|b\|$ )
- ▶  $\theta = \pi = 180^\circ$ :  $a$  and  $b$  are *anti-aligned* ( $a^T b = -\|a\| \|b\|$ )
- ▶  $\theta \leq \pi/2 = 90^\circ$ :  $a$  and  $b$  make an *acute angle* ( $a^T b \geq 0$ )
- ▶  $\theta \geq \pi/2 = 90^\circ$ :  $a$  and  $b$  make an *obtuse angle* ( $a^T b \leq 0$ )



## Spherical distance

if  $a, b$  are on sphere of radius  $R$ , distance *along the sphere* is  $R\angle(a,b)$



## Document dissimilarity by angles

- ▶ measure dissimilarity by angle of word count histogram vectors
- ▶ pairwise angles (in degrees) for 5 Wikipedia pages shown below

	Veterans Day	Memorial Day	Academy Awards	Golden Globe Awards	Super Bowl
Veterans Day	0	60.6	85.7	87.0	87.7
Memorial Day	60.6	0	85.6	87.5	87.5
Academy A.	85.7	85.6	0	58.7	85.7
Golden Globe A.	87.0	87.5	58.7	0	86.0
Super Bowl	87.7	87.5	86.1	86.0	0

## Correlation coefficient

- ▶ vectors  $a$  and  $b$ , and de-meaned vectors

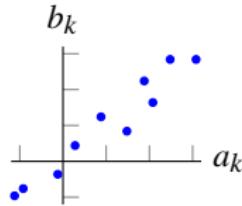
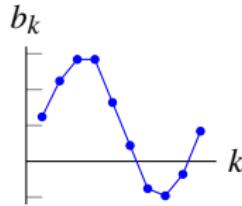
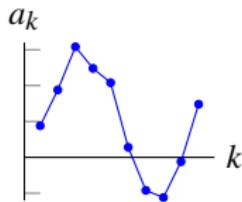
$$\tilde{a} = a - \text{avg}(a)\mathbf{1}, \quad \tilde{b} = b - \text{avg}(b)\mathbf{1}$$

- ▶ correlation coefficient (between  $a$  and  $b$ , with  $\tilde{a} \neq 0, \tilde{b} \neq 0$ )

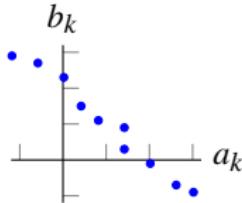
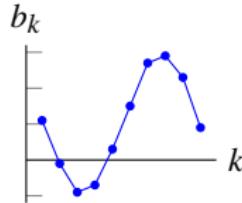
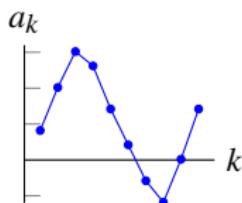
$$\rho = \frac{\tilde{a}^T \tilde{b}}{\|\tilde{a}\| \|\tilde{b}\|}$$

- ▶  $\rho = \cos \angle(\tilde{a}, \tilde{b})$ 
  - $\rho = 0$ :  $a$  and  $b$  are *uncorrelated*
  - $\rho > 0.8$  (or so):  $a$  and  $b$  are *highly correlated*
  - $\rho < -0.8$  (or so):  $a$  and  $b$  are *highly anti-correlated*
- ▶ very roughly: highly correlated means  $a_i$  and  $b_i$  are typically both above (below) their means together

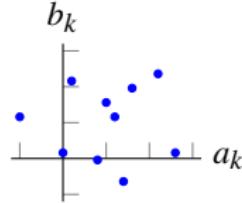
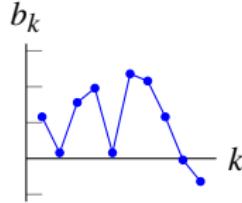
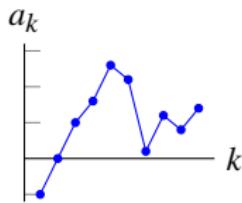
## Examples



$$\rho = 97\%$$



$$\rho = -99\%$$



$$\rho = 0.4\%$$

## Examples

- ▶ highly correlated vectors:
  - rainfall time series at nearby locations
  - daily returns of similar companies in same industry
  - word count vectors of closely related documents  
(e.g., same author, topic, ...)
  - sales of shoes and socks (at different locations or periods)
- ▶ approximately uncorrelated vectors
  - unrelated vectors
  - audio signals (even different tracks in multi-track recording)
- ▶ (somewhat) negatively correlated vectors
  - daily temperatures in Palo Alto and Melbourne

## 4. Clustering

# Outline

Clustering

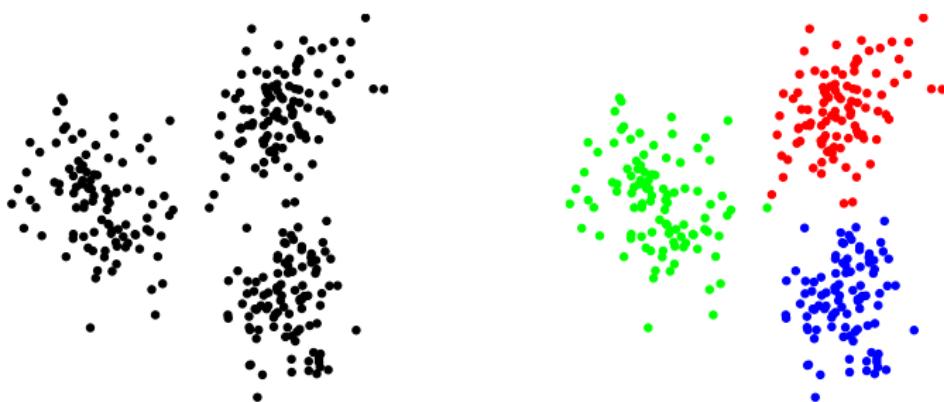
Algorithm

Examples

Applications

# Clustering

- ▶ given  $N$   $n$ -vectors  $x_1, \dots, x_N$
- ▶ goal: partition (divide, cluster) into  $k$  groups
- ▶ want vectors in the same group to be close to one another



## Example settings

- ▶ topic discovery and document classification
  - $x_i$  is word count histogram for document  $i$
- ▶ patient clustering
  - $x_i$  are patient attributes, test results, symptoms
- ▶ customer market segmentation
  - $x_i$  is purchase history and other attributes of customer  $i$
- ▶ color compression of images
  - $x_i$  are RGB pixel values
- ▶ financial sectors
  - $x_i$  are  $n$ -vectors of financial attributes of company  $i$

## Clustering objective

- ▶  $G_j \subset \{1, \dots, N\}$  is group  $j$ , for  $j = 1, \dots, k$
- ▶  $c_i$  is group that  $x_i$  is in:  $i \in G_{c_i}$
- ▶ group *representatives*:  $n$ -vectors  $z_1, \dots, z_k$
- ▶ clustering objective is

$$J^{\text{clust}} = \frac{1}{N} \sum_{i=1}^N \|x_i - z_{c_i}\|^2$$

mean square distance from vectors to associated representative

- ▶  $J^{\text{clust}}$  small means good clustering
- ▶ goal: choose clustering  $c_i$  and representatives  $z_j$  to minimize  $J^{\text{clust}}$

# Outline

Clustering

Algorithm

Examples

Applications

## Partitioning the vectors given the representatives

- ▶ suppose representatives  $z_1, \dots, z_k$  are given
- ▶ how do we assign the vectors to groups, i.e., choose  $c_1, \dots, c_N$ ?
- ▶  $c_i$  only appears in term  $\|x_i - z_{c_i}\|^2$  in  $J^{\text{clust}}$
- ▶ to minimize over  $c_i$ , choose  $c_i$  so  $\|x_i - z_{c_i}\|^2 = \min_j \|x_i - z_j\|^2$
- ▶ i.e., *assign each vector to its nearest representative*

## Choosing representatives given the partition

- ▶ given the partition  $G_1, \dots, G_k$ , how do we choose representatives  $z_1, \dots, z_k$  to minimize  $J^{\text{clust}}$ ?
- ▶  $J^{\text{clust}}$  splits into a sum of  $k$  sums, one for each  $z_j$ :

$$J^{\text{clust}} = J_1 + \cdots + J_k, \quad J_j = (1/N) \sum_{i \in G_j} \|x_i - z_j\|^2$$

- ▶ so we choose  $z_j$  to minimize mean square distance to the points in its partition
- ▶ this is the mean (or average or centroid) of the points in the partition:

$$z_j = (1/|G_j|) \sum_{i \in G_j} x_i$$

## *k*-means algorithm

- ▶ alternate between updating the partition, then the representatives
  - ▶ a famous algorithm called *k-means*
  - ▶ objective  $J^{\text{clust}}$  decreases in each step
- 

**given**  $x_1, \dots, x_N \in \mathbf{R}^n$  and  $z_1, \dots, z_k \in \mathbf{R}^n$

**repeat**

*Update partition:* assign  $i$  to  $G_j$ ,  $j = \operatorname{argmin}_{j'} \|x_i - z_{j'}\|^2$

*Update centroids:*  $z_j = \frac{1}{|G_j|} \sum_{i \in G_j} x_i$

**until**  $z_1, \dots, z_k$  stop changing

---

## Convergence of $k$ -means algorithm

- ▶  $J^{\text{clust}}$  goes down in each step, until the  $z_j$ 's stop changing
- ▶ but (in general) the  $k$ -means algorithm *does not find the partition that minimizes  $J^{\text{clust}}$*
- ▶  $k$ -means is a *heuristic*: it is not guaranteed to find the smallest possible value of  $J^{\text{clust}}$
- ▶ the final partition (and its value of  $J^{\text{clust}}$ ) can depend on the initial representatives
- ▶ common approach:
  - run  $k$ -means 10 times, with different (often random) initial representatives
  - take as final partition the one with the smallest value of  $J^{\text{clust}}$

# Outline

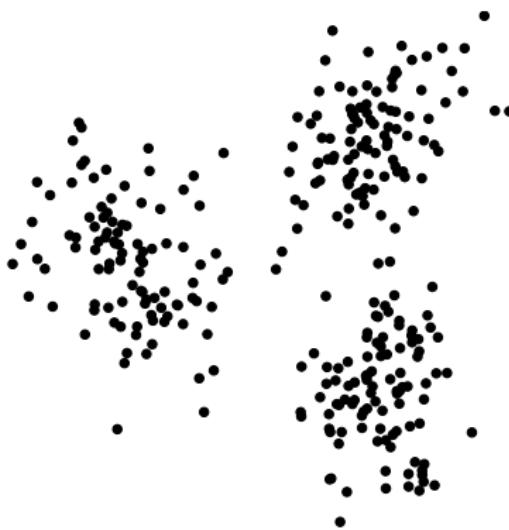
Clustering

Algorithm

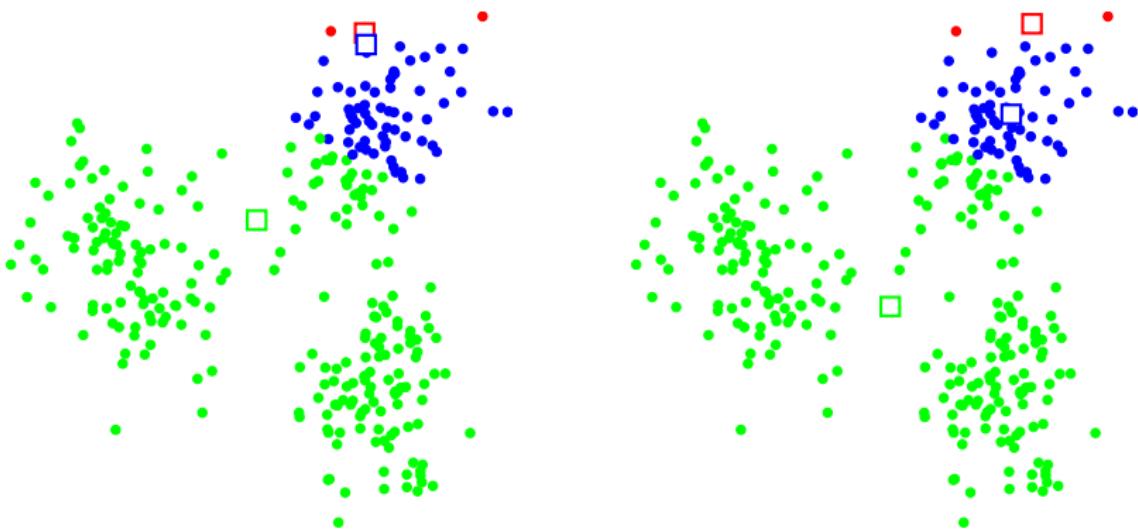
Examples

Applications

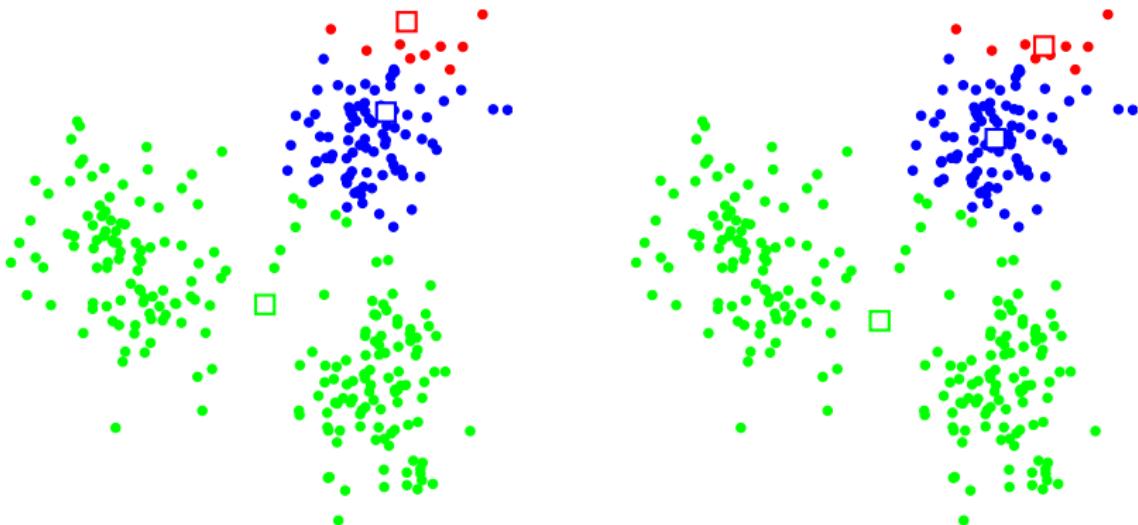
# Data



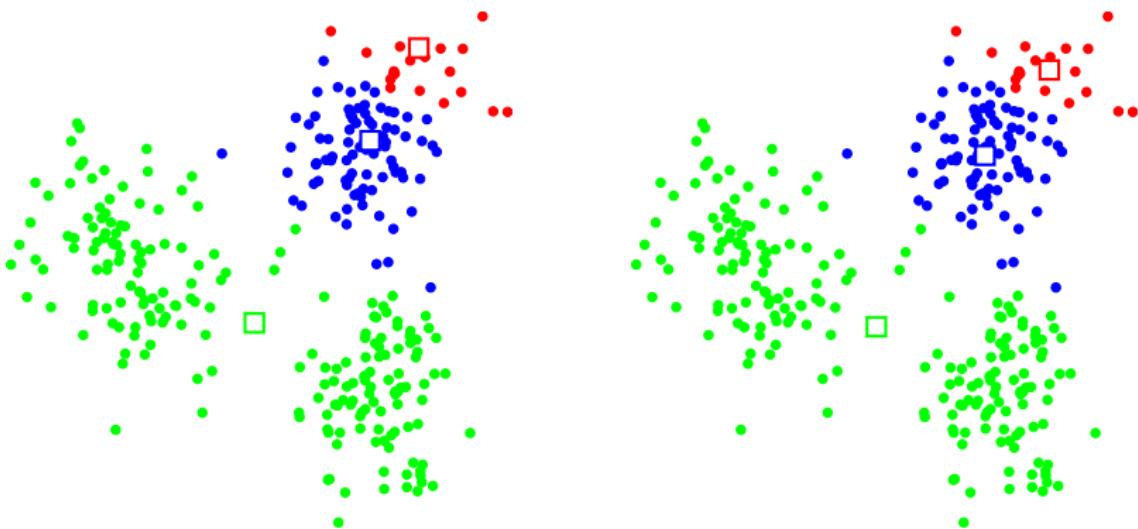
## Iteration 1



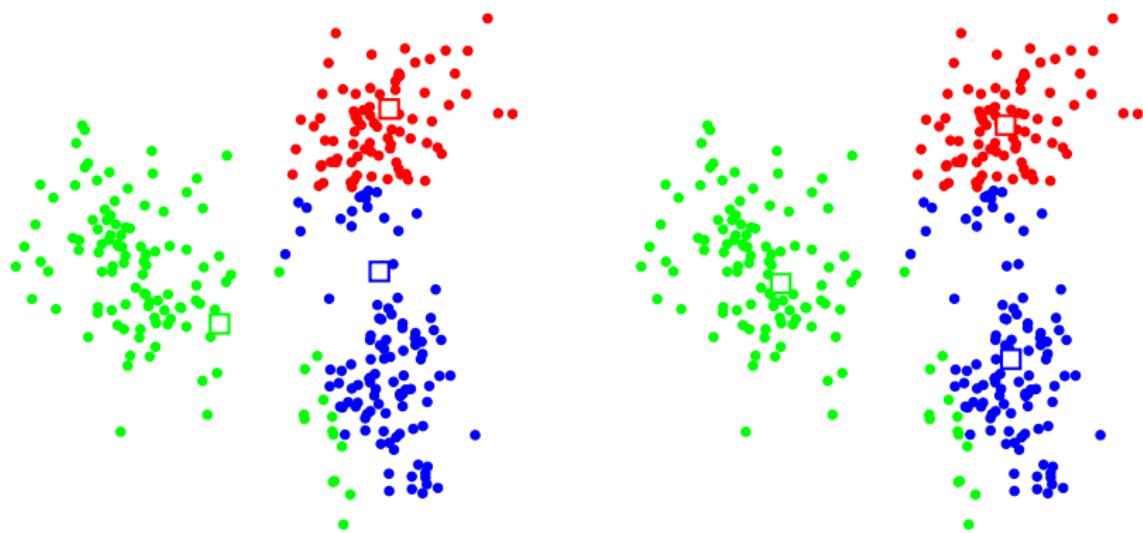
## Iteration 2



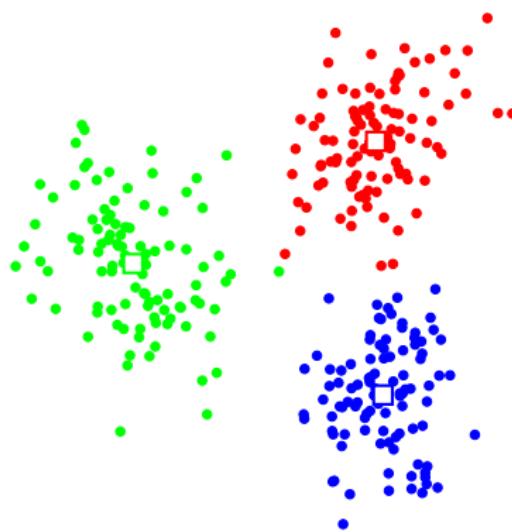
## Iteration 3



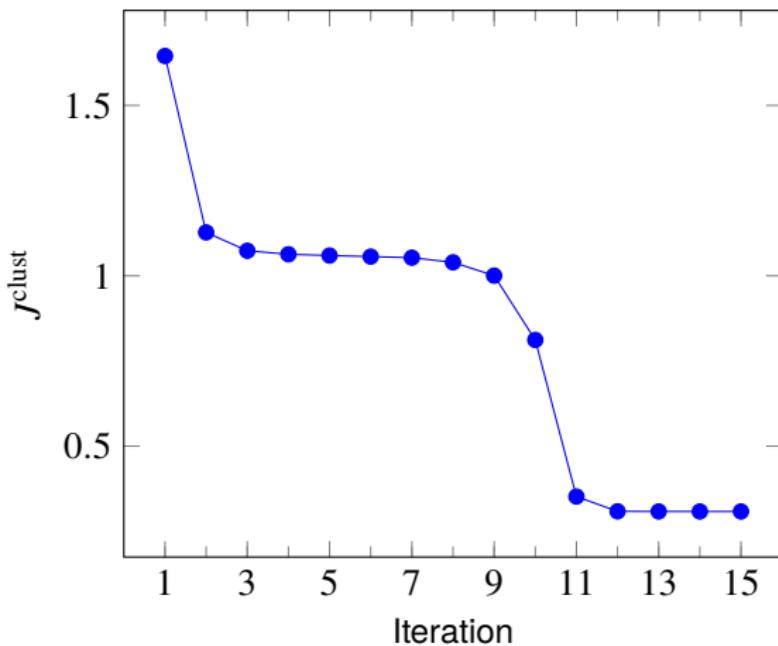
## Iteration 10



## Final clustering



## Convergence



# Outline

Clustering

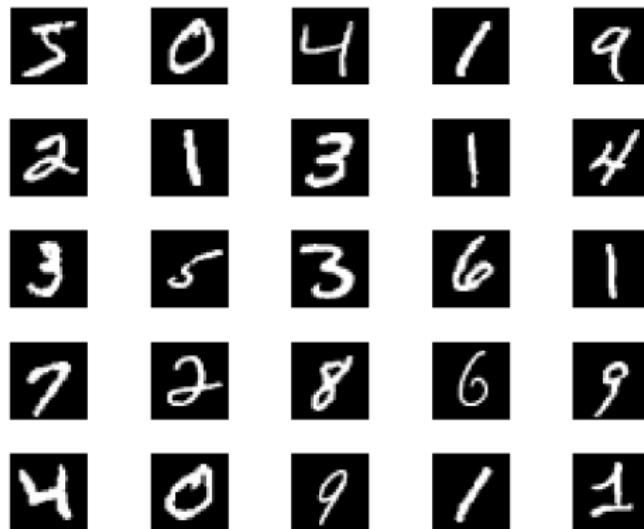
Algorithm

Examples

Applications

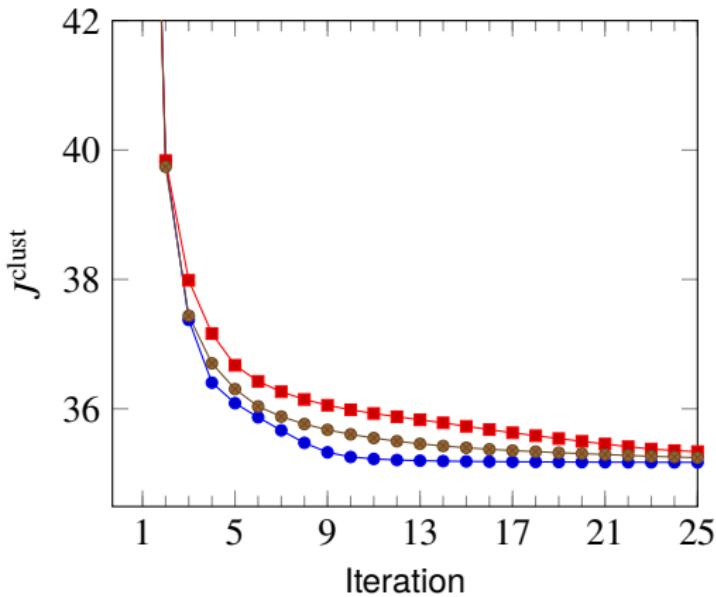
## Handwritten digit image set

- ▶ MNIST images of handwritten digits (via Yann Lecun)
- ▶  $N = 60,000$   $28 \times 28$  images, represented as 784-vectors  $x_i$
- ▶ 25 examples shown below

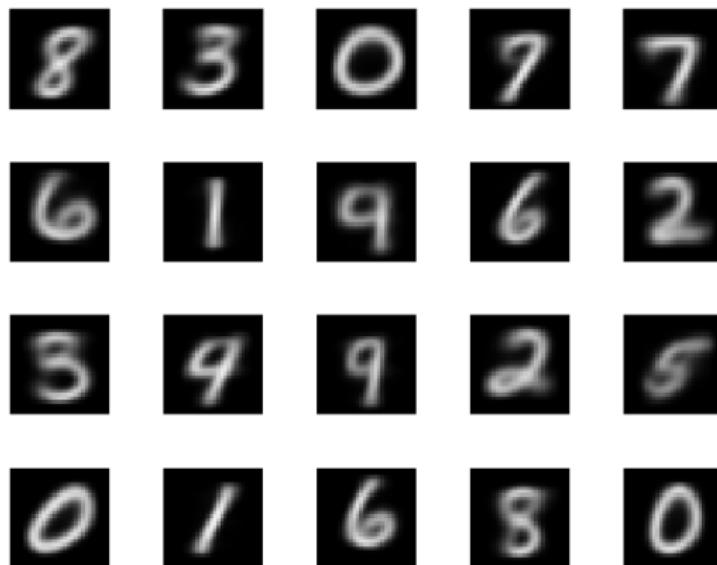


## *k*-means image clustering

- ▶  $k = 20$ , run 20 times with different initial assignments
- ▶ convergence shown below (including best and worst)

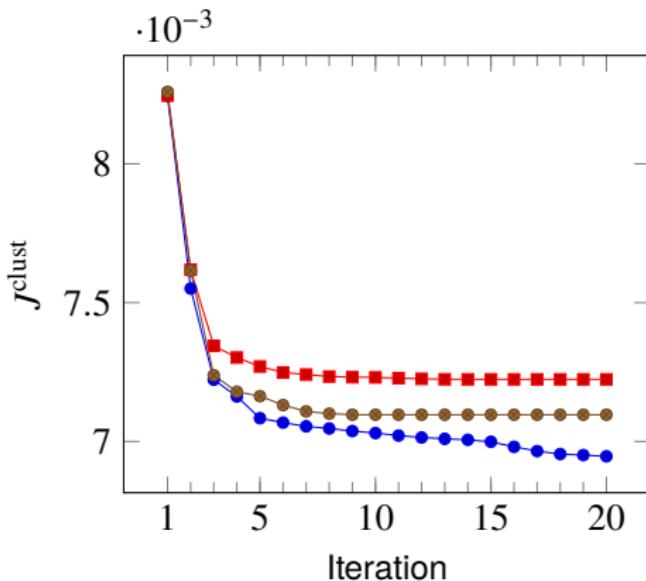


## Group representatives, best clustering



## Topic discovery

- ▶  $N = 500$  Wikipedia articles, word count histograms with  $n = 4423$
- ▶  $k = 9$ , run 20 times with different initial assignments
- ▶ convergence shown below (including best and worst)



## Topics discovered (clusters 1–3)

- ▶ words with largest representative coefficients

Cluster 1		Cluster 2		Cluster 3	
Word	Coef.	Word	Coef.	Word	Coef.
fight	0.038	holiday	0.012	united	0.004
win	0.022	celebrate	0.009	family	0.003
event	0.019	festival	0.007	party	0.003
champion	0.015	celebration	0.007	president	0.003
fighter	0.015	calendar	0.006	government	0.003

- ▶ titles of articles closest to cluster representative

1. “Floyd Mayweather, Jr”, “Kimbo Slice”, “Ronda Rousey”, “José Aldo”, “Joe Frazier”, “Wladimir Klitschko”, “Saul Álvarez”, “Gennady Golovkin”, “Nate Diaz”, ...
2. “Halloween”, “Guy Fawkes Night” “Diwali”, “Hanukkah”, “Groundhog Day”, “Rosh Hashanah”, “Yom Kippur”, “Seventh-day Adventist Church”, “Remembrance Day”, ...
3. “Mahatma Gandhi”, “Sigmund Freud”, “Carly Fiorina”, “Frederick Douglass”, “Marco Rubio”, “Christopher Columbus”, “Fidel Castro”, “Jim Webb”, ...

## Topics discovered (clusters 4–6)

- ▶ words with largest representative coefficients

Cluster 4		Cluster 5		Cluster 6	
Word	Coef.	Word	Coef.	Word	Coef.
album	0.031	game	0.023	series	0.029
release	0.016	season	0.020	season	0.027
song	0.015	team	0.018	episode	0.013
music	0.014	win	0.017	character	0.011
single	0.011	player	0.014	film	0.008

- ▶ titles of articles closest to cluster representative

1. “David Bowie”, “Kanye West” “Celine Dion”, “Kesha”, “Ariana Grande”, “Adele”, “Gwen Stefani”, “Anti (album)”, “Dolly Parton”, “Sia Furler”, ...
2. “Kobe Bryant”, “Lamar Odom”, “Johan Cruyff”, “Yogi Berra”, “José Mourinho”, “Halo 5: Guardians”, “Tom Brady”, “Eli Manning”, “Stephen Curry”, “Carolina Panthers”, ...
3. “The X-Files”, “Game of Thrones”, “House of Cards (U.S. TV series)”, “Daredevil (TV series)”, “Supergirl (U.S. TV series)”, “American Horror Story”, ...

## Topics discovered (clusters 7–9)

- ▶ words with largest representative coefficients

Cluster 7		Cluster 8		Cluster 9	
Word	Coef.	Word	Coef.	Word	Coef.
match	0.065	film	0.036	film	0.061
win	0.018	star	0.014	million	0.019
championship	0.016	role	0.014	release	0.013
team	0.015	play	0.010	star	0.010
event	0.015	series	0.009	character	0.006

- ▶ titles of articles closest to cluster representative

1. “Wrestlemania 32”, “Payback (2016)”, “Survivor Series (2015)”, “Royal Rumble (2016)”, “Night of Champions (2015)”, “Fastlane (2016)”, “Extreme Rules (2016)”, ...
2. “Ben Affleck”, “Johnny Depp”, “Maureen O’Hara”, “Kate Beckinsale”, “Leonardo DiCaprio”, “Keanu Reeves”, “Charlie Sheen”, “Kate Winslet”, “Carrie Fisher”, ...
3. “Star Wars: The Force Awakens”, “Star Wars Episode I: The Phantom Menace”, “The Martian (film)”, “The Revenant (2015 film)”, “The Hateful Eight”, ...

## 5. Linear independence

# Outline

Linear independence

Basis

Orthonormal vectors

Gram–Schmidt algorithm

## Linear dependence

- ▶ set of  $n$ -vectors  $\{a_1, \dots, a_k\}$  (with  $k \geq 1$ ) is *linearly dependent* if

$$\beta_1 a_1 + \cdots + \beta_k a_k = 0$$

holds for some  $\beta_1, \dots, \beta_k$ , that are not all zero

- ▶ equivalent to: at least one  $a_i$  is a linear combination of the others
- ▶ we say ' $a_1, \dots, a_k$  are linearly dependent'
- ▶  $\{a_1\}$  is linearly dependent only if  $a_1 = 0$
- ▶  $\{a_1, a_2\}$  is linearly dependent only if one  $a_i$  is a multiple of the other
- ▶ for more than two vectors, there is no simple to state condition

## 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, since  $a_1 + 2a_2 - 3a_3 = 0$

- ▶ can express any of them as linear combination of the other two, e.g.,

$$a_2 = (-1/2)a_1 + (3/2)a_3$$

## Linear independence

- ▶ set of  $n$ -vectors  $\{a_1, \dots, a_k\}$  (with  $k \geq 1$ ) is *linearly independent* if it is not linearly dependent, i.e.,

$$\beta_1 a_1 + \cdots + \beta_k a_k = 0$$

holds only when  $\beta_1 = \cdots = \beta_k = 0$

- ▶ we say ' $a_1, \dots, a_k$  are linearly independent'
- ▶ equivalent to: no  $a_i$  is a linear combination of the others
- ▶ example: the unit  $n$ -vectors  $e_1, \dots, e_n$  are linearly independent

## Linear combinations of linearly independent vectors

- ▶ suppose  $x$  is linear combination of linearly independent vectors  $a_1, \dots, a_k$ :

$$x = \beta_1 a_1 + \cdots + \beta_k a_k$$

- ▶ the coefficients  $\beta_1, \dots, \beta_k$  are *unique*, i.e., if

$$x = \gamma_1 a_1 + \cdots + \gamma_k a_k$$

then  $\beta_i = \gamma_i$  for  $i = 1, \dots, k$

- ▶ this means that (in principle) we can deduce the coefficients from  $x$
- ▶ to see why, note that

$$(\beta_1 - \gamma_1)a_1 + \cdots + (\beta_k - \gamma_k)a_k = 0$$

and so (by linear independence)  $\beta_1 - \gamma_1 = \cdots = \beta_k - \gamma_k = 0$

# Outline

Linear independence

Basis

Orthonormal vectors

Gram–Schmidt algorithm

## Independence-dimension inequality

- ▶ *a linearly independent set of  $n$ -vectors can have at most  $n$  elements*
- ▶ put another way: *any set of  $n + 1$  or more  $n$ -vectors is linearly dependent*

## Basis

- ▶ a set of  $n$  linearly independent  $n$ -vectors  $a_1, \dots, a_n$  is called a *basis*
- ▶ any  $n$ -vector  $b$  can be expressed as a linear combination of them:

$$b = \beta_1 a_1 + \cdots + \beta_n a_n$$

for some  $\beta_1, \dots, \beta_n$

- ▶ and these coefficients are unique
- ▶ formula above is called *expansion of  $b$  in the  $a_1, \dots, a_n$  basis*
- ▶ example:  $e_1, \dots, e_n$  is a basis, expansion of  $b$  is

$$b = b_1 e_1 + \cdots + b_n e_n$$

# Outline

Linear independence

Basis

Orthonormal vectors

Gram–Schmidt algorithm

## Orthonormal vectors

- ▶ set of  $n$ -vectors  $a_1, \dots, a_k$  are (*mutually*) *orthogonal* if  $a_i \perp a_j$  for  $i \neq j$
- ▶ they are *normalized* if  $\|a_i\| = 1$  for  $i = 1, \dots, k$
- ▶ they are *orthonormal* if both hold
- ▶ can be expressed using inner products as

$$a_i^T a_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

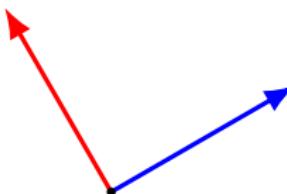
- ▶ orthonormal sets of vectors are linearly independent
- ▶ by independence-dimension inequality, must have  $k \leq n$
- ▶ when  $k = n$ ,  $a_1, \dots, a_n$  are an *orthonormal basis*

## Examples of orthonormal bases

- ▶ standard unit  $n$ -vectors  $e_1, \dots, e_n$
- ▶ the 3-vectors

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

- ▶ the 2-vectors shown below



## Orthonormal expansion

- ▶ if  $a_1, \dots, a_n$  is an orthonormal basis, we have for any  $n$ -vector  $x$

$$x = (a_1^T x) a_1 + \cdots + (a_n^T x) a_n$$

- ▶ called *orthonormal expansion of  $x$*  (in the orthonormal basis)
- ▶ to verify formula, take inner product of both sides with  $a_i$

# Outline

Linear independence

Basis

Orthonormal vectors

Gram–Schmidt algorithm

## Gram–Schmidt (orthogonalization) algorithm

- ▶ an algorithm to check if  $a_1, \dots, a_k$  are linearly independent
- ▶ we'll see later it has many other uses

## Gram–Schmidt algorithm

---

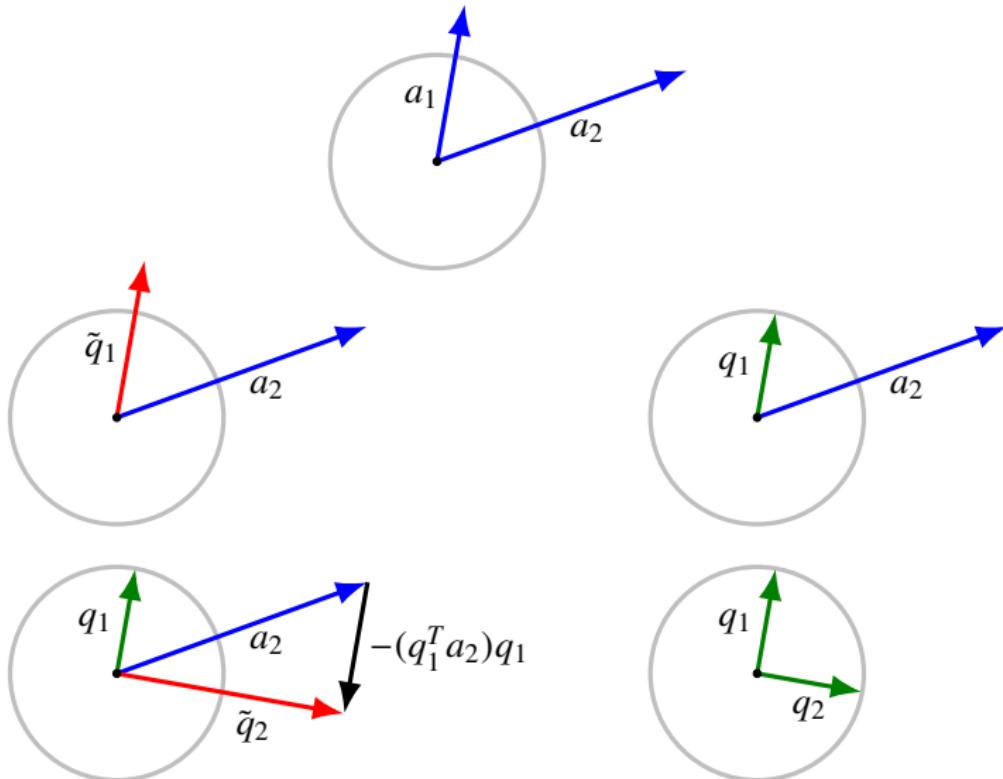
**given**  $n$ -vectors  $a_1, \dots, a_k$

**for**  $i = 1, \dots, k$

1. *Orthogonalization:*  $\tilde{q}_i = a_i - (q_1^T a_i) q_1 - \cdots - (q_{i-1}^T a_i) q_{i-1}$
  2. *Test for linear dependence:* if  $\tilde{q}_i = 0$ , quit
  3. *Normalization:*  $q_i = \tilde{q}_i / \|\tilde{q}_i\|$
- 

- ▶ if G–S does not stop early (in step 2),  $a_1, \dots, a_k$  are linearly independent
- ▶ if G–S stops early in iteration  $i = j$ , then  $a_j$  is a linear combination of  $a_1, \dots, a_{j-1}$  (so  $a_1, \dots, a_k$  are linearly dependent)

## Example



## Analysis

let's show by induction that  $q_1, \dots, q_i$  are orthonormal

- ▶ assume it's true for  $i - 1$
- ▶ orthogonalization step ensures that

$$\tilde{q}_i \perp q_1, \dots, \tilde{q}_i \perp q_{i-1}$$

- ▶ to see this, take inner product of both sides with  $q_j, j < i$

$$\begin{aligned} q_j^T \tilde{q}_i &= q_j^T a_i - (q_1^T a_i)(q_j^T q_1) - \cdots - (q_{i-1}^T a_i)(q_j^T q_{i-1}) \\ &= q_j^T a_i - q_j^T a_i = 0 \end{aligned}$$

- ▶ so  $q_i \perp q_1, \dots, q_i \perp q_{i-1}$
- ▶ normalization step ensures that  $\|q_i\| = 1$

## Analysis

assuming G–S has not terminated before iteration  $i$

- ▶  $a_i$  is a linear combination of  $q_1, \dots, q_i$ :

$$a_i = \|\tilde{q}_i\|q_i + (q_1^T a_i)q_1 + \cdots + (q_{i-1}^T a_i)q_{i-1}$$

- ▶  $q_i$  is a linear combination of  $a_1, \dots, a_i$ : by induction on  $i$ ,

$$q_i = (1/\|\tilde{q}_i\|) \left( a_i - (q_1^T a_i)q_1 - \cdots - (q_{i-1}^T a_i)q_{i-1} \right)$$

and (by induction assumption) each  $q_1, \dots, q_{i-1}$  is a linear combination of  $a_1, \dots, a_{i-1}$

## Early termination

suppose G–S terminates in step  $j$

- ▶  $a_j$  is linear combination of  $q_1, \dots, q_{j-1}$

$$a_j = (q_1^T a_j)q_1 + \cdots + (q_{j-1}^T a_j)q_{j-1}$$

- ▶ and each of  $q_1, \dots, q_{j-1}$  is linear combination of  $a_1, \dots, a_{j-1}$
- ▶ so  $a_j$  is a linear combination of  $a_1, \dots, a_{j-1}$

## Complexity of Gram–Schmidt algorithm

- ▶ step 1 of iteration  $i$  requires  $i - 1$  inner products,

$$q_1^T a_i, \dots, q_{i-1}^T a_i$$

which costs  $(i - 1)(2n - 1)$  flops

- ▶  $n(i - 1)$  flops to compute  $\tilde{q}_i$
- ▶  $3n$  flops to compute  $\|\tilde{q}_i\|$  and  $q_i$
- ▶ total is

$$\sum_{i=1}^k ((4n - 1)(i - 1) + 3n) = (4n - 1) \frac{k(k - 1)}{2} + 3nk \approx 2nk^2$$

using  $\sum_{i=1}^k (i - 1) = k(k - 1)/2$

## 6. Matrices

# Outline

Matrices

Matrix-vector multiplication

Examples

## Matrices

- ▶ a *matrix* is a rectangular array of numbers, e.g.,

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

- ▶ its *size* is given by (row dimension)  $\times$  (column dimension)  
e.g., matrix above is  $3 \times 4$
- ▶ *elements* also called *entries* or *coefficients*
- ▶  $B_{ij}$  is  $i,j$  element of matrix  $B$
- ▶  $i$  is the *row index*,  $j$  is the *column index*; indexes start at 1
- ▶ two matrices are *equal* (denoted with  $=$ ) if they are the same size and corresponding entries are equal

## Matrix shapes

an  $m \times n$  matrix  $A$  is

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

## Column and row vectors

- ▶ we consider an  $n \times 1$  matrix to be an  $n$ -vector
- ▶ we consider a  $1 \times 1$  matrix to be a number
- ▶ a  $1 \times n$  matrix is called a *row vector*, e.g.,

$$\begin{bmatrix} 1.2 & -0.3 & 1.4 & 2.6 \end{bmatrix}$$

which is *not* the same as the (column) vector

$$\begin{bmatrix} 1.2 \\ -0.3 \\ 1.4 \\ 2.6 \end{bmatrix}$$

## Columns and rows of a matrix

- ▶ suppose  $A$  is an  $m \times n$  matrix with entries  $A_{ij}$  for  $i = 1, \dots, m, j = 1, \dots, n$
- ▶ its  $j$ th *column* is (the  $m$ -vector)

$$\begin{bmatrix} A_{1j} \\ \vdots \\ A_{mj} \end{bmatrix}$$

- ▶ its  $i$ th *row* is (the  $n$ -row-vector)

$$\begin{bmatrix} A_{i1} & \cdots & A_{in} \end{bmatrix}$$

- ▶ *slice* of matrix:  $A_{p:q,r:s}$  is the  $(q - p + 1) \times (s - r + 1)$  matrix

$$A_{p:q,r:s} = \begin{bmatrix} A_{pr} & A_{p,r+1} & \cdots & A_{ps} \\ A_{p+1,r} & A_{p+1,r+1} & \cdots & A_{p+1,s} \\ \vdots & \vdots & & \vdots \\ A_{qr} & A_{q,r+1} & \cdots & A_{qs} \end{bmatrix}$$

## Block matrices

- ▶ we can form *block matrices*, whose entries are matrices, such as

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

where  $B$ ,  $C$ ,  $D$ , and  $E$  are matrices (called *submatrices* or *blocks* of  $A$ )

- ▶ matrices in each block row must have same height (row dimension)
- ▶ matrices in each block column must have same width (column dimension)
- ▶ example: if

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

then

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

## Column and row representation of matrix

- ▶  $A$  is an  $m \times n$  matrix
- ▶ can express as block matrix with its ( $m$ -vector) columns  $a_1, \dots, a_n$

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

- ▶ or as block matrix with its ( $n$ -row-vector) rows  $b_1, \dots, b_m$

$$A = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

## Examples

- ▶ *image*:  $X_{ij}$  is  $i,j$  pixel value in a monochrome image
- ▶ *rainfall data*:  $A_{ij}$  is rainfall at location  $i$  on day  $j$
- ▶ *multiple asset returns*:  $R_{ij}$  is return of asset  $j$  in period  $i$
- ▶ *contingency table*:  $A_{ij}$  is number of objects with first attribute  $i$  and second attribute  $j$
- ▶ *feature matrix*:  $X_{ij}$  is value of feature  $i$  for entity  $j$

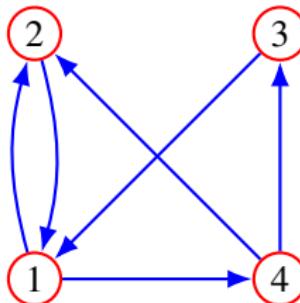
in each of these, what do the rows and columns mean?

## Graph or relation

- ▶ a *relation* is a set of pairs of *objects*, labeled  $1, \dots, n$ , such as

$$\mathcal{R} = \{(1,2), (1,3), (2,1), (2,4), (3,4), (4,1)\}$$

- ▶ same as *directed graph*



- ▶ can be represented as  $n \times n$  matrix with  $A_{ij} = 1$  if  $(i,j) \in \mathcal{R}$

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

## Special matrices

- ▶  $m \times n$  zero matrix has all entries zero, written as  $0_{m \times n}$  or just 0
- ▶ identity matrix is square matrix with  $I_{ii} = 1$  and  $I_{ij} = 0$  for  $i \neq j$ , e.g.,

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

- ▶ sparse matrix: most entries are zero
  - examples: 0 and  $I$
  - can be stored and manipulated efficiently
  - $\text{nnz}(A)$  is number of nonzero entries

## Diagonal and triangular matrices

- ▶ *diagonal matrix*: square matrix with  $A_{ij} = 0$  when  $i \neq j$
- ▶  $\text{diag}(a_1, \dots, a_n)$  denotes the diagonal matrix with  $A_{ii} = a_i$  for  $i = 1, \dots, n$
- ▶ example:

$$\text{diag}(0.2, -3, 1.2) = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 1.2 \end{bmatrix}$$

- ▶ *lower triangular matrix*:  $A_{ij} = 0$  for  $i < j$
- ▶ *upper triangular matrix*:  $A_{ij} = 0$  for  $i > j$
- ▶ examples:

$$\begin{bmatrix} 1 & -1 & 0.7 \\ 0 & 1.2 & -1.1 \\ 0 & 0 & 3.2 \end{bmatrix} \text{ (upper triangular),} \quad \begin{bmatrix} -0.6 & 0 \\ -0.3 & 3.5 \end{bmatrix} \text{ (lower triangular)}$$

## Transpose

- ▶ the *transpose* of an  $m \times n$  matrix  $A$  is denoted  $A^T$ , and defined by

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

- ▶ for example,

$$\begin{bmatrix} 0 & 4 \\ 7 & 0 \\ 3 & 1 \end{bmatrix}^T = \begin{bmatrix} 0 & 7 & 3 \\ 4 & 0 & 1 \end{bmatrix}$$

- ▶ transpose converts column to row vectors (and vice versa)
- ▶  $(A^T)^T = A$

## Addition, subtraction, and scalar multiplication

- ▶ (just like vectors) we can add or subtract matrices of the same size:

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

(subtraction is similar)

- ▶ scalar multiplication:

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

- ▶ many obvious properties, e.g.,

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

## Matrix norm

- ▶ for  $m \times n$  matrix  $A$ , we define

$$\|A\| = \left( \sum_{i=1}^m \sum_{j=1}^n A_{ij}^2 \right)^{1/2}$$

- ▶ agrees with vector norm when  $n = 1$
- ▶ satisfies norm properties:

$$\|\alpha A\| = |\alpha| \|A\|$$

$$\|A + B\| \leq \|A\| + \|B\|$$

$$\|A\| \geq 0$$

$$\|A\| = 0 \text{ only if } A = 0$$

- ▶ distance between two matrices:  $\|A - B\|$
- ▶ (there are other matrix norms, which we won't use)

# Outline

Matrices

Matrix-vector multiplication

Examples

## Matrix-vector product

- *matrix-vector product* of  $m \times n$  matrix  $A$ ,  $n$ -vector  $x$ , denoted  $y = Ax$ , with

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

- for example,

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

## Row interpretation

- ▶  $y = Ax$  can be expressed as

$$y_i = b_i^T x, \quad i = 1, \dots, m$$

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

- ▶ so  $y = Ax$  is a ‘batch’ inner product of all rows of  $A$  with  $x$
- ▶ example:  $A\mathbf{1}$  is vector of row sums of matrix  $A$

## Column interpretation

- ▶  $y = Ax$  can be expressed as

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

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

- ▶ so  $y = Ax$  is linear combination of columns of  $A$ , with coefficients  $x_1, \dots, x_n$
- ▶ important example:  $Ae_j = a_j$
- ▶ columns of  $A$  are linearly independent if  $Ax = 0$  implies  $x = 0$

# Outline

Matrices

Matrix-vector multiplication

Examples

## General examples

- ▶  $0x = 0$ , i.e., multiplying by zero matrix gives zero
- ▶  $Ix = x$ , i.e., multiplying by identity matrix does nothing
- ▶ inner product  $a^T b$  is matrix-vector product of  $1 \times n$  matrix  $a^T$  and  $n$ -vector  $b$
- ▶  $\tilde{x} = Ax$  is de-meaned version of  $x$ , with

$$A = \begin{bmatrix} 1 - 1/n & -1/n & \cdots & -1/n \\ -1/n & 1 - 1/n & \cdots & -1/n \\ \vdots & & \ddots & \vdots \\ -1/n & -1/n & \cdots & 1 - 1/n \end{bmatrix}$$

## Difference matrix

- $(n - 1) \times n$  difference matrix is

$$D = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 & 0 \\ \ddots & \ddots & \ddots & & & & \\ & \ddots & \ddots & \ddots & & & \\ 0 & 0 & 0 & \cdots & -1 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 1 \end{bmatrix}$$

$y = Dx$  is  $(n - 1)$ -vector of differences of consecutive entries of  $x$ :

$$Dx = \begin{bmatrix} x_2 - x_1 \\ x_3 - x_2 \\ \vdots \\ x_n - x_{n-1} \end{bmatrix}$$

- *Dirichlet energy:*  $\|Dx\|^2$  is measure of wiggliness for  $x$  a time series

## Return matrix – portfolio vector

- ▶  $R$  is  $T \times n$  matrix of asset returns
- ▶  $R_{ij}$  is return of asset  $j$  in period  $i$  (say, in percentage)
- ▶  $n$ -vector  $w$  gives portfolio (investments in the assets)
- ▶  $T$ -vector  $Rw$  is time series of the portfolio return
- ▶ **avg**( $Rw$ ) is the portfolio (mean) return, **std**( $Rw$ ) is its risk

## Feature matrix – weight vector

- ▶  $X = [x_1 \ \cdots \ x_N]$  is  $n \times N$  *feature matrix*
- ▶ column  $x_j$  is feature  $n$ -vector for object or example  $j$
- ▶  $X_{ij}$  is value of feature  $i$  for example  $j$
- ▶  $n$ -vector  $w$  is weight vector
- ▶  $s = X^T w$  is vector of scores for each example;  $s_j = x_j^T w$

## Input – output matrix

- ▶  $A$  is  $m \times n$  matrix
- ▶  $y = Ax$
- ▶  $n$ -vector  $x$  is *input* or *action*
- ▶  $m$ -vector  $y$  is *output* or *result*
- ▶  $A_{ij}$  is the factor by which  $y_i$  depends on  $x_j$
- ▶  $A_{ij}$  is the *gain* from input  $j$  to output  $i$
- ▶ e.g., if  $A$  is lower triangular, then  $y_i$  only depends on  $x_1, \dots, x_i$

# Complexity

- ▶  $m \times n$  matrix stored  $A$  as  $m \times n$  array of numbers  
(for sparse  $A$ , store only  $\text{nnz}(A)$  nonzero values)
- ▶ matrix addition, scalar-matrix multiplication cost  $mn$  flops
- ▶ matrix-vector multiplication costs  $m(2n - 1) \approx 2mn$  flops  
(for sparse  $A$ , around  $2\text{nnz}(A)$  flops)

## 7. Matrix examples

# Outline

Geometric transformations

Selectors

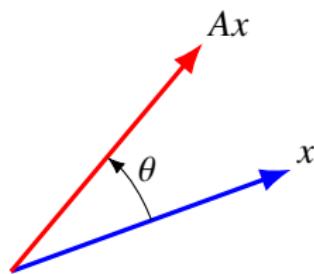
Incidence matrix

Convolution

## Geometric transformations

- ▶ many geometric transformations and mappings of 2-D and 3-D vectors can be represented via matrix multiplication  $y = Ax$
- ▶ for example, rotation by  $\theta$ :

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



(to get the entries, look at  $Ae_1$  and  $Ae_2$ )

# Outline

Geometric transformations

Selectors

Incidence matrix

Convolution

## Selectors

- ▶ an  $m \times n$  *selector matrix*: each row is a unit vector (transposed)

$$A = \begin{bmatrix} e_{k_1}^T \\ \vdots \\ e_{k_m}^T \end{bmatrix}$$

- ▶ multiplying by  $A$  selects entries of  $x$ :

$$Ax = (x_{k_1}, x_{k_2}, \dots, x_{k_m})$$

- ▶ example: the  $m \times 2m$  matrix

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

'down-samples' by 2: if  $x$  is a  $2m$ -vector then  $y = Ax = (x_1, x_3, \dots, x_{2m-1})$

- ▶ other examples: image cropping, permutation, ...

# Outline

Geometric transformations

Selectors

Incidence matrix

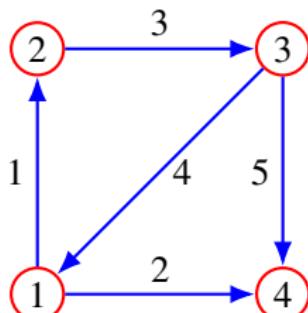
Convolution

## Incidence matrix

- ▶ graph with  $n$  vertices or nodes,  $m$  (directed) edges or links
- ▶ incidence matrix is  $n \times m$  matrix

$$A_{ij} = \begin{cases} 1 & \text{edge } j \text{ points to node } i \\ -1 & \text{edge } j \text{ points from node } i \\ 0 & \text{otherwise} \end{cases}$$

- ▶ example with  $n = 4, m = 5$ :



This is a node to edge graph

$$A \stackrel{\textcolor{orange}{1}}{=} \left[ \begin{array}{ccccc} -1 & -1 & 0 & 1 & 0 \\ \textcolor{orange}{1} & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 \\ \textcolor{orange}{4} & 0 & 1 & 0 & 0 \end{array} \right]$$

## Flow conservation

- ▶  $m$ -vector  $x$  gives flows (of something) along the edges
- ▶ examples: heat, money, power, mass, people, ...
- ▶  $x_j > 0$  means flow follows edge direction
- ▶  $Ax$  is  $n$ -vector that gives the total or net flows
- ▶  $(Ax)_i$  is the net flow into node  $i$
- ▶  $Ax = 0$  is *flow conservation*;  $x$  is called a *circulation*

## Potentials and Dirichlet energy

- ▶ suppose  $v$  is an  $n$ -vector, called a *potential*
- ▶  $v_i$  is potential value at node  $i$   $\mathbf{vT}=[v_1 \ v_2 \ v_3 \ v_4]$
- ▶  $u = A^T v$  is an  $m$ -vector of *potential differences* across the  $m$  edges  $(u_3=v_3-v_2)$
- ▶  $u_j = v_l - v_k$ , where edge  $j$  goes from  $k$  to node  $l$   $2 \dashrightarrow 3$
- ▶ *Dirichlet energy* is  $\mathcal{D}(v) = \|A^T v\|^2$ ,

$$\mathcal{D}(v) = \sum_{\text{edges } (k,l)} (v_l - v_k)^2$$

(sum of squares of potential differences across the edges)

- ▶  $\mathcal{D}(v)$  is small when potential values of neighboring nodes are similar

# Outline

Geometric transformations

Selectors

Incidence matrix

Convolution

## Convolution

- ▶ for  $n$ -vector  $a$ ,  $m$ -vector  $b$ , the *convolution*  $c = a * b$  is the  $(n + m - 1)$ -vector

$$c_k = \sum_{i+j=k+1} a_i b_j, \quad k = 1, \dots, n+m-1$$

- ▶ for example with  $n = 4$ ,  $m = 3$ , we have

$$\begin{aligned} c_1 &= a_1 b_1 \\ c_2 &= a_1 b_2 + a_2 b_1 \\ c_3 &= a_1 b_3 + a_2 b_2 + a_3 b_1 \\ c_4 &= a_2 b_3 + a_3 b_2 + a_4 b_1 \\ c_5 &= a_3 b_3 + a_4 b_2 \\ c_6 &= a_4 b_3 \end{aligned}$$

- ▶ example:  $(1, 0, -1) * (2, 1, -1) = (2, 1, -3, -1, 1)$

## Polynomial multiplication

- ▶  $a$  and  $b$  are coefficients of two polynomials:

$$p(x) = a_1 + a_2x + \cdots + a_nx^{n-1}, \quad q(x) = b_1 + b_2x + \cdots + b_mx^{m-1}$$

- ▶ convolution  $c = a * b$  gives the coefficients of the product  $p(x)q(x)$ :

$$p(x)q(x) = c_1 + c_2x + \cdots + c_{n+m-1}x^{n+m-2}$$

- ▶ this gives simple proofs of many properties of convolution; for example,

$$a * b = b * a$$

$$(a * b) * c = a * (b * c)$$

$$a * b = 0 \text{ only if } a = 0 \text{ or } b = 0$$

## Toeplitz matrices

- ▶ function  $f(b) = a * b$  is linear; in fact  $c = T(b)a$  with

$$T(b) = \begin{bmatrix} b_1 & 0 & 0 & 0 \\ b_2 & b_1 & 0 & 0 \\ b_3 & b_2 & b_1 & 0 \\ 0 & b_3 & b_2 & b_1 \\ 0 & 0 & b_3 & b_2 \\ 0 & 0 & 0 & b_3 \end{bmatrix}$$

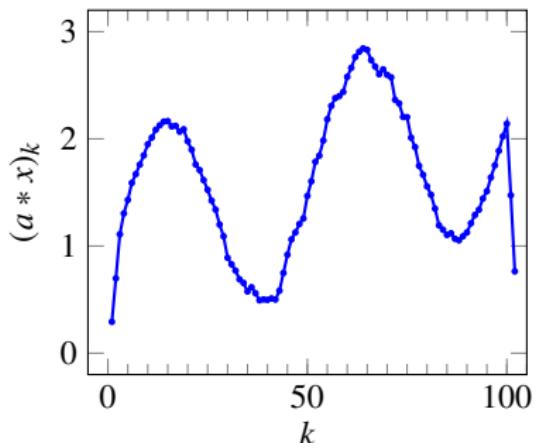
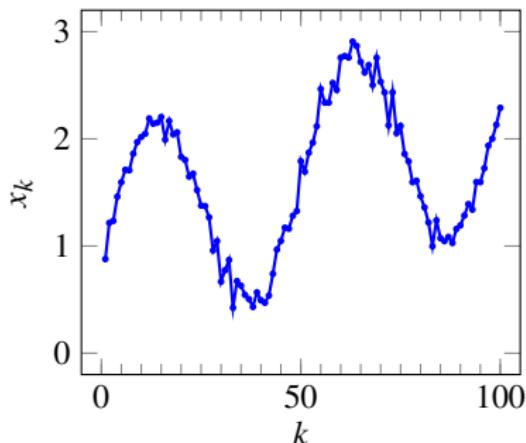
- ▶  $T(b)$  is a Toeplitz matrix (values on diagonals are equal)

## Moving average of time series

- ▶  $n$ -vector  $x$  represents a time series
- ▶ convolution  $y = a * x$  with  $a = (1/3, 1/3, 1/3)$  is 3-period moving average:

$$y_k = \frac{1}{3}(x_k + x_{k-1} + x_{k-2}), \quad k = 1, 2, \dots, n+2$$

(with  $x_k$  interpreted as zero for  $k < 1$  and  $k > n$ )



## Input-output convolution system

- ▶  $m$ -vector  $u$  represents a time series *input*
- ▶  $m + n - 1$  vector  $y$  represents a time series *output*
- ▶  $y = h * u$  is a *convolution model*
- ▶  $n$ -vector  $h$  is called the *system impulse response*
- ▶ we have

$$y_i = \sum_{j=1}^n u_{i-j+1} h_j$$

(interpreting  $u_k$  as zero for  $k < n$  or  $k > n$ )

- ▶ interpretation:  $y_i$ , output at time  $i$  is a linear combination of  $u_i, \dots, u_{i-n+1}$
- ▶  $h_3$  is the factor by which current output depends on what the input was 2 time steps before

## 8. Linear equations

# Outline

Linear functions

Linear function models

Linear equations

Balancing chemical equations

## Superposition

- ▶  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  means  $f$  is a function that maps  $n$ -vectors to  $m$ -vectors
- ▶ we write  $f(x) = (f_1(x), \dots, f_m(x))$  to emphasize components of  $f(x)$
- ▶ we write  $f(x) = f(x_1, \dots, x_n)$  to emphasize components of  $x$
- ▶  $f$  satisfies *superposition* if for all  $x, y, \alpha, \beta$

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

(this innocent looking equation says a lot ...)

- ▶ such an  $f$  is called *linear*

## Matrix-vector product function

- with  $A$  an  $m \times n$  matrix, define  $f$  as  $f(x) = Ax$
- $f$  is linear:

$$\begin{aligned}f(ax + \beta y) &= A(ax + \beta y) \\&= A(ax) + A(\beta y) \\&= \alpha(Ax) + \beta(Ay) \\&= \alpha f(x) + \beta f(y)\end{aligned}$$

- converse is true: if  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  is linear, then

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

with  $A = \left[ \begin{array}{cccc} f(e_1) & f(e_2) & \cdots & f(e_n) \end{array} \right]$

## Examples

- reversal:  $f(x) = (x_n, x_{n-1}, \dots, x_1)$

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

- running sum:  $f(x) = (x_1, x_1 + x_2, x_1 + x_2 + x_3, \dots, x_1 + x_2 + \dots + x_n)$

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

## Affine functions

- ▶ function  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  is *affine* if it is a linear function plus a constant, i.e.,

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

- ▶ same as:

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

holds for all  $x, y$ , and  $\alpha, \beta$  with  $\alpha + \beta = 1$

- ▶ can recover  $A$  and  $b$  from  $f$  using

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

- ▶ affine functions sometimes (incorrectly) called linear

# Outline

Linear functions

Linear function models

Linear equations

Balancing chemical equations

## Linear and affine functions models

- ▶ in many applications, relations between  $n$ -vectors and  $m$  vectors are *approximated* as linear or affine
- ▶ sometimes the approximation is excellent, and holds over large ranges of the variables (e.g., electromagnetics)
- ▶ sometimes the approximation is reasonably good over smaller ranges (e.g., aircraft dynamics)
- ▶ in other cases it is quite approximate, but still useful (e.g., econometric models)

## Price elasticity of demand

- ▶  $n$  goods or services
- ▶ prices given by  $n$ -vector  $p$ , demand given as  $n$ -vector  $d$
- ▶  $\delta_i^{\text{price}} = (p_i^{\text{new}} - p_i)/p_i$  is fractional changes in prices
- ▶  $\delta_i^{\text{dem}} = (d_i^{\text{new}} - d_i)/d_i$  is fractional change in demands
- ▶ *price-demand elasticity model:*  $\delta^{\text{dem}} = E\delta^{\text{price}}$
  
- ▶ what do the following mean?

$$E_{11} = -0.3, \quad E_{12} = +0.1, \quad E_{23} = -0.05$$

## Taylor series approximation

- ▶ suppose  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$  is differentiable
- ▶ first order Taylor approximation  $\hat{f}$  of  $f$  near  $z$ :

$$\begin{aligned}\hat{f}_i(x) &= f_i(z) + \frac{\partial f_i}{\partial x_1}(z)(x_1 - z_1) + \cdots + \frac{\partial f_i}{\partial x_n}(z)(x_n - z_n) \\ &= f_i(z) + \nabla f_i(z)^T (x - z)\end{aligned}$$

- ▶ in compact notation:  $\hat{f}(x) = f(z) + Df(z)(x - z)$
- ▶  $Df(x)$  is the  $m \times n$  derivative or Jacobian matrix of  $f$  at  $z$

$$Df(z)_{ij} = \frac{\partial f_i}{\partial x_j}(z), \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

- ▶  $\hat{f}(x)$  is a very good approximation of  $f(x)$  for  $x$  near  $z$
- ▶  $\hat{f}(x)$  is an affine function of  $x$

## Regression model

- ▶ regression model:  $\hat{y} = x^T \beta + v$ 
  - $x$  is  $n$ -vector of features or regressors
  - $\beta$  is  $n$ -vector of model parameters;  $v$  is offset parameter
  - (scalar)  $\hat{y}$  is our prediction of  $y$
- ▶ now suppose we have  $N$  *examples* or *samples*  $x^{(1)}, \dots, x^{(N)}$ , and associated responses  $y^{(1)}, \dots, y^{(N)}$
- ▶ associated predictions are  $\hat{y}^{(i)} = (x^{(i)})^T \beta + v$
- ▶ write as  $\hat{y}^d = X^T \beta + v\mathbf{1}$ 
  - $X$  is feature matrix with columns  $x^{(1)}, \dots, x^{(N)}$
  - $y^d$  is  $N$ -vector of responses  $(y^{(1)}, \dots, y^{(N)})$
  - $\hat{y}^d$  is  $N$ -vector of predictions  $(\hat{y}^{(1)}, \dots, \hat{y}^{(N)})$
- ▶ *prediction error* (vector) is  $y^d - \hat{y}^d = y^d - X^T \beta - v\mathbf{1}$

# Outline

Linear functions

Linear function models

Linear equations

Balancing chemical equations

## Systems of linear equations

- ▶ set (or *system*) of  $m$  linear equations in  $n$  variables  $x_1, \dots, x_n$ :

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

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

$$\vdots$$

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

- ▶  $n$ -vector  $x$  is called the variable or unknowns
- ▶  $A_{ij}$  are the *coefficients*;  $A$  is the coefficient matrix
- ▶  $b$  is called the *right-hand side*
- ▶ can express very compactly as  $Ax = b$

## Systems of linear equations

- ▶ systems of linear equations classified as
  - under-determined if  $m < n$  ( $A$  wide)
  - square if  $m = n$  ( $A$  square)
  - over-determined if  $m > n$  ( $A$  tall)
- ▶  $x$  is called a *solution* if  $Ax = b$
- ▶ depending on  $A$  and  $b$ , there can be
  - no solution
  - one solution
  - many solutions
- ▶ we'll see how to solve linear equations later

# Outline

Linear functions

Linear function models

Linear equations

Balancing chemical equations

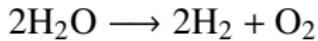
## Chemical equations

- ▶ a chemical reaction involves  $p$  reactants,  $q$  products (molecules)
- ▶ expressed as



- ▶  $R_1, \dots, R_p$  are reactants
- ▶  $P_1, \dots, P_q$  are products
- ▶  $a_1, \dots, a_p, b_1, \dots, b_q$  are positive coefficients
- ▶ coefficients usually integers, but can be scaled
  - e.g., multiplying all coefficients by 1/2 doesn't change the reaction

## Example: electrolysis of water



- ▶ one reactant: water ( $\text{H}_2\text{O}$ )
- ▶ two products: hydrogen ( $\text{H}_2$ ) and oxygen ( $\text{O}_2$ )
- ▶ reaction consumes 2 water molecules and produces 2 hydrogen molecules and 1 oxygen molecule

## Balancing equations

- ▶ each molecule (reactant/product) contains specific numbers of (types of) atoms, given in its formula
  - e.g.,  $\text{H}_2\text{O}$  contains two H and one O
- ▶ *conservation of mass*: total number of each type of atom in a chemical equation must *balance*
- ▶ for each atom, total number on LHS must equal total on RHS
- ▶ e.g., electrolysis reaction is balanced:
  - 4 units of H on LHS and RHS
  - 2 units of O on LHS and RHS
- ▶ finding (nonzero) coefficients to achieve balance is called *balancing* equations

## Reactant and product matrices

- ▶ consider reaction with  $m$  types of atoms,  $p$  reactants,  $q$  products
- ▶  $m \times p$  reactant matrix  $R$  is defined by

$$R_{ij} = \text{number of atoms of type } i \text{ in reactant } R_j,$$

for  $i = 1, \dots, m$  and  $j = 1, \dots, p$

- ▶ with  $a = (a_1, \dots, a_p)$  (vector of reactant coefficients)

$Ra$  = (vector of) total numbers of atoms of each type in reactants

- ▶ define product  $m \times q$  matrix  $P$  in similar way
- ▶  $m$ -vector  $Pb$  is total numbers of atoms of each type in products
- ▶ conservation of mass is  $Ra = Pb$

## Balancing equations via linear equations

- ▶ conservation of mass is

$$\begin{bmatrix} R & -P \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = 0$$

- ▶ simple solution is  $a = b = 0$
- ▶ to find a nonzero solution, set any coefficient (say,  $a_1$ ) to be 1
- ▶ balancing chemical equations can be expressed as solving a set of  $m + 1$  linear equations in  $p + q$  variables

$$\begin{bmatrix} R & -P \\ e_1^T & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = e_{m+1}$$

(we ignore here that  $a_i$  and  $b_i$  should be nonnegative integers)

## Conservation of charge

- ▶ can extend to include charge, e.g.,  $\text{Cr}_2\text{O}_7^{2-}$  has charge  $-2$
- ▶ *conservation of charge*: total charge on each side of reaction must balance
- ▶ we can simply treat charge as another type of atom to balance

## Example



- ▶ 5 atoms/charge: Cr, O, Fe, H, charge
- ▶ reactant and product matrix:

$$R = \begin{bmatrix} 2 & 0 & 0 \\ 7 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & 2 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \\ 3 & 3 & 0 \end{bmatrix}$$

- ▶ balancing equations (including  $a_1 = 1$  constraint)

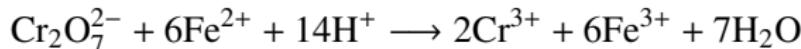
$$\begin{bmatrix} 2 & 0 & 0 & -1 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -2 \\ -2 & 2 & 1 & -3 & -3 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

## Balancing equations example

- ▶ solving the system yields

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 14 \\ 2 \\ 6 \\ 7 \end{bmatrix}$$

- ▶ the balanced equation is



## 9. Linear dynamical systems

# Outline

Linear dynamical systems

Population dynamics

Epidemic dynamics

## State sequence

- ▶ sequence of  $n$ -vectors  $x_1, x_2, \dots$
- ▶  $t$  denotes time or period
- ▶  $x_t$  is called *state* at time  $t$ ; sequence is called *state trajectory*
- ▶ assuming  $t$  is current time,
  - $x_t$  is current state
  - $x_{t-1}$  is previous state
  - $x_{t+1}$  is next state
- ▶ examples:  $x_t$  represents
  - age distribution in a population
  - economic output in  $n$  sectors
  - mechanical variables

## Linear dynamics

- ▶ linear dynamical system:

$$x_{t+1} = A_t x_t, \quad t = 1, 2, \dots$$

- ▶  $A_t$  are  $n \times n$  dynamics matrices
- ▶  $(A_t)_{ij}(x_t)_j$  is contribution to  $(x_{t+1})_i$  from  $(x_t)_j$
- ▶ system is called *time-invariant* if  $A_t = A$  doesn't depend on time
- ▶ can simulate evolution of  $x_t$  using recursion  $x_{t+1} = A_t x_t$

## Variations

- ▶ linear dynamical system with input

$$x_{t+1} = A_t x_t + B_t u_t + c_t, \quad t = 1, 2, \dots$$

- $u_t$  is an *input m-vector*
- $B_t$  is  $n \times m$  *input matrix*
- $c_t$  is *offset*

- ▶  $K$ -Markov model:

$$x_{t+1} = A_1 x_t + \cdots + A_K x_{t-K+1}, \quad t = K, K+1, \dots$$

- next state depends on current state and  $K - 1$  previous states
- also known as *auto-regressive model*
- for  $K = 1$ , this is the standard linear dynamical system  $x_{t+1} = Ax_t$

# Outline

Linear dynamical systems

Population dynamics

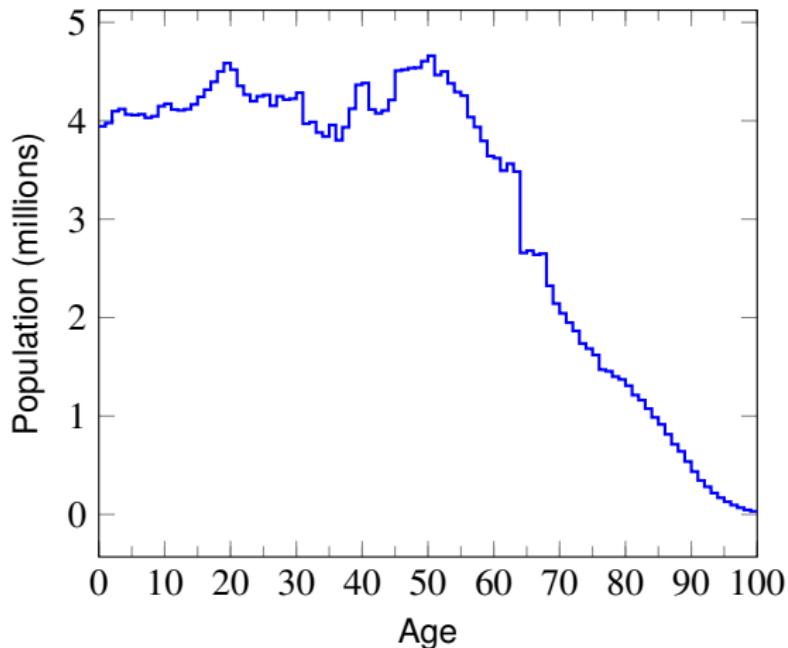
Epidemic dynamics

## Population distribution

- ▶  $x_t \in \mathbf{R}^{100}$  gives population distribution in year  $t = 1, \dots, T$
- ▶  $(x_t)_i$  is the number of people with age  $i - 1$  in year  $t$  (say, on January 1)
- ▶ total population in year  $t$  is  $\mathbf{1}^T x_t$
- ▶ number of people age 70 or older in year  $t$  is  $(0_{70}, \mathbf{1}_{30})^T x_t$

## Population distribution of the U.S.

(from 2010 census)

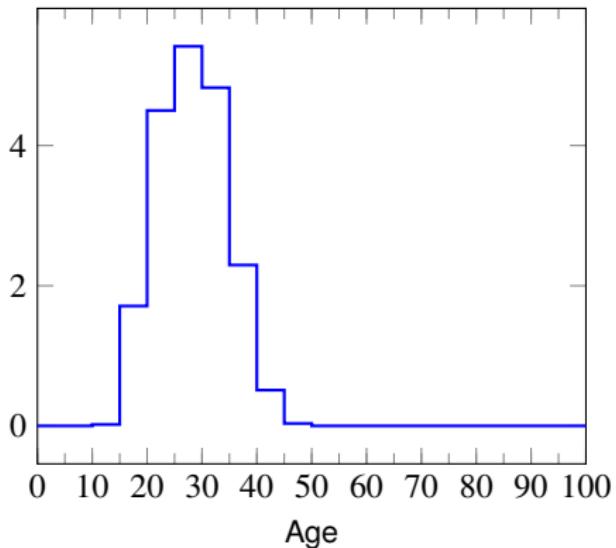


## Birth and death rates

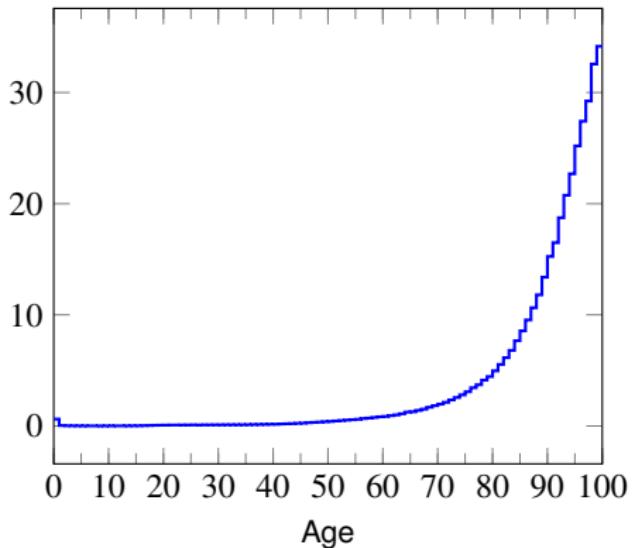
- ▶ birth rate  $b \in \mathbf{R}^{100}$ , death (or mortality) rate  $d \in \mathbf{R}^{100}$
- ▶  $b_i$  is the number of births per person with age  $i - 1$
- ▶  $d_i$  is the portion of those aged  $i - 1$  who will die this year  
(we'll take  $d_{100} = 1$ )
- ▶  $b$  and  $d$  can vary with time, but we'll assume they are constant

## Birth and death rates in the U.S.

Approximate birth rate (%)



Death rate (%)



## Dynamics

- ▶ let's find next year's population distribution  $x_{t+1}$  (ignoring immigration)
- ▶ number of 0-year-olds next year is total births this year:

$$(x_{t+1})_1 = b^T x_t$$

- ▶ number of  $i$ -year-olds next year is number of  $(i - 1)$ -year-olds this year, minus those who die:

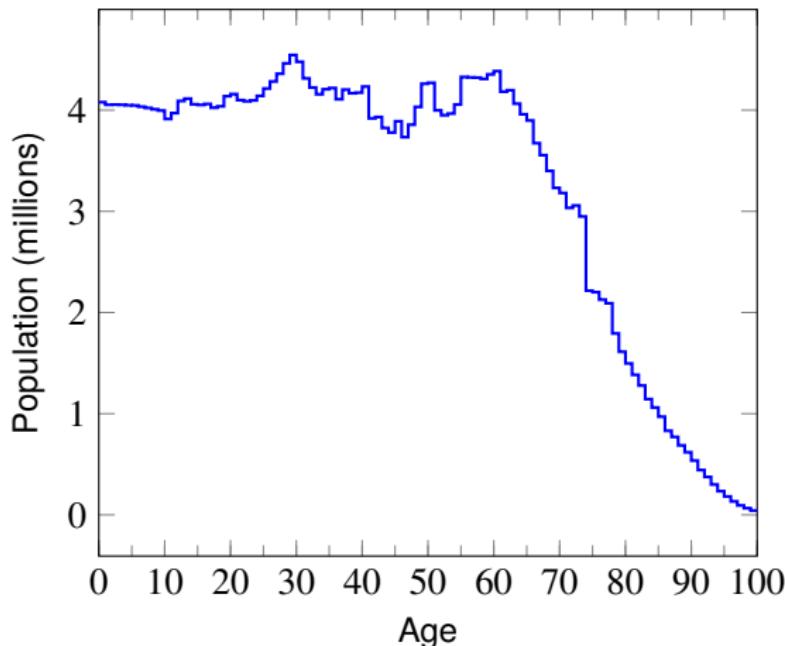
$$(x_{t+1})_{i+1} = (1 - d_i)(x_t)_i, \quad i = 1, \dots, 99$$

- ▶  $x_{t+1} = Ax_t$ , where

$$A = \begin{bmatrix} b_1 & b_2 & \cdots & b_{99} & b_{100} \\ 1 - d_1 & 0 & \cdots & 0 & 0 \\ 0 & 1 - d_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 - d_{99} & 0 \end{bmatrix}$$

## Predicting future population distributions

predicting U.S. 2020 distribution from 2010 (ignoring immigration)



# Outline

Linear dynamical systems

Population dynamics

Epidemic dynamics

## SIR model

- ▶ 4-vector  $x_t$  gives proportion of population in 4 infection states

*Susceptible:* can acquire the disease the next day

*Infected:* have the disease

*Recovered:* had the disease, recovered, now immune

*Deceased:* had the disease, and unfortunately died

- ▶ sometimes called *SIR model*
- ▶ e.g.,  $x_t = (0.75, 0.10, 0.10, 0.05)$

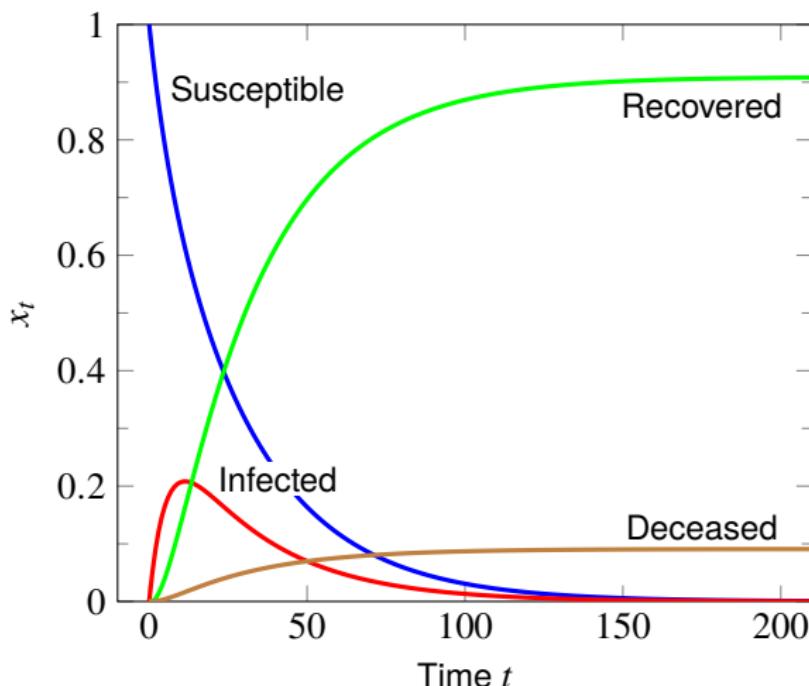
## Epidemic dynamics

over each day,

- ▶ among susceptible population,
  - 5% acquires the disease
  - 95% remain susceptible
- ▶ among infected population,
  - 1% dies
  - 10% recovers with immunity
  - 4% recover without immunity (*i.e.*, become susceptible)
  - 85% remain infected
- ▶ 100% of immune and dead people remain in their state
- ▶ epidemic dynamics as linear dynamical system

$$x_{t+1} = \begin{bmatrix} 0.95 & 0.04 & 0 & 0 \\ 0.05 & 0.85 & 0 & 0 \\ 0 & 0.10 & 1 & 0 \\ 0 & 0.01 & 0 & 1 \end{bmatrix} x_t$$

## Simulation from $x_1 = (1, 0, 0, 0)$



## 10. Matrix multiplication

# Outline

Matrix multiplication

Composition of linear functions

Matrix powers

QR factorization

## Matrix multiplication

- ▶ can multiply  $m \times p$  matrix  $A$  and  $p \times n$  matrix  $B$  to get  $C = AB$ :

$$C_{ij} = \sum_{k=1}^p A_{ik}B_{kj} = A_{i1}B_{1j} + \cdots + A_{ip}B_{pj}$$

for  $i = 1, \dots, m, j = 1, \dots, n$

- ▶ to get  $C_{ij}$ : move along  $i$ th row of  $A$ ,  $j$ th column of  $B$
- ▶ example:

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

## Special cases of matrix multiplication

- ▶ scalar-vector product (with scalar on right!)  $x\alpha$
- ▶ inner product  $a^T b$
- ▶ matrix-vector multiplication  $Ax$
- ▶ *outer product* of  $m$ -vector  $a$  and  $n$ -vector  $b$

$$ab^T = \begin{bmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & \cdots & a_2b_n \\ \vdots & \vdots & & \vdots \\ a_mb_1 & a_mb_2 & \cdots & a_mb_n \end{bmatrix}$$

# Properties

- ▶  $(AB)C = A(BC)$ , so both can be written  $ABC$
- ▶  $A(B + C) = AB + AC$
- ▶  $(AB)^T = B^T A^T$
- ▶  $AI = A$  and  $IA = A$
- ▶  $AB = BA$  does not hold in general

## Block matrices

block matrices can be multiplied using the same formula, e.g.,

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

(provided the products all make sense)

## Column interpretation

- ▶ denote columns of  $B$  by  $b_i$ :

$$B = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}$$

- ▶ then we have

$$\begin{aligned} AB &= A \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix} \\ &= \begin{bmatrix} Ab_1 & Ab_2 & \cdots & Ab_n \end{bmatrix} \end{aligned}$$

- ▶ so  $AB$  is ‘batch’ multiply of  $A$  times columns of  $B$

## Multiple sets of linear equations

- ▶ given  $k$  systems of linear equations, with same  $m \times n$  coefficient matrix

$$Ax_i = b_i, \quad i = 1, \dots, k$$

- ▶ write in compact matrix form as  $AX = B$
- ▶  $X = [x_1 \ \cdots \ x_k]$ ,  $B = [b_1 \ \cdots \ b_k]$

## Inner product interpretation

- ▶ with  $a_i^T$  the rows of  $A$ ,  $b_j$  the columns of  $B$ , we have

$$AB = \begin{bmatrix} a_1^T b_1 & a_1^T b_2 & \cdots & a_1^T b_n \\ a_2^T b_1 & a_2^T b_2 & \cdots & a_2^T b_n \\ \vdots & \vdots & & \vdots \\ a_m^T b_1 & a_m^T b_2 & \cdots & a_m^T b_n \end{bmatrix}$$

- ▶ so matrix product is all inner products of rows of  $A$  and columns of  $B$ , arranged in a matrix

## Gram matrix

- ▶ let  $A$  be an  $m \times n$  matrix with columns  $a_1, \dots, a_n$
- ▶ the *Gram matrix* of  $A$  is

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

- ▶ Gram matrix gives all inner products of columns of  $A$
- ▶ example:  $G = A^T A = I$  means columns of  $A$  are orthonormal

## Complexity

- ▶ to compute  $C_{ij} = (AB)_{ij}$  is inner product of  $p$ -vectors
- ▶ so total required flops is  $(mn)(2p) = 2mnp$  flops
- ▶ multiplying two  $1000 \times 1000$  matrices requires 2 billion flops
- ▶ ... and can be done in well under a second on current computers

# Outline

Matrix multiplication

Composition of linear functions

Matrix powers

QR factorization

## Composition of linear functions

- ▶  $A$  is an  $m \times p$  matrix,  $B$  is  $p \times n$
- ▶ define  $f : \mathbf{R}^p \rightarrow \mathbf{R}^m$  and  $g : \mathbf{R}^n \rightarrow \mathbf{R}^p$  as

$$f(u) = Au, \quad g(v) = Bv$$

- ▶  $f$  and  $g$  are linear functions
- ▶ composition of  $f$  and  $g$  is  $h : \mathbf{R}^n \rightarrow \mathbf{R}^m$  with  $h(x) = f(g(x))$
- ▶ we have

$$h(x) = f(g(x)) = A(Bx) = (AB)x$$

- ▶ composition of linear functions is linear
- ▶ associated matrix is product of matrices of the functions

## Second difference matrix

- $D_n$  is  $(n - 1) \times n$  difference matrix:

$$D_n x = (x_2 - x_1, \dots, x_n - x_{n-1})$$

- $D_{n-1}$  is  $(n - 2) \times (n - 1)$  difference matrix:

$$D_n y = (y_2 - y_1, \dots, y_{n-1} - y_{n-2})$$

- $\Delta = D_{n-1}D_n$  is  $(n - 2) \times n$  second difference matrix:

$$\Delta x = (x_1 - 2x_2 + x_3, x_2 - 2x_3 + x_4, \dots, x_{n-2} - 2x_{n-1} + x_n)$$

- for  $n = 5$ ,  $\Delta = D_{n-1}D_n$  is

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

# Outline

Matrix multiplication

Composition of linear functions

Matrix powers

QR factorization

## Matrix powers

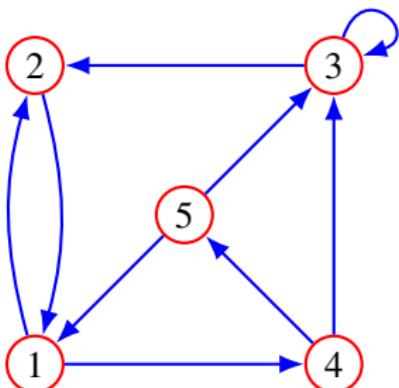
- ▶ for  $A$  square,  $A^2$  means  $AA$ , and same for higher powers
- ▶ with convention  $A^0 = I$  we have  $A^k A^l = A^{k+l}$
- ▶ negative powers later; fractional powers in other courses

## Directed graph

- $n \times n$  matrix  $A$  is adjacency matrix of directed graph:

$$A_{ij} = \begin{cases} 1 & \text{there is a edge from vertex } j \text{ to vertex } i \\ 0 & \text{otherwise} \end{cases}$$

- example:



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

## Paths in directed graph

- ▶ square of adjacency matrix:

$$(A^2)_{ij} = \sum_{k=1}^n A_{ik}A_{kj}$$

- ▶  $(A^2)_{ij}$  is number of paths of length 2 from  $j$  to  $i$
- ▶ for the example,

$$A^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

e.g., there are two paths from 4 to 3 (via 3 and 5)

- ▶ more generally,  $(A^\ell)_{ij} = \text{number of paths of length } \ell \text{ from } j$

# Outline

Matrix multiplication

Composition of linear functions

Matrix powers

QR factorization

## Gram–Schmidt in matrix notation

- ▶ run Gram–Schmidt on columns  $a_1, \dots, a_k$  of  $n \times k$  matrix  $A$
- ▶ if columns are linearly independent, get orthonormal  $q_1, \dots, q_k$
- ▶ define  $n \times k$  matrix  $Q$  with columns  $q_1, \dots, q_k$
- ▶  $Q^T Q = I$
- ▶ from Gram–Schmidt algorithm

$$\begin{aligned} a_i &= (q_1^T a_i)q_1 + \cdots + (q_{i-1}^T a_i)q_{i-1} + \|\tilde{q}_i\|q_i \\ &= R_{1i}q_1 + \cdots + R_{ii}q_i \end{aligned}$$

with  $R_{ij} = q_i^T a_j$  for  $i < j$  and  $R_{ii} = \|\tilde{q}_i\|$

- ▶ defining  $R_{ij} = 0$  for  $i > j$  we have  $A = QR$
- ▶  $R$  is upper triangular, with positive diagonal entries

## QR factorization

- ▶  $A = QR$  is called *QR factorization* of  $A$
- ▶ factors satisfy  $Q^T Q = I$ ,  $R$  upper triangular with positive diagonal entries
- ▶ can be computed using Gram–Schmidt algorithm (or some variations)
- ▶ has a *huge* number of uses, which we'll see soon

## 11. Matrix inverses

# Outline

Left and right inverses

Inverse

Solving linear equations

Examples

Pseudo-inverse

## Left inverses

- ▶ a number  $x$  that satisfies  $xa = 1$  is called the inverse of  $a$
- ▶ inverse (i.e.,  $1/a$ ) exists if and only if  $a \neq 0$ , and is unique
- ▶ a matrix  $X$  that satisfies  $XA = I$  is called a *left inverse* of  $A$
- ▶ if a left inverse exists we say that  $A$  is *left-invertible*
- ▶ example: the matrix

$$A = \begin{bmatrix} -3 & -4 \\ 4 & 6 \\ 1 & 1 \end{bmatrix}$$

has two different left inverses:

$$B = \frac{1}{9} \begin{bmatrix} -11 & -10 & 16 \\ 7 & 8 & -11 \end{bmatrix}, \quad C = \frac{1}{2} \begin{bmatrix} 0 & -1 & 6 \\ 0 & 1 & -4 \end{bmatrix}$$

## Left inverse and column independence

- ▶ if  $A$  has a left inverse  $C$  then the columns of  $A$  are linearly independent
- ▶ to see this: if  $Ax = 0$  and  $CA = I$  then

$$0 = C0 = C(Ax) = (CA)x = Ix = x$$

- ▶ we'll see later the converse is also true, so  
*a matrix is left-invertible if and only if its columns are linearly independent*
- ▶ matrix generalization of  
*a number is invertible if and only if it is nonzero*
- ▶ so left-invertible matrices are tall or square

## Solving linear equations with a left inverse

- ▶ suppose  $Ax = b$ , and  $A$  has a left inverse  $C$
- ▶ then  $Cb = C(Ax) = (CA)x = Ix = x$
- ▶ so multiplying the right-hand side by a left inverse yields the solution

## Example

$$A = \begin{bmatrix} -3 & -4 \\ 4 & 6 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

- ▶ over-determined equations  $Ax = b$  have (unique) solution  $x = (1, -1)$
- ▶  $A$  has two different left inverses,

$$B = \frac{1}{9} \begin{bmatrix} -11 & -10 & 16 \\ 7 & 8 & -11 \end{bmatrix}, \quad C = \frac{1}{2} \begin{bmatrix} 0 & -1 & 6 \\ 0 & 1 & -4 \end{bmatrix}$$

- ▶ multiplying the right-hand side with the left inverse  $B$  we get

$$Bb = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

- ▶ and also

$$Cb = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

## Right inverses

- ▶ a matrix  $X$  that satisfies  $AX = I$  is a *right inverse* of  $A$
- ▶ if a right inverse exists we say that  $A$  is *right-invertible*
- ▶  $A$  is right-invertible if and only if  $A^T$  is left-invertible:

$$AX = I \iff (AX)^T = I \iff X^T A^T = I$$

- ▶ so we conclude
- $A$  is right-invertible if and only if its rows are linearly independent*
- ▶ right-invertible matrices are wide or square

## Solving linear equations with a right inverse

- ▶ suppose  $A$  has a right inverse  $B$
- ▶ consider the (square or underdetermined) equations  $Ax = b$
- ▶  $x = Bb$  is a solution:

$$Ax = A(Bb) = (AB)b = Ib = b$$

- ▶ so  $Ax = b$  has a solution for *any*  $b$

## Example

- ▶ same  $A, B, C$  in example above
- ▶  $C^T$  and  $B^T$  are both right inverses of  $A^T$
- ▶ under-determined equations  $A^T x = (1, 2)$  has (different) solutions

$$B^T(1, 2) = (1/3, 2/3, 38/9), \quad C^T(1, 2) = (0, 1/2, -1)$$

(there are many other solutions as well)

# Outline

Left and right inverses

Inverse

Solving linear equations

Examples

Pseudo-inverse

## Inverse

- ▶ if  $A$  has a left and a right inverse, they are unique and equal (and we say that  $A$  is *invertible*)
- ▶ so  $A$  must be square
- ▶ to see this: if  $AX = I$ ,  $YA = I$

$$X = IX = (YA)X = Y(AX) = YI = Y$$

- ▶ we denote them by  $A^{-1}$ :

$$A^{-1}A = AA^{-1} = I$$

- ▶ inverse of inverse:  $(A^{-1})^{-1} = A$

## Solving square systems of linear equations

- ▶ suppose  $A$  is invertible
- ▶ for any  $b$ ,  $Ax = b$  has the unique solution

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

- ▶ matrix generalization of simple scalar equation  $ax = b$  having solution  $x = (1/a)b$  (for  $a \neq 0$ )
- ▶ simple-looking formula  $x = A^{-1}b$  is basis for many applications

## Invertible matrices

the following are equivalent for a square matrix  $A$ :

- ▶  $A$  is invertible
- ▶ columns of  $A$  are linearly independent
- ▶ rows of  $A$  are linearly independent
- ▶  $A$  has a left inverse
- ▶  $A$  has a right inverse

if any of these hold, all others do

## Examples

- ▶  $I^{-1} = I$
- ▶ if  $Q$  is orthogonal, i.e., square with  $Q^T Q = I$ , then  $Q^{-1} = Q^T$
- ▶  $2 \times 2$  matrix  $A$  is invertible if and only  $A_{11}A_{22} - A_{12}A_{21} \neq 0$

$$A^{-1} = \frac{1}{A_{11}A_{22} - A_{12}A_{21}} \begin{bmatrix} A_{22} & -A_{12} \\ -A_{21} & A_{11} \end{bmatrix}$$

- you need to know this formula
- there are similar but *much* more complicated formulas for larger matrices  
(and no, you do not need to know them)

## Non-obvious example

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

- ▶  $A$  is invertible, with inverse

$$A^{-1} = \frac{1}{30} \begin{bmatrix} 0 & -20 & -10 \\ -6 & 5 & -2 \\ 6 & 10 & 2 \end{bmatrix}.$$

- ▶ verified by checking  $AA^{-1} = I$  (or  $A^{-1}A = I$ )
- ▶ we'll soon see how to compute the inverse

## Properties

- ▶  $(AB)^{-1} = B^{-1}A^{-1}$  (provided inverses exist)
- ▶  $(A^T)^{-1} = (A^{-1})^T$  (sometimes denoted  $A^{-T}$ )
- ▶ negative matrix powers:  $(A^{-1})^k$  is denoted  $A^{-k}$
- ▶ with  $A^0 = I$ , identity  $A^k A^l = A^{k+l}$  holds for any integers  $k, l$

## Triangular matrices

- ▶ lower triangular  $L$  with nonzero diagonal entries is invertible
- ▶ so see this, write  $Lx = 0$  as

$$\begin{aligned} L_{11}x_1 &= 0 \\ L_{21}x_1 + L_{22}x_2 &= 0 \\ &\vdots \\ L_{n1}x_1 + L_{n2}x_2 + \cdots + L_{n,n-1}x_{n-1} + L_{nn}x_n &= 0 \end{aligned}$$

- from first equation,  $x_1 = 0$  (since  $L_{11} \neq 0$ )
- second equation reduces to  $L_{22}x_2 = 0$ , so  $x_2 = 0$  (since  $L_{22} \neq 0$ )
- and so on

this shows columns of  $L$  are linearly independent, so  $L$  is invertible

- ▶ upper triangular  $R$  with nonzero diagonal entries is invertible

## Inverse via QR factorization

- ▶ suppose  $A$  is square and invertible
- ▶ so its columns are linearly independent
- ▶ so Gram–Schmidt gives QR factorization
  - $A = QR$
  - $Q$  is orthogonal:  $Q^T Q = I$
  - $R$  is upper triangular with positive diagonal entries, hence invertible
- ▶ so we have

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

## Outline

Left and right inverses

Inverse

Solving linear equations

Examples

Pseudo-inverse

## Back substitution

- ▶ suppose  $R$  is upper triangular with nonzero diagonal entries
- ▶ write out  $Rx = b$  as

$$\begin{aligned} R_{11}x_1 + R_{12}x_2 + \cdots + R_{1,n-1}x_{n-1} + R_{1n}x_n &= b_1 \\ &\vdots \\ R_{n-1,n-1}x_{n-1} + R_{n-1,n}x_n &= b_{n-1} \\ R_{nn}x_n &= b_n \end{aligned}$$

- ▶ from last equation we get  $x_n = b_n/R_{nn}$
- ▶ from 2nd to last equation we get

$$x_{n-1} = (b_{n-1} - R_{n-1,n}x_n)/R_{n-1,n-1}$$

- ▶ continue to get  $x_{n-2}, x_{n-3}, \dots, x_1$

## Back substitution

- ▶ called *back substitution* since we find the variables in reverse order, substituting the already known values of  $x_i$
  - ▶ computes  $x = R^{-1}b$
  - ▶ complexity:
    - first step requires 1 flop (division)
    - 2nd step needs 3 flops
    - $i$ th step needs  $2i - 1$  flops
- total is  $1 + 3 + \dots + (2n - 1) = n^2$  flops

## Solving linear equations via QR factorization

- ▶ assuming  $A$  is invertible, let's solve  $Ax = b$ , i.e., compute  $x = A^{-1}b$
- ▶ with  $QR$  factorization  $A = QR$ , we have

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

- ▶ compute  $x = R^{-1}(Q^T b)$  by back substitution

## Solving linear equations via QR factorization

**given** an  $n \times n$  invertible matrix  $A$  and an  $n$ -vector  $b$

1. *QR factorization*: compute the QR factorization  $A = QR$
  2. compute  $Q^T b$ .
  3. *Back substitution*: Solve the triangular equation  $Rx = Q^T b$  using back substitution
- 
- ▶ complexity  $2n^3$  (step 1),  $2n^2$  (step 2),  $n^2$  (step 3)
  - ▶ total is  $2n^3 + 3n^2 \approx 2n^3$

## Multiple right-hand sides

- ▶ let's solve  $Ax_i = b_i$ ,  $i = 1, \dots, k$ , with  $A$  invertible
- ▶ carry out QR factorization once ( $2n^3$  flops)
- ▶ for  $i = 1, \dots, k$ , solve  $Rx_i = Q^T b_i$  via back substitution ( $3kn^2$  flops)
- ▶ total is  $2n^3 + 3kn^2$  flops
- ▶ if  $k$  is small compared to  $n$ , *same cost as solving one set of equations*

# Outline

Left and right inverses

Inverse

Solving linear equations

Examples

Pseudo-inverse

## Polynomial interpolation

- ▶ let's find coefficients of a cubic polynomial

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

that satisfies

$$p(-1.1) = b_1, \quad p(-0.4) = b_2, \quad p(0.1) = b_3, \quad p(0.8) = b_4$$

- ▶ write as  $Ac = b$ , with

$$A = \begin{bmatrix} 1 & -1.1 & (-1.1)^2 & (-1.1)^3 \\ 1 & -0.4 & (-0.4)^2 & (-0.4)^3 \\ 1 & 0.1 & (0.1)^2 & (0.1)^3 \\ 1 & 0.8 & (0.8)^2 & (0.8)^3 \end{bmatrix}$$

## Polynomial interpolation

- ▶ (unique) coefficients given by  $c = A^{-1}b$ , with

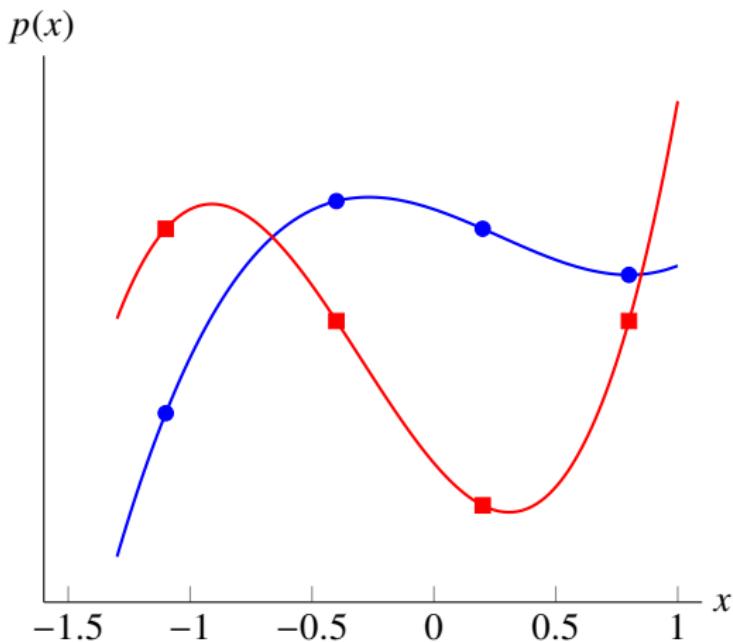
$$A^{-1} = \begin{bmatrix} -0.0201 & 0.2095 & 0.8381 & -0.0276 \\ 0.1754 & -2.1667 & 1.8095 & 0.1817 \\ 0.3133 & 0.4762 & -1.6667 & 0.8772 \\ -0.6266 & 2.381 & -2.381 & 0.6266 \end{bmatrix}$$

- ▶ so, e.g.,  $c_1$  is not very sensitive to  $b_1$  or  $b_4$
- ▶ first column gives coefficients of polynomial that satisfies

$$p(-1.1) = 1, \quad p(-0.4) = 0, \quad p(0.1) = 0, \quad p(0.8) = 0$$

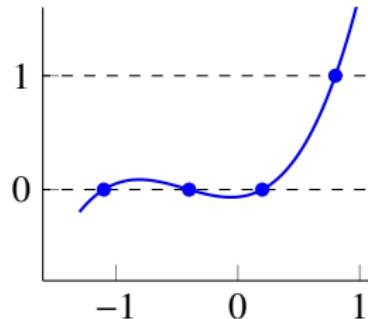
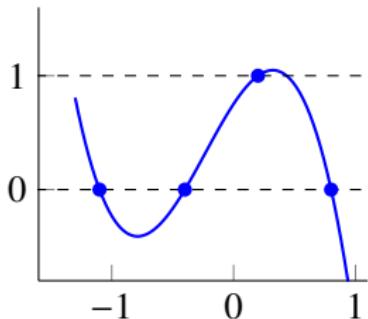
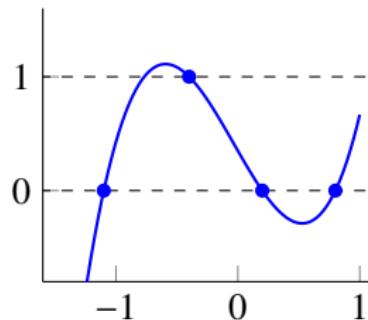
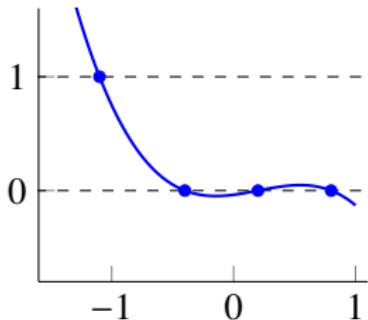
called (first) *Lagrange polynomial*

## Example



## Lagrange polynomials

Lagrange polynomials associates with points  $-1.1, -0.4, 0.2, 0.8$



## Outline

Left and right inverses

Inverse

Solving linear equations

Examples

Pseudo-inverse

## Invertibility of Gram matrix

- ▶  $A$  has linearly independent columns if and only if  $A^T A$  is invertible
- ▶ to see this, we'll show that  $Ax = 0 \Leftrightarrow A^T A x = 0$
- ▶  $\Rightarrow$ : if  $Ax = 0$  then  $(A^T A)x = A^T(Ax) = A^T 0 = 0$
- ▶  $\Leftarrow$ : if  $(A^T A)x = 0$  then

$$0 = x^T (A^T A)x = (Ax)^T (Ax) = \|Ax\|^2 = 0$$

so  $Ax = 0$

## Pseudo-inverse of tall matrix

- ▶ the *pseudo-inverse* of  $A$  with independent columns is

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

- ▶ it is a left inverse of  $A$ :

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

(we'll soon see that it's a very important left inverse of  $A$ )

- ▶ reduces to  $A^{-1}$  when  $A$  is square:

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

## Pseudo-inverse of wide matrix

- ▶ if  $A$  is wide, with linearly independent rows,  $AA^T$  is invertible
- ▶ pseudo-inverse is defined as

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

- ▶  $A^\dagger$  is a right inverse of  $A$ :

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

(we'll see later it is an important right inverse)

- ▶ reduces to  $A^{-1}$  when  $A$  is square:

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

## Pseudo-inverse via QR factorization

- ▶ suppose  $A$  has linearly independent columns,  $A = QR$
- ▶ then  $A^T A = (QR)^T (QR) = R^T Q^T QR = R^T R$
- ▶ so

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

- ▶ can compute  $A^\dagger$  using back substitution on columns of  $Q^T$
- ▶ for  $A$  with linearly independent rows,  $A^\dagger = QR^{-T}$

## 12. Least squares

# Outline

Least squares problem

Solution of least squares problem

Examples

## Least squares problem

- ▶ suppose  $m \times n$  matrix  $A$  is tall, so  $Ax = b$  is over-determined
- ▶ for most choices of  $b$ , there is no  $x$  that satisfies  $Ax = b$
- ▶ residual is  $r = Ax - b$
- ▶ least squares problem: choose  $x$  to minimize  $\|Ax - b\|^2$
- ▶  $\|Ax - b\|^2$  is the *objective function*
- ▶  $\hat{x}$  is a *solution* of least squares problem if

$$\|A\hat{x} - b\|^2 \leq \|Ax - b\|^2$$

for any  $n$ -vector  $x$

- ▶ idea:  $\hat{x}$  makes residual as small as possible, if not 0
- ▶ also called *regression* (in data fitting context)

## Least squares problem

- ▶  $\hat{x}$  called *least squares approximate solution* of  $Ax = b$
- ▶  $\hat{x}$  is sometimes called ‘solution of  $Ax = b$  in the least squares sense’
  - this is very confusing
  - never say this
  - do not associate with people who say this
- ▶  $\hat{x}$  need not (and usually does not) satisfy  $A\hat{x} = b$
- ▶ but if  $\hat{x}$  does satisfy  $A\hat{x} = b$ , then it solves least squares problem

## Column interpretation

- ▶ suppose  $a_1, \dots, a_n$  are columns of  $A$
- ▶ then

$$\|Ax - b\|^2 = \|(x_1a_1 + \dots + x_na_n) - b\|^2$$

- ▶ so least squares problem is to find a linear combination of columns of  $A$  that is closest to  $b$
- ▶ if  $\hat{x}$  is a solution of least squares problem, the  $m$ -vector

$$A\hat{x} = \hat{x}_1a_1 + \dots + \hat{x}_na_n$$

is closest to  $b$  among all linear combinations of columns of  $A$

## Row interpretation

- ▶ suppose  $\tilde{a}_1^T, \dots, \tilde{a}_m^T$  are rows of  $A$
- ▶ residual components are  $r_i = \tilde{a}_i^T x - b_i$
- ▶ least squares objective is

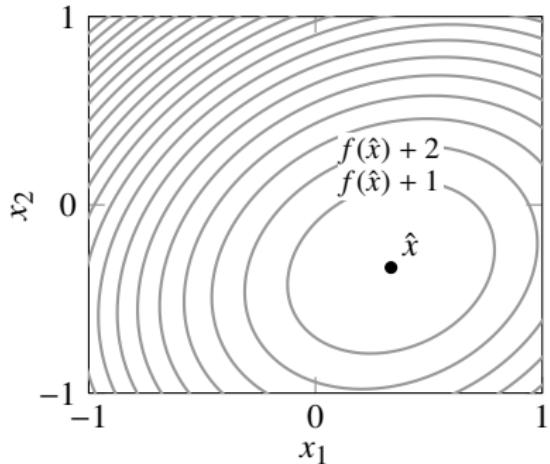
$$\|Ax - b\|^2 = (\tilde{a}_1^T x - b_1)^2 + \dots + (\tilde{a}_m^T x - b_m)^2$$

the sum of squares of the residuals

- ▶ so least squares minimizes sum of squares of residuals
  - solving  $Ax = b$  is making all residuals zero
  - least squares attempts to make them all small

## Example

$$A = \begin{bmatrix} 2 & 0 \\ -1 & 1 \\ 0 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$



- ▶  $Ax = b$  has no solution
- ▶ least squares problem is to choose  $x$  to minimize

$$\|Ax - b\|^2 = (2x_1 - 1)^2 + (-x_1 + x_2)^2 + (2x_2 + 1)^2$$

- ▶ least squares approximate solution is  $\hat{x} = (1/3, 1/3)$  (say, via calculus)
- ▶  $\|A\hat{x} - b\|^2 = 2/3$  is smallest possible value of  $\|Ax - b\|^2$
- ▶  $A\hat{x} = (2/3, -2/3, -2/3)$  is linear combination of columns of  $A$  closest to  $b$

# Outline

Least squares problem

Solution of least squares problem

Examples

## Solution of least squares problem

- ▶ we make one assumption:  $A$  has linearly independent columns
- ▶ this implies that Gram matrix  $A^T A$  is invertible
- ▶ unique solution of least squares problem is

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

- ▶ cf.  $x = A^{-1}b$ , solution of square invertible system  $Ax = b$

## Derivation via calculus

- ▶ define

$$f(x) = \|Ax - b\|^2 = \sum_{i=1}^m \left( \sum_{j=1}^n A_{ij}x_j - b_i \right)^2$$

- ▶ solution  $\hat{x}$  satisfies

$$\frac{\partial f}{\partial x_k}(\hat{x}) = \nabla f(\hat{x})_k = 0, \quad k = 1, \dots, n$$

- ▶ taking partial derivatives we get  $\nabla f(x)_k = (2A^T(Ax - b))_k$
- ▶ in matrix-vector notation:  $\nabla f(\hat{x}) = 2A^T(A\hat{x} - b) = 0$
- ▶ so  $\hat{x}$  satisfies *normal equations*  $(A^T A)\hat{x} = A^T b$
- ▶ and therefore  $\hat{x} = (A^T A)^{-1}A^T b$

## Direct verification

- ▶ let  $\hat{x} = (A^T A)^{-1} A^T b$ , so  $A^T(A\hat{x} - b) = 0$
- ▶ for any  $n$ -vector  $x$  we have

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

- ▶ so for any  $x$ ,  $\|Ax - b\|^2 \geq \|A\hat{x} - b\|^2$
- ▶ if equality holds,  $A(x - \hat{x}) = 0$ , which implies  $x = \hat{x}$  since columns of  $A$  are linearly independent

## Computing least squares approximate solutions

- ▶ compute QR factorization of  $A$ :  $A = QR$  ( $2mn^2$  flops)
  - ▶ QR factorization exists since columns of  $A$  are linearly independent
  - ▶ to compute  $\hat{x} = A^\dagger b = R^{-1}Q^T b$ 
    - form  $Q^T b$  ( $2mn$  flops)
    - compute  $\hat{x} = R^{-1}(Q^T b)$  via back substitution ( $n^2$  flops)
  - ▶ total complexity  $2mn^2$  flops
- 
- ▶ identical to algorithm for solving  $Ax = b$  for square invertible  $A$
  - ▶ but when  $A$  is tall, gives least squares approximate solution

# Outline

Least squares problem

Solution of least squares problem

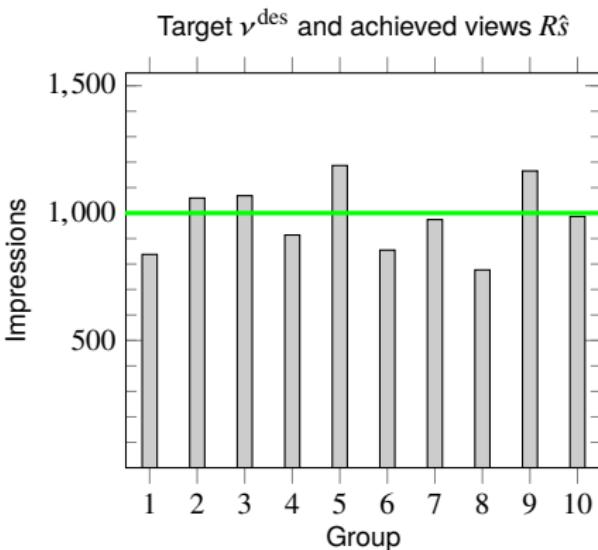
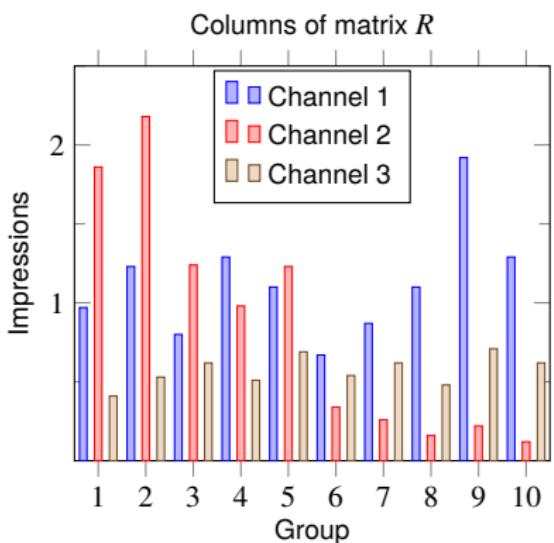
Examples

## Advertising purchases

- ▶  $m$  demographics groups we want to advertise to
- ▶  $v^{\text{des}}$  is  $m$ -vector of target views or impressions
- ▶  $n$ -vector  $s$  gives spending on  $n$  advertising channels
- ▶  $m \times n$  matrix  $R$  gives demographic reach of channels
- ▶  $R_{ij}$  is number of views per dollar spent (in 1000/\$)
- ▶  $v = Rs$  is  $m$ -vector of views across demographic groups
- ▶  $\|v^{\text{des}} - Rs\|/\sqrt{m}$  is RMS deviation from desired views
- ▶ we'll use least squares spending  $\hat{s} = R^\dagger v^{\text{des}}$  (need not be  $\geq 0$ )

## Example

- ▶  $m = 10$  groups,  $n = 3$  channels
- ▶ target views vector  $v^{\text{des}} = 10^3 \times \mathbf{1}$
- ▶ optimal spending is  $\hat{s} = (62, 100, 1443)$



## Illumination

- ▶  $n$  lamps illuminate an area divided in  $m$  regions
- ▶  $A_{ij}$  is illumination in region  $i$  if lamp  $j$  is on with power 1, other lamps are off
- ▶  $x_j$  is power of lamp  $j$
- ▶  $(Ax)_i$  is illumination level at region  $i$
- ▶  $b_i$  is target illumination level at region  $i$

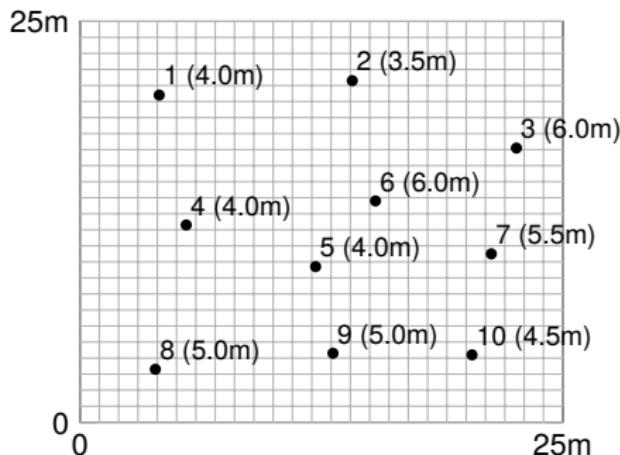
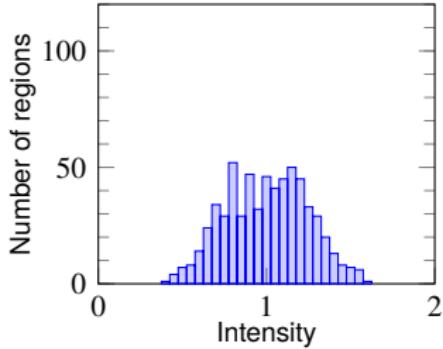
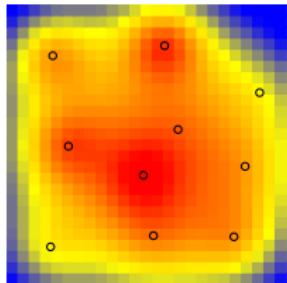


figure shows lamp positions for example with

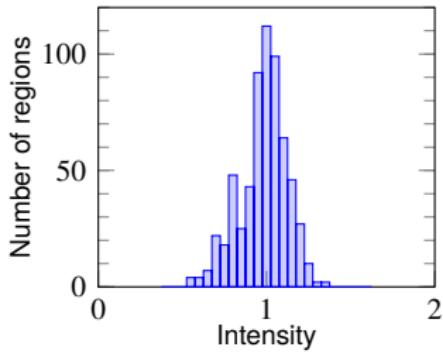
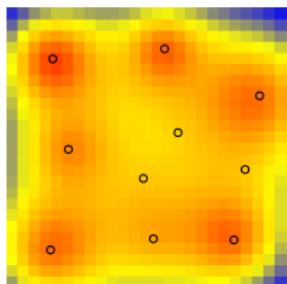
$$m = 25^2, \quad n = 10$$

## Illumination

- equal lamp powers ( $x = \mathbf{1}$ )



- least squares solution  $\hat{x}$ , with  $b = \mathbf{1}$



## 13. Least squares data fitting

# Outline

Least squares model fitting

Validation

Feature engineering

## Setup

- ▶ we believe a scalar  $y$  and an  $n$ -vector  $x$  are related by *model*

$$y \approx f(x)$$

- ▶  $x$  is called the *independent variable*
- ▶  $y$  is called the *outcome* or *response variable*
- ▶  $f : \mathbf{R}^n \rightarrow \mathbf{R}$  gives the relation between  $x$  and  $y$
- ▶ often  $x$  is a feature vector, and  $y$  is something we want to predict
- ▶ we don't know  $f$ , which gives the 'true' relationship between  $x$  and  $y$

## Data

- ▶ we are given some *data*

$$x^{(1)}, \dots, x^{(N)}, \quad y^{(1)}, \dots, y^{(N)}$$

also called *observations*, *examples*, *samples*, or *measurements*

- ▶  $x^{(i)}, y^{(i)}$  is *i*th *data pair*
- ▶  $x_j^{(i)}$  is the *j*th component of *i*th data point  $x^{(i)}$

## Model

- ▶ choose *model*  $\hat{f} : \mathbf{R}^n \rightarrow \mathbf{R}$ , a *guess* or *approximation* of  $f$
- ▶ *linear in the parameters* model form:

$$\hat{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x)$$

- ▶  $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$  are *basis functions* that we choose
- ▶  $\theta_i$  are *model parameters* that we choose
- ▶  $\hat{y}^{(i)} = \hat{f}(x^{(i)})$  is (the model's) *prediction* of  $y^{(i)}$
- ▶ we'd like  $\hat{y}^{(i)} \approx y^{(i)}$ , i.e., model is consistent with observed data

## Least squares data fitting

- ▶ *prediction error or residual* is  $r_i = y^{(i)} - \hat{y}^{(i)}$
- ▶ *least squares data fitting*: choose model parameters  $\theta_i$  to minimize RMS prediction error on data set

$$\left( \frac{(r^{(1)})^2 + \cdots + (r^{(N)})^2}{N} \right)^{1/2}$$

- ▶ this can be formulated (and solved) as a least squares problem

## Least squares data fitting

- ▶ express  $y^{(i)}$ ,  $\hat{y}^{(i)}$ , and  $r^{(i)}$  as  $N$ -vectors
  - $y^d = (y^{(1)}, \dots, y^{(N)})$  is vector of outcomes
  - $\hat{y}^d = (\hat{y}^{(1)}, \dots, \hat{y}^{(N)})$  is vector of predictions
  - $r^d = (r^{(1)}, \dots, r^{(N)})$  is vector of residuals
- ▶  $\text{rms}(r^d)$  is *RMS prediction error*
- ▶ define  $N \times p$  matrix  $A$  with elements  $A_{ij} = f_j(x^{(i)})$ , so  $\hat{y}^d = A\theta$
- ▶ least squares data fitting: choose  $\theta$  to minimize

$$\|r^d\|^2 = \|y^d - \hat{y}^d\|^2 = \|y^d - A\theta\|^2 = \|A\theta - y^d\|^2$$

- ▶  $\hat{\theta} = (A^T A)^{-1} A^T y$  (if columns of  $A$  are linearly independent)
- ▶  $\|A\hat{\theta} - y\|^2/N$  is *minimum mean-square (fitting) error*

## Fitting a constant model

- ▶ simplest possible model:  $p = 1, f_1(x) = 1$ , so model  $\hat{f}(x) = \theta_1$  is a constant

- ▶  $A = \mathbf{1}$ , so

$$\hat{\theta}_1 = (\mathbf{1}^T \mathbf{1})^{-1} \mathbf{1}^T y^d = (1/N) \mathbf{1}^T y^d = \text{avg}(y^d)$$

- ▶ the mean of  $y^{(1)}, \dots, y^{(N)}$  is the least squares fit by a constant
- ▶ MMSE is  $\text{std}(y^d)^2$ ; RMS error is  $\text{std}(y^d)$
- ▶ more sophisticated models are judged against the constant model

## Fitting univariate functions

- ▶ when  $n = 1$ , we seek to approximate a function  $f : \mathbf{R} \rightarrow \mathbf{R}$
- ▶ we can plot the data  $(x_i, y_i)$  and the model function  $\hat{y} = \hat{f}(x)$

## Straight-line fit

- ▶  $p = 2$ , with  $f_1(x) = 1, f_2(x) = x$
- ▶ model has form  $\hat{f}(x) = \theta_1 + \theta_2 x$
- ▶ matrix  $A$  has form

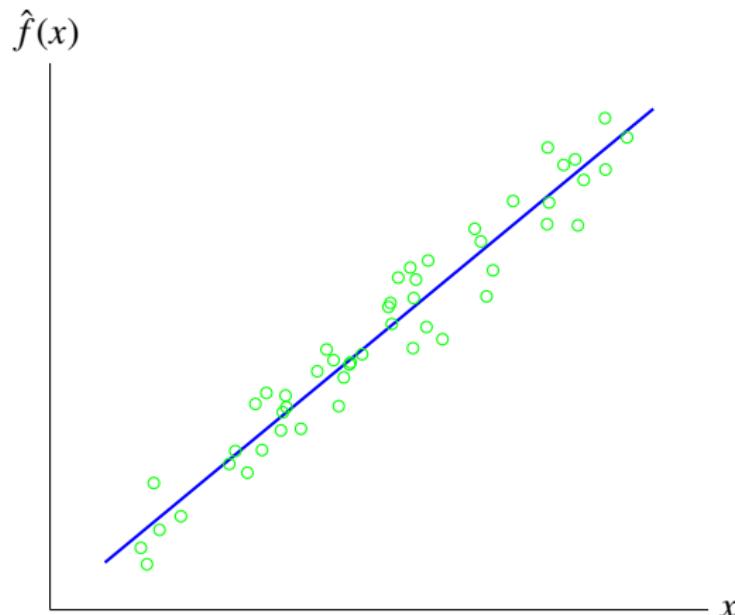
$$A = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \vdots & \vdots \\ 1 & x^{(N)} \end{bmatrix}$$

- ▶ can work out  $\hat{\theta}_1$  and  $\hat{\theta}_2$  explicitly:

$$\hat{f}(x) = \text{avg}(y^d) + \rho \frac{\text{std}(y^d)}{\text{std}(x^d)} (x - \text{avg}(x^d))$$

where  $x^d = (x^{(1)}, \dots, x^{(N)})$

## Example



## Asset $\alpha$ and $\beta$

- ▶  $x$  is return of whole market,  $y$  is return of a particular asset
- ▶ write straight-line model as

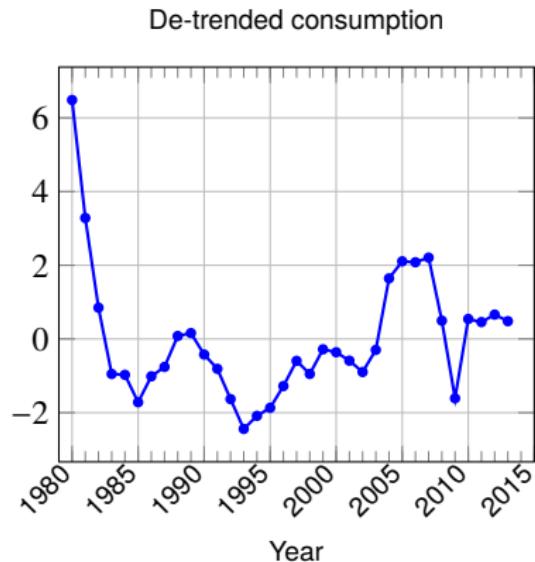
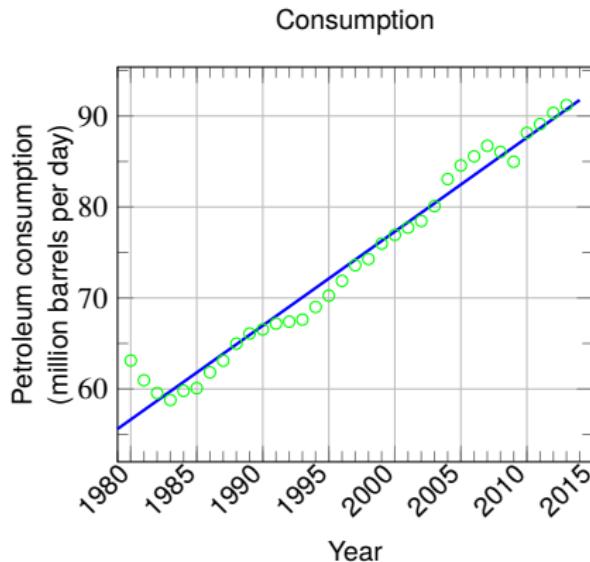
$$\hat{y} = (r^{\text{rf}} + \alpha) + \beta(x - \mu^{\text{mkt}})$$

- $\mu^{\text{mkt}}$  is the average market return
  - $r^{\text{rf}}$  is the risk-free interest rate
  - several other slightly different definitions are used
- ▶ called asset ' $\alpha$ ' and ' $\beta$ ', widely used

## Time series trend

- ▶  $y^{(i)}$  is value of quantity at time  $x^{(i)} = i$
- ▶  $\hat{y}^{(i)} = \hat{\theta}_1 + \hat{\theta}_2 i, \quad i = 1, \dots, N$ , is called *trend line*
- ▶  $y^d - \hat{y}^d$  is called *de-trended time series*
- ▶  $\hat{\theta}_2$  is *trend coefficient*

## World petroleum consumption



## Polynomial fit

- ▶  $f_i(x) = x^{i-1}$ ,  $i = 1, \dots, p$
- ▶ model is a polynomial of degree less than  $p$

$$\hat{f}(x) = \theta_1 + \theta_2 x + \dots + \theta_p x^{p-1}$$

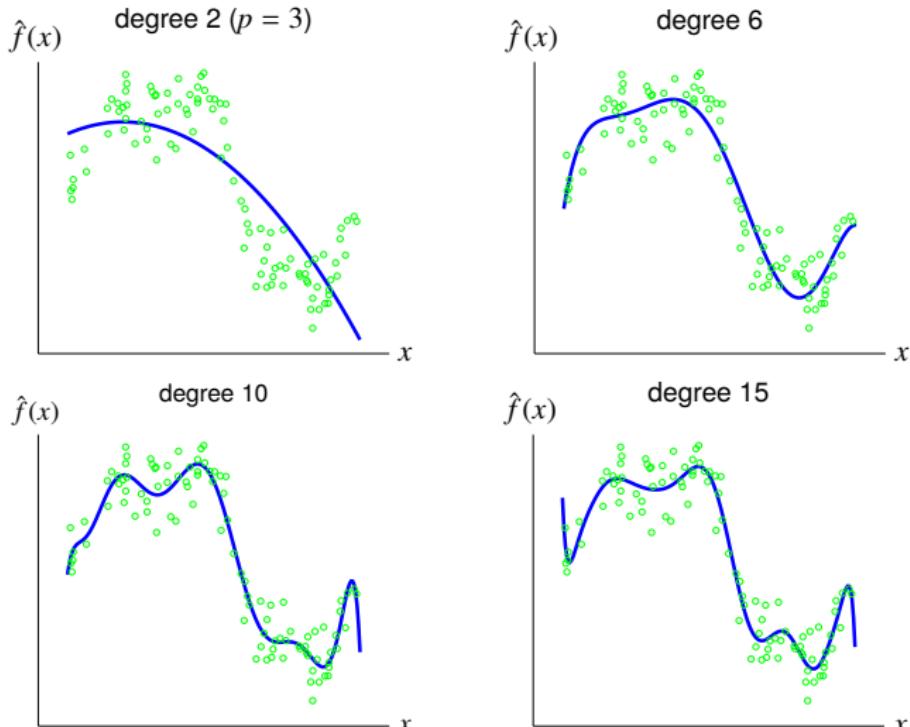
(here  $x^i$  means scalar  $x$  to  $i$ th power;  $x^{(i)}$  is  $i$ th data point)

- ▶  $A$  is Vandermonde matrix

$$A = \begin{bmatrix} 1 & x^{(1)} & \dots & (x^{(1)})^{p-1} \\ 1 & x^{(2)} & \dots & (x^{(2)})^{p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x^{(N)} & \dots & (x^{(N)})^{p-1} \end{bmatrix}$$

## Example

$N = 100$  data points



## Regression as general data fitting

- ▶ regression model is affine function  $\hat{y} = \hat{f}(x) = x^T \beta + v$
- ▶ fits general fitting form with basis functions

$$f_1(x) = 1, \quad f_i(x) = x_{i-1}, \quad i = 2, \dots, n+1$$

so model is

$$\hat{y} = \theta_1 + \theta_2 x_1 + \dots + \theta_{n+1} x_n = x^T \theta_{2:n} + \theta_1$$

- ▶  $\beta = \theta_{2:n+1}$ ,  $v = \theta_1$

## General data fitting as regression

- ▶ general fitting model  $\hat{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x)$
- ▶ common assumption:  $f_1(x) = 1$
- ▶ same as regression model  $\hat{f}(\tilde{x}) = \tilde{x}^T \beta + v$ , with
  - $\tilde{x} = (f_2(x), \dots, f_p(x))$  are ‘transformed features’
  - $v = \theta_1$ ,  $\beta = \theta_{2:p}$

## Auto-regressive time series model

- ▶ time series  $z_1, z_2, \dots$
- ▶ *auto-regressive* (AR) prediction model:

$$\hat{z}_{t+1} = \theta_1 z_t + \cdots + \theta_M z_{t-M+1}, \quad t = M, M+1, \dots$$

- ▶  $M$  is *memory* of model
- ▶  $\hat{z}_{t+1}$  is prediction of next value, based on previous  $M$  values
- ▶ we'll choose  $\beta$  to minimize sum of squares of prediction errors,

$$(\hat{z}_{M+1} - z_{M+1})^2 + \cdots + (\hat{z}_T - z_T)^2$$

- ▶ put in general form with

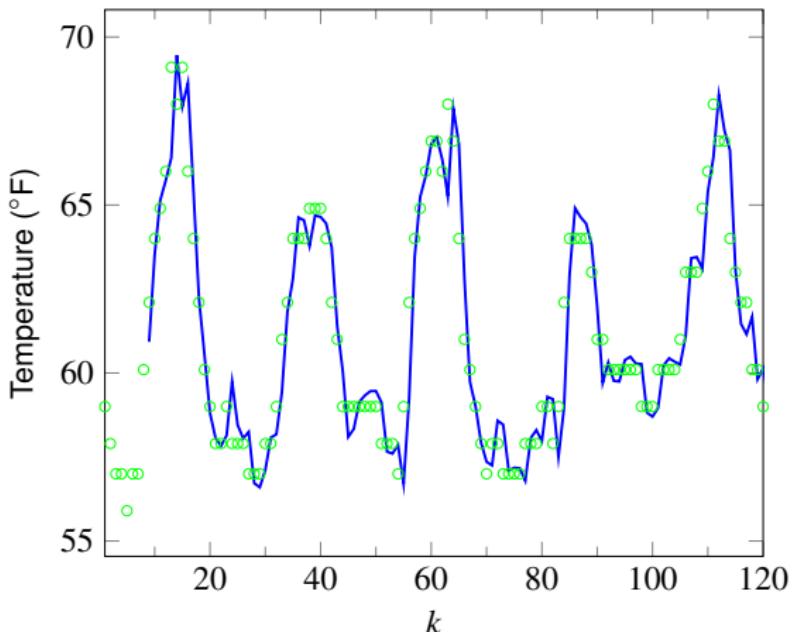
$$y^{(i)} = z_{M+i}, \quad x^{(i)} = (z_{M+i-1}, \dots, z_i), \quad i = 1, \dots, T - M$$

## Example

- ▶ hourly temperature at LAX in May 2016, length 744
- ▶ average is  $61.76^{\circ}\text{F}$ , standard deviation  $3.05^{\circ}\text{F}$
- ▶ predictor  $\hat{z}_{t+1} = z_t$  gives RMS error  $1.16^{\circ}\text{F}$
- ▶ predictor  $\hat{z}_{t+1} = z_{t-23}$  gives RMS error  $1.73^{\circ}\text{F}$
- ▶ AR model with  $M = 8$  gives RMS error  $0.98^{\circ}\text{F}$

## Example

solid line shows one-hour ahead predictions from AR model, first 5 days



# Outline

Least squares model fitting

Validation

Feature engineering

## Generalization

basic idea:

- ▶ goal of model is *not* to predict outcome for the given data
  - ▶ instead it is to *predict the outcome on new, unseen data*
- 
- ▶ a model that makes reasonable predictions on new, unseen data has *generalization ability*, or *generalizes*
  - ▶ a model that makes poor predictions on new, unseen data is said to suffer from *over-fit*

## Validation

a simple and effective method to guess if a model will generalize

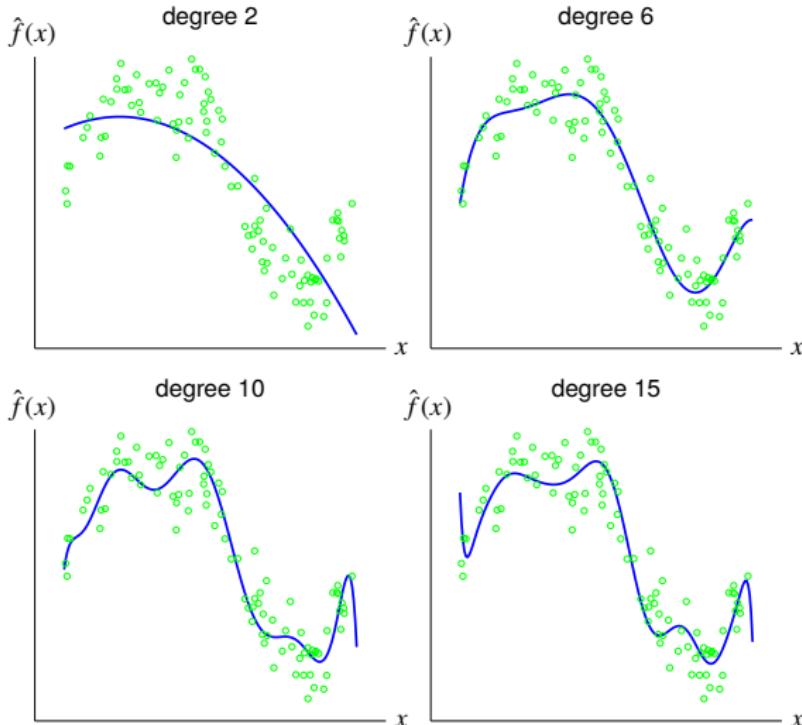
- ▶ split original data into a *training set* and a *test set*
- ▶ typical splits: 80%/20%, 90%/10%
- ▶ build ('train') model on training data set
- ▶ then *check the model's predictions on the test data set*
- ▶ (can also compare RMS prediction error on train and test data)
- ▶ if they are similar, we can *guess* the model will generalize

## Validation

- ▶ can be used to choose among different candidate models, *e.g.*
  - polynomials of different degrees
  - regression models with different sets of regressors
- ▶ we'd use one with low, or lowest, test error

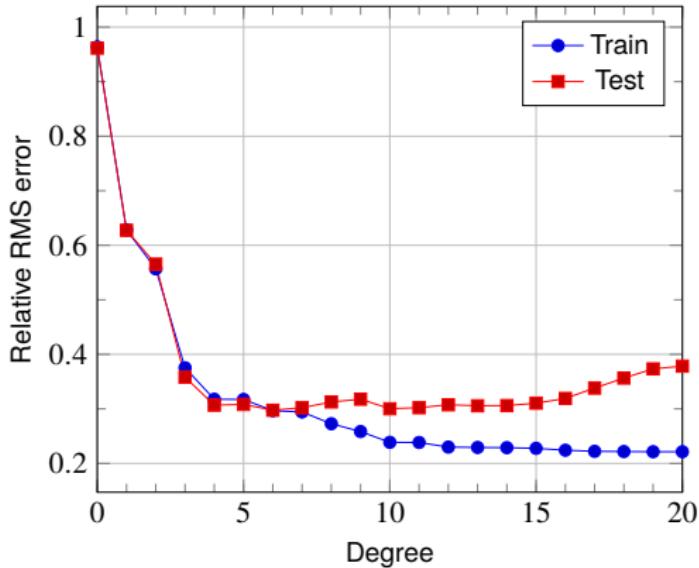
## Example

models fit using *training set* of 100 points; plots show *test set* of 100 points



## Example

- ▶ suggests degree 4, 5, or 6 are reasonable choices



## Cross validation

to carry out cross validation:

- ▶ divide data into 10 *folds*
- ▶ for  $i = 1, \dots, 10$ , build (train) model using all folds except  $i$
- ▶ test model on data in fold  $i$

interpreting cross validation results:

- ▶ if test RMS errors are much larger than train RMS errors, model is over-fit
- ▶ if test and train RMS errors are similar and consistent, we can guess the model will have a similar RMS error on future data

## Example

- ▶ house price, regression fit with  $x = (\text{area}/1000 \text{ ft.}^2, \text{bedrooms})$
- ▶ 774 sales, divided into 5 folds of 155 sales each
- ▶ fit 5 regression models, removing each fold

Fold	Model parameters			RMS error	
	$\nu$	$\beta_1$	$\beta_2$	Train	Test
1	60.65	143.36	-18.00	74.00	78.44
2	54.00	151.11	-20.30	75.11	73.89
3	49.06	157.75	-21.10	76.22	69.93
4	47.96	142.65	-14.35	71.16	88.35
5	60.24	150.13	-21.11	77.28	64.20

# Outline

Least squares model fitting

Validation

Feature engineering

## Feature engineering

- ▶ start with original or base feature  $n$ -vector  $x$
- ▶ choose basis functions  $f_1, \dots, f_p$  to create ‘mapped’ feature  $p$ -vector

$$(f_1(x), \dots, f_p(x))$$

- ▶ now fit linear in parameters model with mapped features

$$\hat{y} = \theta_1 f_1(x) + \dots + \theta_p f_p(x)$$

- ▶ *check the model using validation*

## Transforming features

- ▶ *standardizing features*: replace  $x_i$  with

$$(x_i - b_i)/a_i$$

- $b_i \approx$  mean value of the feature across the data
- $a_i \approx$  standard deviation of the feature across the data

new features are called *z-scores*

- ▶ *log transform*: if  $x_i$  is nonnegative and spans a wide range, replace it with

$$\log(1 + x_i)$$

- ▶ *hi and lo features*: create new features given by

$$\max\{x_1 - b, 0\}, \quad \min\{x_1 - a, 0\}$$

(called hi and lo versions of original feature  $x_i$ )

## Example

- ▶ house price prediction
- ▶ start with base features
  - $x_1$  is area of house (in 1000ft.<sup>2</sup>)
  - $x_2$  is number of bedrooms
  - $x_3$  is 1 for condo, 0 for house
  - $x_4$  is zip code of address (62 values)
- ▶ we'll use  $p = 8$  basis functions:
  - $f_1(x) = 1, f_2(x) = x_1, f_3(x) = \max\{x_1 - 1.5, 0\}$
  - $f_4(x) = x_2, f_5(x) = x_3$
  - $f_6(x), f_7(x), f_8(x)$  are Boolean functions of  $x_4$  which encode 4 groups of nearby zip codes (*i.e.*, neighborhood)
- ▶ five fold model validation

## Example

Fold	Model parameters								RMS error	
	$\theta_1$	$\theta_2$	$\theta_3$	$\theta_4$	$\theta_5$	$\theta_6$	$\theta_7$	$\theta_8$	Train	Test
1	122.35	166.87	-39.27	-16.31	-23.97	-100.42	-106.66	-25.98	67.29	72.78
2	100.95	186.65	-55.80	-18.66	-14.81	-99.10	-109.62	-17.94	67.83	70.81
3	133.61	167.15	-23.62	-18.66	-14.71	-109.32	-114.41	-28.46	69.70	63.80
4	108.43	171.21	-41.25	-15.42	-17.68	-94.17	-103.63	-29.83	65.58	78.91
5	114.45	185.69	-52.71	-20.87	-23.26	-102.84	-110.46	-23.43	70.69	58.27

## 14. Least squares classification

# Outline

## Classification

Least squares classification

Multi-class classifiers

## Classification

- ▶ data fitting with outcome that takes on (non-numerical) values like
  - TRUE OR FALSE
  - SPAM OR NOT SPAM
  - DOG, HORSE, OR MOUSE
- ▶ outcome values are called *labels* or *categories*
- ▶ data fitting is called *classification*
- ▶ we start with case when there are two possible outcomes
- ▶ called *Boolean* or *2-way classification*
- ▶ we encode outcomes as +1 (TRUE) and -1 (FALSE)
- ▶ classifier has form  $\hat{y} = \hat{f}(x), f : \mathbf{R}^n \rightarrow \{-1, +1\}$

## Applications

- ▶ email spam detection
  - $x$  contains features of an email message (word counts, ...)
- ▶ financial transaction fraud detection
  - $x$  contains features of proposed transaction, initiator
- ▶ document classification (say, politics or not)
  - $x$  is word count histogram of document
- ▶ disease detection
  - $x$  contains patient features, results of medical tests
- ▶ digital communications receiver
  - $y$  is transmitted bit;  $x$  contain  $n$  measurements of received signal

## Prediction errors

- ▶ data point  $(x, y)$ , predicted outcome  $\hat{y} = \hat{f}(x)$
- ▶ only four possibilities:
  - *True positive.*  $y = +1$  and  $\hat{y} = +1$ .
  - *True negative.*  $y = -1$  and  $\hat{y} = -1$ .

(in these two cases, the prediction is *correct*)

- *False positive.*  $y = -1$  and  $\hat{y} = +1$ .
- *False negative.*  $y = +1$  and  $\hat{y} = -1$ .

(in these two cases, the prediction is *wrong*)

- ▶ the errors have many other names, like Type I and Type II

## Confusion matrix

- ▶ given data set  $x^{(1)}, \dots, x^{(N)}$ ,  $y^{(1)}, \dots, y^{(N)}$  and classifier  $\hat{f}$
- ▶ count each of the four outcomes

		$\hat{y} = +1$	$\hat{y} = -1$	Total
$y = +1$	$N_{tp}$	$N_{fn}$	$N_p$	
$y = -1$	$N_{fp}$	$N_{tn}$	$N_n$	
All	$N_{tp} + N_{fp}$	$N_{fn} + N_{tp}$	$N$	

- ▶ off-diagonal terms are prediction errors
- ▶ many error rates and accuracy measures are used
  - *error rate* is  $(N_{fp} + N_{fn})/N$
  - *true positive* (or *recall*) rate is  $N_{tp}/N_p$
  - *false positive rate* (or *false alarm rate*) is  $N_{fp}/N_n$
- ▶ a proposed classifier is judged by its error rate(s) on a test set

## Example

- ▶ spam filter performance on a test set (say)

	$\hat{y} = +1$ (SPAM)	$\hat{y} = -1$ (not SPAM)	Total
$y = +1$ (SPAM)	95	32	127
$y = -1$ (not SPAM)	19	1120	1139
All	114	1152	1266

- ▶ error rate is  $(19 + 32)/1266 = 4.03\%$
- ▶ false positive rate is  $19/1139 = 1.67\%$

# Outline

Classification

Least squares classification

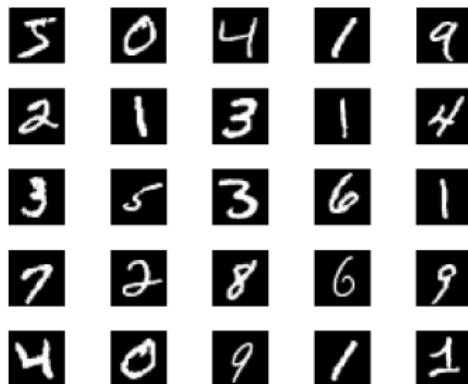
Multi-class classifiers

## Least squares classification

- ▶ fit model  $\tilde{f}$  to encoded ( $\pm 1$ )  $y^{(i)}$  values *using standard least squares data fitting*
- ▶  $\tilde{f}(x)$  should be near  $+1$  when  $y = +1$ , and near  $-1$  when  $y = -1$
- ▶  $\tilde{f}(x)$  is a *number*
- ▶ use model  $\hat{f}(x) = \text{sign}(\tilde{f}(x))$
- ▶ (size of  $\tilde{f}(x)$  is related to the ‘confidence’ in the prediction)

## Handwritten digits example

- ▶ MNIST data set of 70000  $28 \times 28$  images of digits 0, ..., 9



- ▶ divided into training set (60000) and test set (10000)
- ▶  $x$  is 494-vector, constant 1 plus the 493 pixel values with nonzero values in at least 600 training examples
- ▶  $y = +1$  if digit is 0;  $-1$  otherwise

## Least squares classifier results

- ▶ training set results (error rate 1.6%)

	$\hat{y} = +1$	$\hat{y} = -1$	Total
$y = +1$	5158	765	5923
$y = -1$	167	53910	54077
All	5325	54675	60000

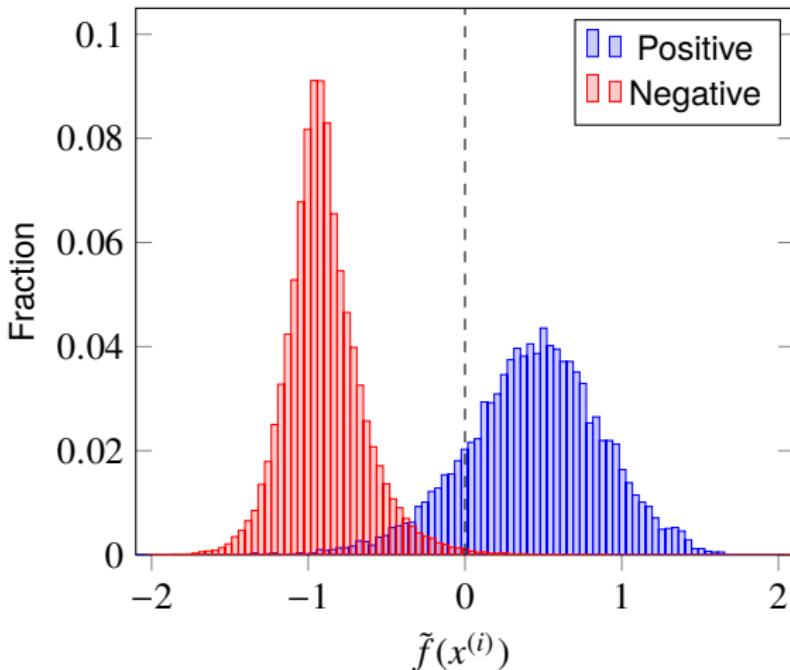
- ▶ test set results (error rate 1.6%)

	$\hat{y} = +1$	$\hat{y} = -1$	Total
$y = +1$	864	116	980
$y = -1$	42	8978	9020
All	906	9094	10000

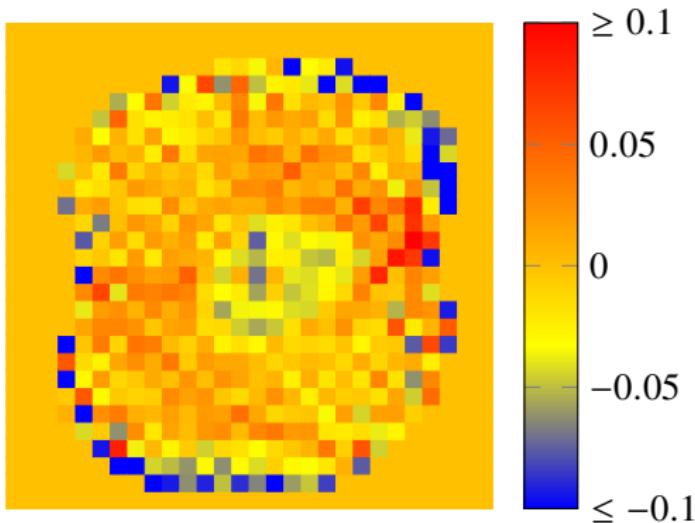
- ▶ we can likely achieve 1.6% error rate on unseen images

## Distribution of least squares fit

distribution of values of  $\tilde{f}(x^{(i)})$  over training set



## Coefficients in least squares classifier



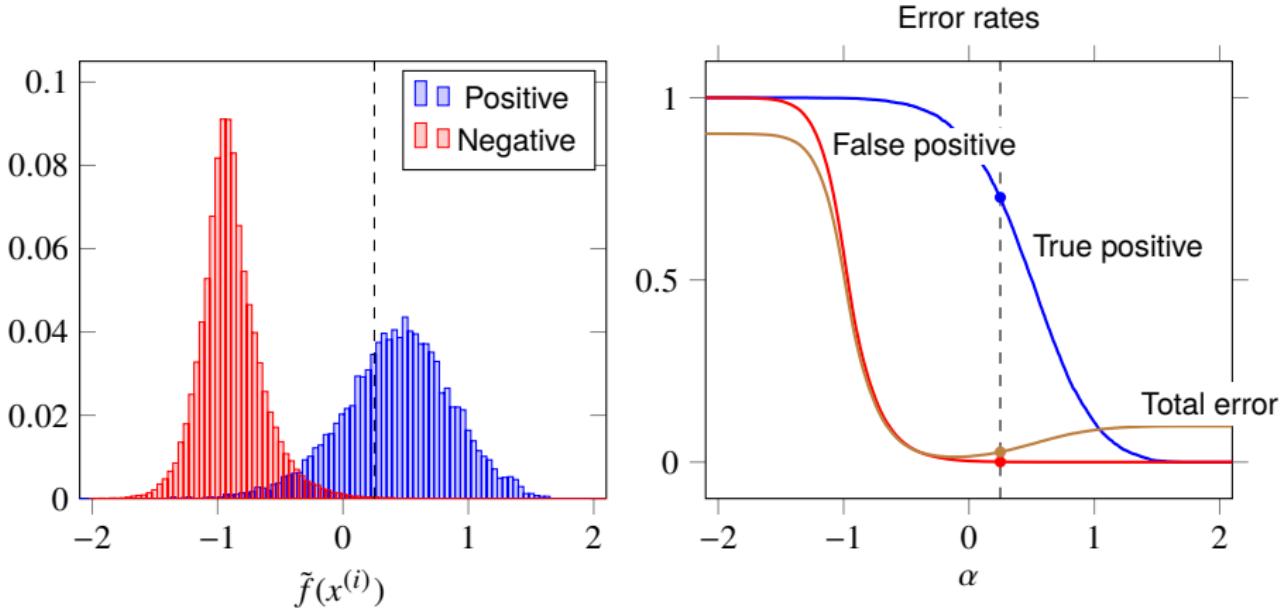
## Skewed decision threshold

- ▶ use predictor  $\hat{f}(x) = \text{sign}(\tilde{f}(x) - \alpha)$ , i.e.,

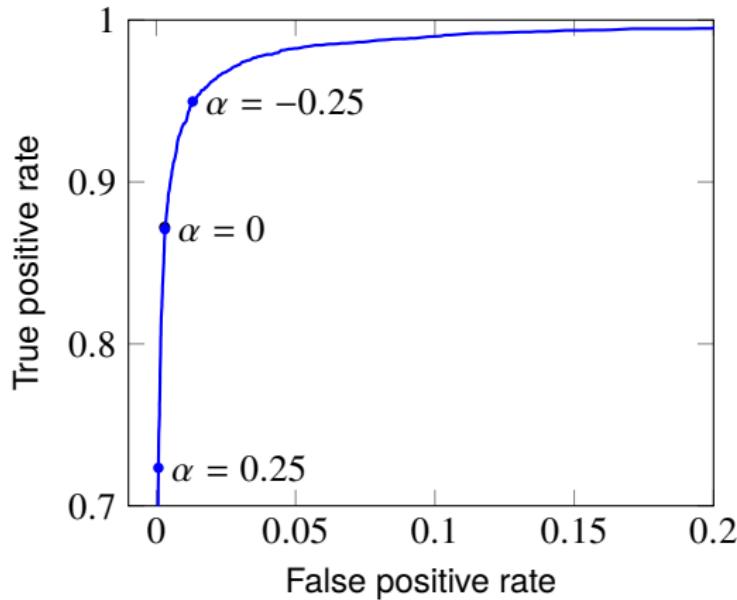
$$\hat{f}(x) = \begin{cases} +1 & \tilde{f}(x) \geq \alpha \\ -1 & \tilde{f}(x) < \alpha \end{cases}$$

- ▶  $\alpha$  is the *decision threshold*
- ▶ for positive  $\alpha$ , false positive rate is lower but so is true positive rate
- ▶ for negative  $\alpha$ , false positive rate is higher but so is true positive rate
- ▶ trade off curve of true positive versus false positive rates is called *receiver operating characteristic* (ROC)

## Example



## ROC curve



# Outline

Classification

Least squares classification

Multi-class classifiers

## Multi-class classifiers

- ▶ we have  $K > 2$  possible labels, with label set  $\{1, \dots, K\}$
- ▶ predictor is  $\hat{f} : \mathbf{R}^n \rightarrow \{1, \dots, K\}$
- ▶ for given predictor and data set, confusion matrix is  $K \times K$
- ▶ some off-diagonal entries may be much worse than others

## Examples

- ▶ handwritten digit classification
  - guess the digit written, from the pixel values
- ▶ marketing demographic classification
  - guess the demographic group, from purchase history
- ▶ disease diagnosis
  - guess diagnosis from among a set of candidates, from test results, patient features
- ▶ translation word choice
  - choose how to translate a word into several choices, given context features
- ▶ document topic prediction
  - guess topic from word count histogram

## Least squares multi-class classifier

- ▶ create a least squares classifier for each label versus the others
- ▶ take as classifier

$$\hat{f}(x) = \operatorname{argmax}_{\ell \in \{1, \dots, K\}} \tilde{f}_\ell(x)$$

(i.e., choose  $\ell$  with largest value of  $\tilde{f}_\ell(x)$ )

- ▶ for example, with

$$\tilde{f}_1(x) = -0.7, \quad \tilde{f}_2(x) = +0.2, \quad \tilde{f}_3(x) = +0.8$$

we choose  $\hat{f}(x) = 3$

## Handwritten digit classification

confusion matrix, test set

Digit	Prediction										Total
	0	1	2	3	4	5	6	7	8	9	
0	944	0	1	2	2	8	13	2	7	1	980
1	0	1107	2	2	3	1	5	1	14	0	1135
2	18	54	815	26	16	0	38	22	39	4	1032
3	4	18	22	884	5	16	10	22	20	9	1010
4	0	22	6	0	883	3	9	1	12	46	982
5	24	19	3	74	24	656	24	13	38	17	892
6	17	9	10	0	22	17	876	0	7	0	958
7	5	43	14	6	25	1	1	883	1	49	1028
8	14	48	11	31	26	40	17	13	756	18	974
9	16	10	3	17	80	0	1	75	4	803	1009
All	1042	1330	887	1042	1086	742	994	1032	898	947	10000

error rate is around 14% (same as for training set)

## Adding new features

- ▶ let's add 5000 random features (!),  $\max\{(Rx)_j, 0\}$ 
  - $R$  is  $5000 \times 494$  matrix with entries  $\pm 1$ , chosen randomly
- ▶ now use least squares classification with 5494 feature vector
- ▶ results: training set error 1.5%, test set error 2.6%
- ▶ can do better with a little more thought in generating new features
- ▶ indeed, even better than humans can do (!!)

## Results with new features

confusion matrix, test set

Digit	Prediction										Total
	0	1	2	3	4	5	6	7	8	9	
0	972	0	0	2	0	1	1	1	3	0	980
1	0	1126	3	1	1	0	3	0	1	0	1135
2	6	0	998	3	2	0	4	7	11	1	1032
3	0	0	3	977	0	13	0	5	8	4	1010
4	2	1	3	0	953	0	6	3	1	13	982
5	2	0	1	5	0	875	5	0	3	1	892
6	8	3	0	0	4	6	933	0	4	0	958
7	0	8	12	0	2	0	1	992	3	10	1028
8	3	1	3	6	4	3	2	2	946	4	974
9	4	3	1	12	11	7	1	3	3	964	1009
All	997	1142	1024	1006	977	905	956	1013	983	997	10000

## 15. Multi-objective least squares

# Outline

Multi-objective least squares problem

Control

Estimation and inversion

Regularized data fitting

## Multi-objective least squares

- ▶ goal: choose  $n$ -vector  $x$  so that  $k$  norm squared objectives

$$J_1 = \|A_1x - b_1\|^2, \dots, J_k = \|A_kx - b_k\|^2$$

are all small

- ▶  $A_i$  is an  $m_i \times n$  matrix,  $b_i$  is an  $m_i$ -vector,  $i = 1, \dots, k$
- ▶  $J_i$  are the objectives in a *multi-objective optimization problem* (also called a *multi-criterion problem*)
- ▶ could choose  $x$  to minimize any one  $J_i$ , but we want *one*  $x$  that makes them all small

## Weighted sum objective

- ▶ choose positive *weights*  $\lambda_1, \dots, \lambda_k$  and form *weighted sum objective*

$$J = \lambda_1 J_1 + \dots + \lambda_k J_k = \lambda_1 \|A_1 x - b_1\|^2 + \dots + \lambda_k \|A_k x - b_k\|^2$$

- ▶ we'll choose  $x$  to minimize  $J$
- ▶ we can take  $\lambda_1 = 1$ , and call  $J_1$  the *primary objective*
- ▶ interpretation of  $\lambda_i$ : how much we care about  $J_i$  being small, relative to primary objective
- ▶ for a bi-criterion problem, we will minimize

$$J_1 + \lambda J_2 = \|A_1 x - b_1\|^2 + \lambda \|A_2 x - b_2\|^2$$

## Weighted sum minimization via stacking

- ▶ write weighted-sum objective as

$$J = \left\| \begin{bmatrix} \sqrt{\lambda_1}(A_1x - b_1) \\ \vdots \\ \sqrt{\lambda_k}(A_kx - b_k) \end{bmatrix} \right\|^2$$

- ▶ so we have  $J = \|\tilde{A}x - \tilde{b}\|^2$ , with

$$\tilde{A} = \begin{bmatrix} \sqrt{\lambda_1}A_1 \\ \vdots \\ \sqrt{\lambda_k}A_k \end{bmatrix}, \quad \tilde{b} = \begin{bmatrix} \sqrt{\lambda_1}b_1 \\ \vdots \\ \sqrt{\lambda_k}b_k \end{bmatrix}$$

- ▶ so we can minimize  $J$  using basic ('single-criterion') least squares

## Weighted sum solution

- ▶ assuming columns of  $\tilde{A}$  are independent,

$$\begin{aligned}\hat{x} &= (\tilde{A}^T \tilde{A})^{-1} \tilde{A}^T \tilde{b} \\ &= (\lambda_1 A_1^T A_1 + \cdots + \lambda_k A_k^T A_k)^{-1} (\lambda_1 A_1^T b_1 + \cdots + \lambda_k A_k^T b_k)\end{aligned}$$

- ▶ can compute  $\hat{x}$  via QR factorization of  $\tilde{A}$
- ▶  $A_i$  can be wide, or have dependent columns

## Optimal trade-off curve

- ▶ bi-criterion problem with objectives  $J_1, J_2$
- ▶ let  $\hat{x}(\lambda)$  be minimizer of  $J_1 + \lambda J_2$
- ▶ called *Pareto optimal*: there is no point  $z$  that satisfies

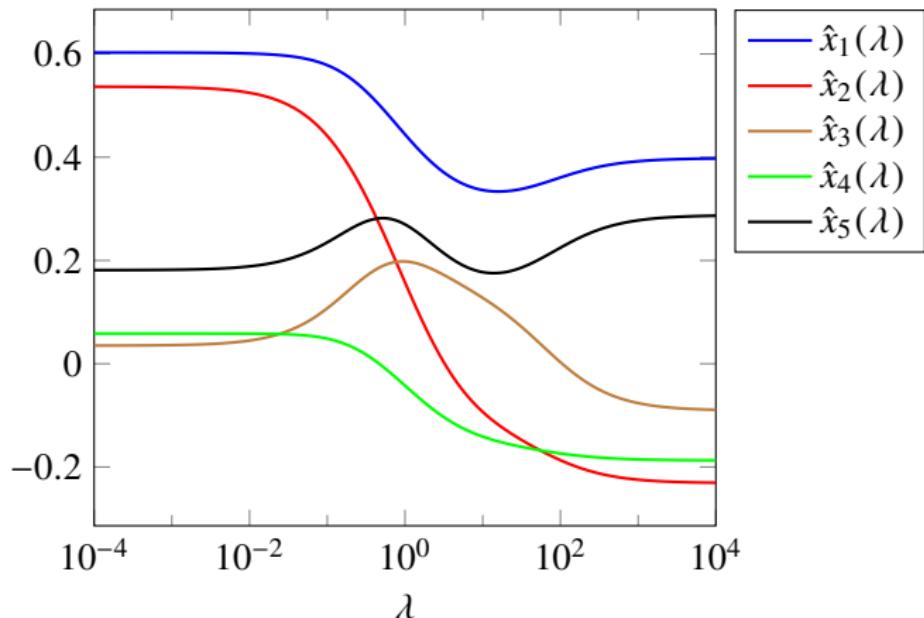
$$J_1(z) < J_1(\hat{x}(\lambda)), \quad J_2(z) < J_2(\hat{x}(\lambda))$$

i.e., no other point  $x$  beats  $\hat{x}$  on both objectives

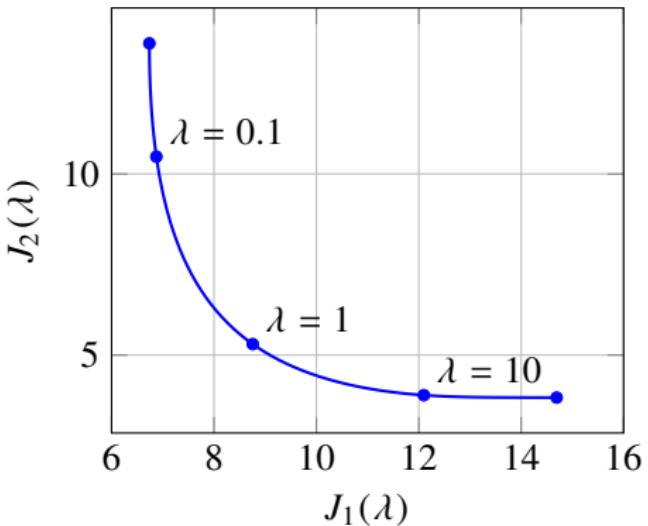
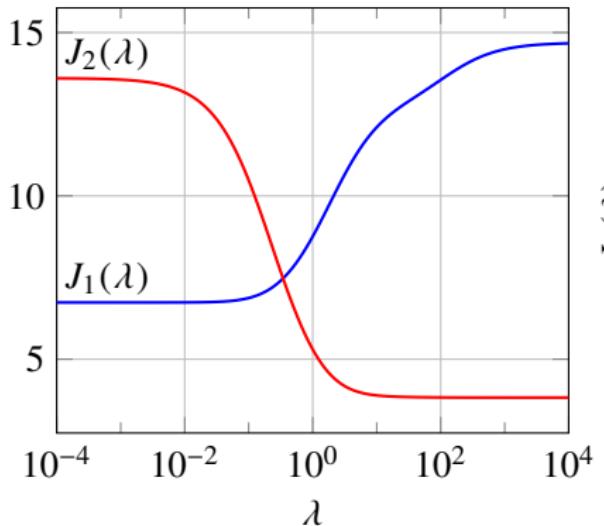
- ▶ *optimal trade-off curve*:  $(J_1(\hat{x}(\lambda)), J_2(\hat{x}(\lambda)))$  for  $\lambda > 0$

## Example

$A_1$  and  $A_2$  both  $10 \times 5$



## Objectives versus $\lambda$ and optimal trade-off curve



## Using multi-objective least squares

- ▶ identify the primary objective
  - the basic quantity we want to minimize
- ▶ choose one or more secondary objectives
  - quantities we'd also like to be small, if possible
  - *e.g.*, size of  $x$ , roughness of  $x$ , distance from some given point
- ▶ tweak/tune the weights until we like (or can tolerate)  $\hat{x}(\lambda)$
- ▶ for bi-criterion problem with  $J = J_1 + \lambda J_2$ :
  - if  $J_2$  is too big, increase  $\lambda$
  - if  $J_1$  is too big, decrease  $\lambda$

# Outline

Multi-objective least squares problem

Control

Estimation and inversion

Regularized data fitting

## Control

- ▶  $n$ -vector  $x$  corresponds to *actions* or *inputs*
- ▶  $m$ -vector  $y$  corresponds to *results* or *outputs*
- ▶ inputs and outputs are related by affine input-output model

$$y = Ax + b$$

- ▶  $A$  and  $b$  are known (from analytical models, data fitting ...)
- ▶ the goal is to choose  $x$  (which determines  $y$ ), to optimize multiple objectives on  $x$  and  $y$

## Multi-objective control

- ▶ typical primary objective:  $J_1 = \|y - y^{\text{des}}\|^2$ , where  $y^{\text{des}}$  is a given desired or target output
- ▶ typical secondary objectives:
  - $x$  is small:  $J_2 = \|x\|^2$
  - $x$  is not far from a nominal input:  $J_2 = \|x - x^{\text{nom}}\|^2$

## Product demand shaping

- ▶ we will change prices of  $n$  products by  $n$ -vector  $\delta^{\text{price}}$
- ▶ this induces change in demand  $\delta^{\text{dem}} = E^d \delta^{\text{price}}$
- ▶  $E^d$  is the  $n \times n$  price elasticity of demand matrix
- ▶ we want  $J_1 = \|\delta^{\text{dem}} - \delta^{\text{tar}}\|^2$  small
- ▶ and also, we want  $J_2 = \|\delta^{\text{price}}\|^2$  small
- ▶ so we minimize  $J_1 + \lambda J_2$ , and adjust  $\lambda > 0$
- ▶ trades off deviation from target demand and price change magnitude

## Robust control

- ▶ we have  $K$  different input-output models (a.k.a. *scenarios*)

$$y^{(k)} = A^{(k)}x + b^{(k)}, \quad k = 1, \dots, K$$

- ▶ these represent uncertainty in the system
- ▶  $y^{(k)}$  is the output with input  $x$ , if system model  $k$  is correct
- ▶ average cost across the models:

$$\frac{1}{K} \sum_{k=1}^K \|y^{(k)} - y^{\text{des}}\|^2$$

- ▶ can add terms for  $x$  as well, e.g.,  $\lambda \|x\|^2$
- ▶ yields choice of  $x$  that does well under all scenarios

# Outline

Multi-objective least squares problem

Control

Estimation and inversion

Regularized data fitting

## Estimation

- ▶ measurement model:  $y = Ax + v$
- ▶  $n$ -vector  $x$  contains parameters we want to estimate
- ▶  $m$ -vector  $y$  contains the measurements
- ▶  $m$ -vector  $v$  are (unknown) *noises* or *measurement errors*
- ▶  $m \times n$  matrix  $A$  connects parameters to measurements
- ▶ *basic least squares estimation*: assuming  $v$  is small (and  $A$  has independent columns), we guess  $x$  by minimizing  $J_1 = \|Ax - y\|^2$

## Regularized inversion

- ▶ can get far better results by incorporating prior information about  $x$  into estimation, e.g.,
  - $x$  should be not too large
  - $x$  should be smooth
- ▶ express these as secondary objectives:
  - $J_2 = \|x\|^2$  ('Tikhonov regularization')
  - $J_2 = \|Dx\|^2$
- ▶ we minimize  $J_1 + \lambda J_2$
- ▶ adjust  $\lambda$  until you like the results
- ▶ curve of  $\hat{x}(\lambda)$  versus  $\lambda$  is called *regularization path*
- ▶ with Tikhonov regularization, works even when  $A$  has dependent columns (e.g., when it is wide)

## Image de-blurring

- ▶  $x$  is an image
- ▶  $A$  is a blurring operator
- ▶  $y = Ax + v$  is a blurred, noisy image
- ▶ least squares de-blurring: choose  $x$  to minimize

$$\|Ax - y\|^2 + \lambda(\|D_v x\|^2 + \|D_h x\|^2)$$

$D_v, D_h$  are vertical and horizontal differencing operations

- ▶  $\lambda$  controls smoothing of de-blurred image

## Example

blurred, noisy image



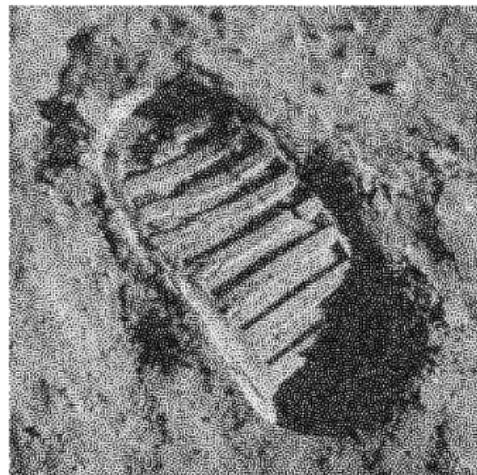
regularized inversion with  $\lambda = 0.007$



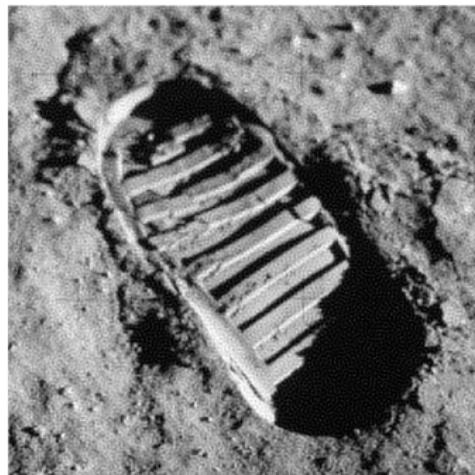
Image credit: NASA

## Regularization path

$$\lambda = 10^{-6}$$



$$\lambda = 10^{-4}$$



## Regularization path

$$\lambda = 10^{-2}$$



$$\lambda = 1$$

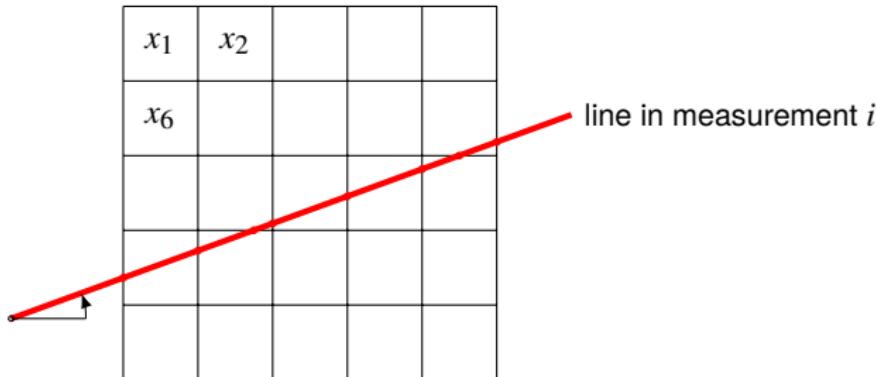


## Tomography

- ▶  $x$  represents values in region of interest of  $n$  voxels (pixels)
- ▶  $y = Ax + v$  are measurements of integrals along lines through region

$$y_i = \sum_{j=1}^n A_{ij}x_j + v_i$$

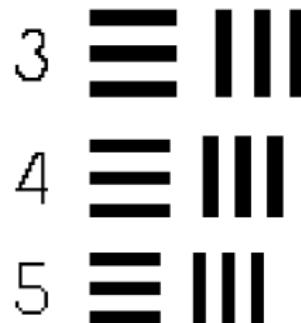
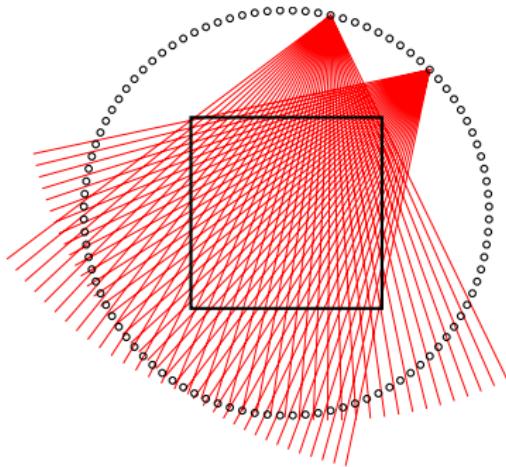
- ▶  $A_{ij}$  is the length of the intersection of the line in measurement  $i$  with voxel  $j$



## Least squares tomographic reconstruction

- ▶ primary objective is  $\|Ax - y\|^2$
- ▶ regularization terms capture prior information about  $x$
- ▶ for example, if  $x$  varies smoothly over region, use Dirichlet energy for graph that connects each voxel to its neighbors

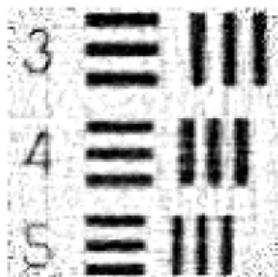
## Example



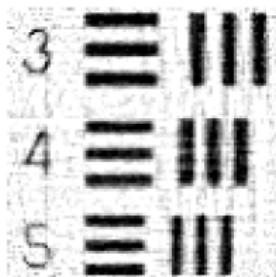
- ▶ left: 4000 lines (100 points, 40 lines per point)
- ▶ right: object placed in the square region on the left
- ▶ region of interest is divided in 10000 pixels

## Regularized least squares reconstruction

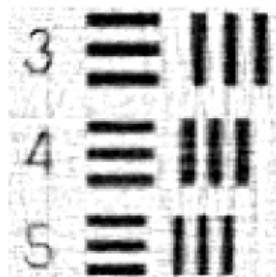
$$\lambda = 10^{-2}$$



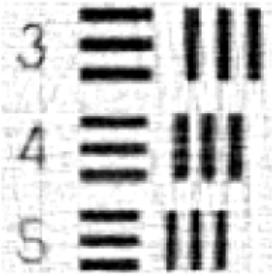
$$\lambda = 10^{-1}$$



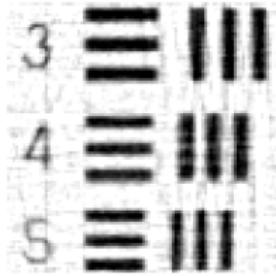
$$\lambda = 1$$



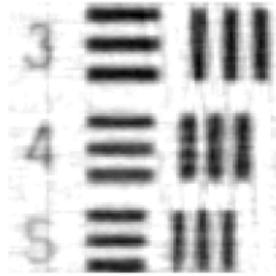
$$\lambda = 5$$



$$\lambda = 10$$



$$\lambda = 100$$



# Outline

Multi-objective least squares problem

Control

Estimation and inversion

Regularized data fitting

## Motivation for regularization

- ▶ consider data fitting model (of relationship  $y \approx f(x)$ )

$$\hat{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x)$$

with  $f_1(x) = 1$

- ▶  $\theta_i$  is the sensitivity of  $\hat{f}(x)$  to  $f_i(x)$
- ▶ so large  $\theta_i$  means the model is very sensitive to  $f_i(x)$
- ▶  $\theta_1$  is an exception, since  $f_1(x) = 1$  never varies
- ▶ so, we don't want  $\theta_2, \dots, \theta_p$  to be too large

## Regularized data fitting

- ▶ suppose we have data  $(x_1, y_1), \dots, (x_N, y_N)$
- ▶ express fitting error as  $A\theta - y$
- ▶ *regularized data fitting*: choose  $\theta$  to minimize

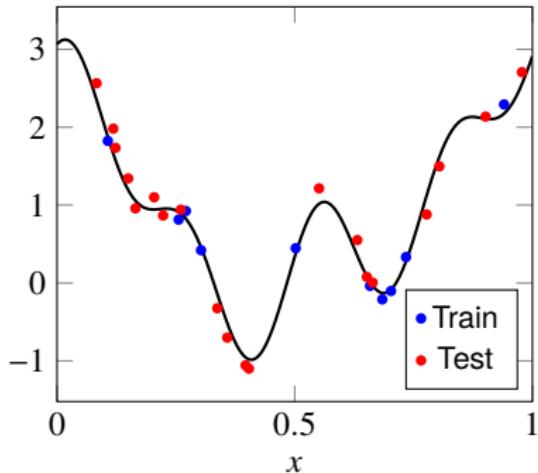
$$\|A\theta - y\|^2 + \lambda \|\theta_{2:p}\|^2$$

- ▶  $\lambda > 0$  is the *regularization parameter*
- ▶ for regression model  $\hat{y} = X^T \beta + v\mathbf{1}$ , we minimize

$$\|X^T \beta + v\mathbf{1} - y\|^2 + \lambda \|\beta\|^2$$

- ▶ choose  $\lambda$  by validation on a test set

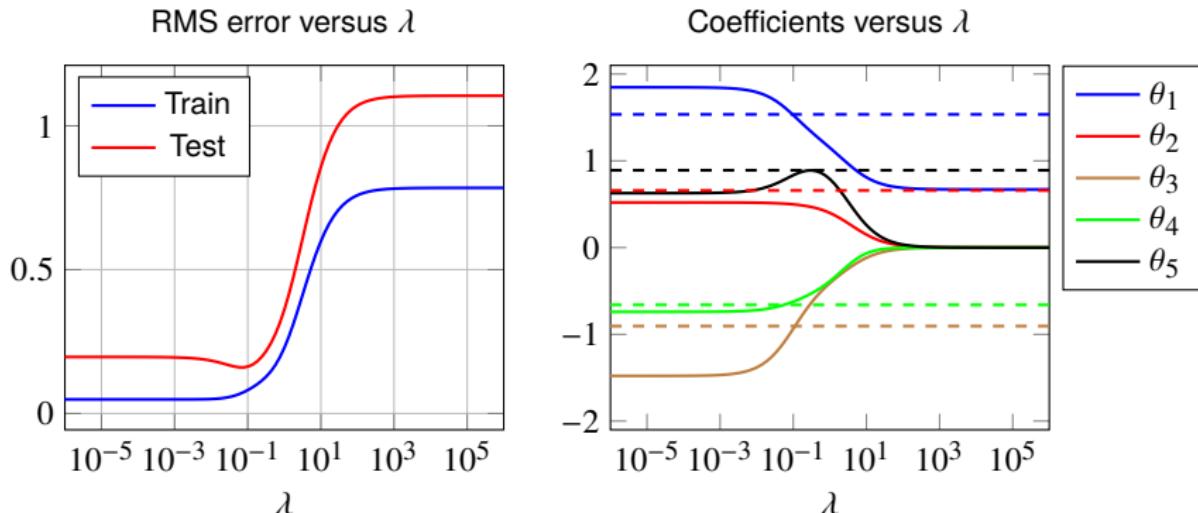
## Example



- ▶ solid line is signal used to generate synthetic (simulated) data
- ▶ 10 blue points are used as training set; 20 red points are used as test set
- ▶ we fit a model with five parameters  $\theta_1, \dots, \theta_5$ :

$$\hat{f}(x) = \theta_1 + \sum_{k=1}^4 \theta_{k+1} \cos(\omega_k x + \phi_k) \quad (\text{with given } \omega_k, \phi_k)$$

## Result of regularized least squares fit



- ▶ minimum test RMS error is for  $\lambda$  around 0.08
- ▶ increasing  $\lambda$  ‘shrinks’ the coefficients  $\theta_2, \dots, \theta_5$
- ▶ dashed lines show coefficients used to generate the data
- ▶ for  $\lambda$  near 0.08, estimated coefficients are close to these ‘true’ values

## 16. Constrained least squares

# Outline

Linearly constrained least squares

Least norm problem

Solving the constrained least squares problem

## Least squares with equality constraints

- ▶ the (linearly) *constrained least squares problem* (CLS) is

$$\begin{aligned} & \text{minimize} && \|Ax - b\|^2 \\ & \text{subject to} && Cx = d \end{aligned}$$

- ▶ variable (to be chosen/found) is  $n$ -vector  $x$
- ▶  $m \times n$  matrix  $A$ ,  $m$ -vector  $b$ ,  $p \times n$  matrix  $C$ , and  $p$ -vector  $d$  are *problem data* (i.e., they are given)
- ▶  $\|Ax - b\|^2$  is the *objective function*
- ▶  $Cx = d$  are the *equality constraints*
- ▶  $x$  is *feasible* if  $Cx = d$
- ▶  $\hat{x}$  is a *solution* of CLS if  $C\hat{x} = d$  and  $\|A\hat{x} - b\|^2 \leq \|Ax - b\|^2$  holds for any  $n$ -vector  $x$  that satisfies  $Cx = d$

## Least squares with equality constraints

- ▶ CLS combines solving linear equations with least squares problem
- ▶ like a bi-objective least squares problem, with infinite weight on second objective  $\|Cx - d\|^2$

## Piecewise-polynomial fitting

- ▶ piecewise-polynomial  $\hat{f}$  has form

$$\hat{f}(x) = \begin{cases} p(x) = \theta_1 + \theta_2x + \theta_3x^2 + \theta_4x^3 & x \leq a \\ q(x) = \theta_5 + \theta_6x + \theta_7x^2 + \theta_8x^3 & x > a \end{cases}$$

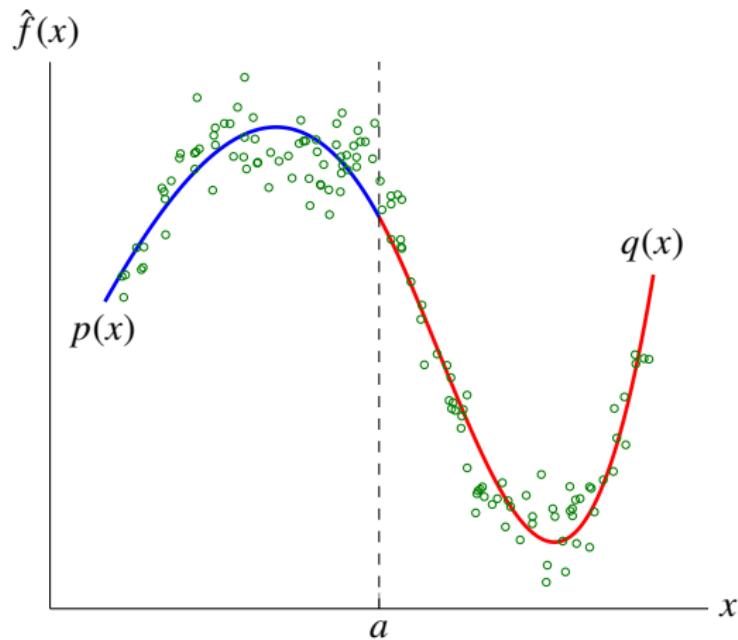
( $a$  is given)

- ▶ we require  $p(a) = q(a), p'(a) = q'(a)$
- ▶ fit  $\hat{f}$  to data  $(x_i, y_i), i = 1, \dots, N$  by minimizing sum square error

$$\sum_{i=1}^N (\hat{f}(x_i) - y_i)^2$$

- ▶ can express as a constrained least squares problem

## Example



## Piecewise-polynomial fitting

- constraints are (linear equations in  $\theta$ )

$$\begin{aligned}\theta_1 + \theta_2a + \theta_3a^2 + \theta_4a^3 - \theta_5 - \theta_6a - \theta_7a^2 - \theta_8a^3 &= 0 \\ \theta_2 + 2\theta_3a + 3\theta_4a^2 - \theta_6 - 2\theta_7a - 3\theta_8a^2 &= 0\end{aligned}$$

- prediction error on  $(x_i, y_i)$  is  $a_i^T \theta - y_i$ , with

$$(a_i)_j = \begin{cases} (1, x_i, x_i^2, x_i^3, 0, 0, 0, 0) & x_i \leq a \\ (0, 0, 0, 0, 1, x_i, x_i^2, x_i^3) & x_i > a \end{cases}$$

- sum square error is  $\|A\theta - y\|^2$ , where  $a_i^T$  are rows of  $A$

# Outline

Linearly constrained least squares

Least norm problem

Solving the constrained least squares problem

## Least norm problem

- ▶ special case of constrained least squares problem, with  $A = I, b = 0$
- ▶ *least-norm problem:*

$$\begin{aligned} & \text{minimize} && \|x\|^2 \\ & \text{subject to} && Cx = d \end{aligned}$$

i.e., find the smallest vector that satisfies a set of linear equations

## Force sequence

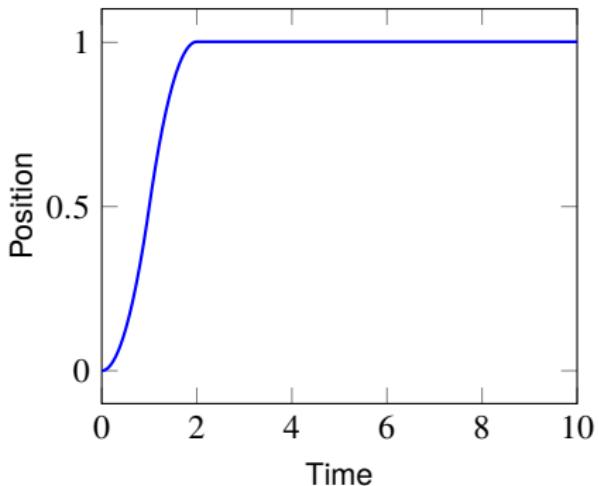
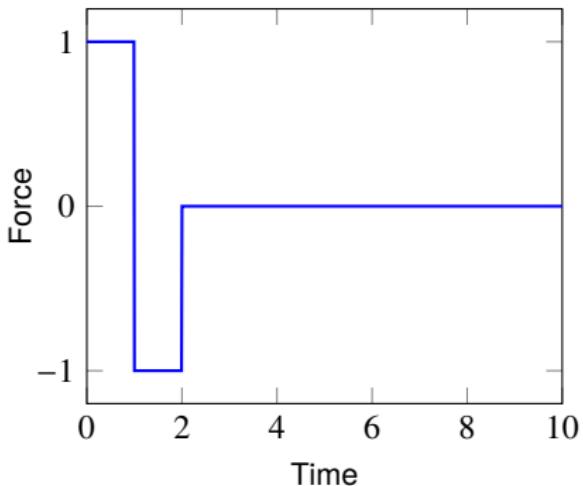
- ▶ unit mass on frictionless surface, initially at rest
- ▶ 10-vector  $f$  gives forces applied for one second each
- ▶ final velocity and position are

$$v^{\text{fin}} = f_1 + f_2 + \cdots + f_{10}$$

$$p^{\text{fin}} = (19/2)f_1 + (17/2)f_2 + \cdots + (1/2)f_{10}$$

- ▶ let's find  $f$  for which  $v^{\text{fin}} = 0, p^{\text{fin}} = 1$
- ▶  $f^{\text{bb}} = (1, -1, 0, \dots, 0)$  works (called 'bang-bang')

## Bang-bang force sequence



## Least norm force sequence

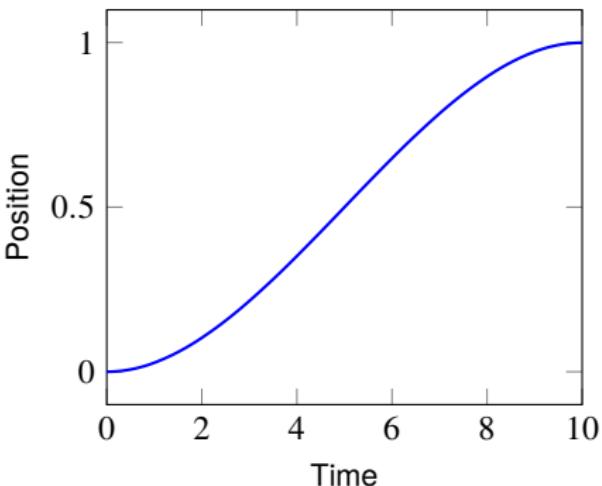
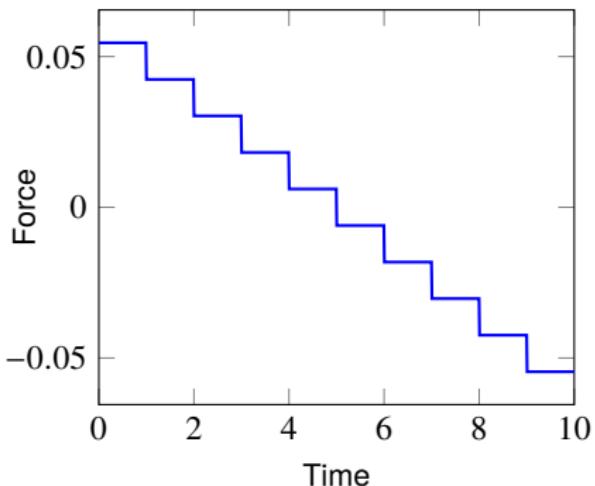
- ▶ let's find least-norm  $f$  that satisfies  $p^{\text{fin}} = 1$ ,  $v^{\text{fin}} = 0$
- ▶ least-norm problem:

$$\begin{array}{ll}\text{minimize} & \|f\|^2 \\ \text{subject to} & \left[ \begin{array}{ccccc} 1 & 1 & \cdots & 1 & 1 \\ 19/2 & 17/2 & \cdots & 3/2 & 1/2 \end{array} \right] f = \left[ \begin{array}{c} 0 \\ 1 \end{array} \right]\end{array}$$

with variable  $f$

- ▶ solution  $f^{\text{ln}}$  satisfies  $\|f^{\text{ln}}\|^2 = 0.0121$  (compare to  $\|f^{\text{bb}}\|^2 = 2$ )

## Least norm force sequence



## Outline

Linearly constrained least squares

Least norm problem

Solving the constrained least squares problem

## Optimality conditions via calculus

to solve constrained optimization problem

$$\begin{array}{ll}\text{minimize} & f(x) = \|Ax - b\|^2 \\ \text{subject to} & c_i^T x = d_i, \quad i = 1, \dots, p\end{array}$$

1. form *Lagrangian* function, with *Lagrange multipliers*  $z_1, \dots, z_p$

$$L(x, z) = f(x) + z_1(c_1^T x - d_1) + \dots + z_p(c_p^T x - d_p)$$

2. optimality conditions are

$$\frac{\partial L}{\partial x_i}(\hat{x}, z) = 0, \quad i = 1, \dots, n, \quad \frac{\partial L}{\partial z_i}(\hat{x}, z) = 0, \quad i = 1, \dots, p$$

## Optimality conditions via calculus

- ▶  $\frac{\partial L}{\partial z_i}(\hat{x}, z) = c_i^T \hat{x} - d_i = 0$ , which we already knew
- ▶ first  $n$  equations are more interesting:

$$\frac{\partial L}{\partial x_i}(\hat{x}, z) = 2 \sum_{j=1}^n (A^T A)_{ij} \hat{x}_j - 2(A^T b)_i + \sum_{j=1}^p z_j c_{ji} = 0$$

- ▶ in matrix-vector form:  $2(A^T A)\hat{x} - 2A^T b + C^T z = 0$
- ▶ put together with  $C\hat{x} = d$  to get *Karush–Kuhn–Tucker (KKT) conditions*

$$\begin{bmatrix} 2A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} 2A^T b \\ d \end{bmatrix}$$

a square set of  $n + p$  linear equations in variables  $\hat{x}, z$

- ▶ KKT equations are extension of normal equations to CLS

## Solution of constrained least squares problem

- ▶ assuming the KKT matrix is invertible, we have

$$\begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} 2A^T A & C^T \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2A^T b \\ d \end{bmatrix}$$

- ▶ KKT matrix is invertible if and only if

*C has linearly independent rows,  $\begin{bmatrix} A \\ C \end{bmatrix}$  has linearly independent columns*

- ▶ implies  $m + p \geq n, p \leq n$
- ▶ can compute  $\hat{x}$  in  $2mn^2 + 2(n + p)^3$  flops; order is  $n^3$  flops

## Direct verification of solution

- ▶ to show that  $\hat{x}$  is solution, suppose  $x$  satisfies  $Cx = d$
- ▶ then

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

- ▶ expand last term, using  $2A^T(A\hat{x} - b) = -C^T z$ ,  $Cx = C\hat{x} = d$ :

$$\begin{aligned}2(Ax - A\hat{x})^T(A\hat{x} - b) &= 2(x - \hat{x})^T A^T(A\hat{x} - b) \\ &= -(x - \hat{x})^T C^T z \\ &= -(C(x - \hat{x}))^T z \\ &= 0\end{aligned}$$

- ▶ so  $\|Ax - b\|^2 = \|A(x - \hat{x})\|^2 + \|A\hat{x} - b\|^2 \geq \|A\hat{x} - b\|^2$
- ▶ and we conclude  $\hat{x}$  is solution

## Solution of least-norm problem

- ▶ least-norm problem: minimize  $\|x\|^2$  subject to  $Cx = d$
- ▶ matrix  $\begin{bmatrix} I \\ C \end{bmatrix}$  always has independent columns
- ▶ we assume that  $C$  has independent rows
- ▶ optimality condition reduces to

$$\begin{bmatrix} 2I & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ d \end{bmatrix}$$

- ▶ so  $\hat{x} = -(1/2)C^Tz$ ; second equation is then  $-(1/2)CC^Tz = d$
- ▶ plug  $z = -2(CC^T)^{-1}d$  into first equation to get

$$\hat{x} = C^T(CC^T)^{-1}d = C^\dagger d$$

where  $C^\dagger$  is (our old friend) the pseudo-inverse

so when  $C$  has linearly independent rows:

- ▶  $C^\dagger$  is a right inverse of  $C$
- ▶ so for any  $d$ ,  $\hat{x} = C^\dagger d$  satisfies  $C\hat{x} = d$
- ▶ and we now know:  $\hat{x}$  is the *smallest* solution of  $Cx = d$

## 17. Constrained least squares applications

# Outline

Portfolio optimization

Linear quadratic control

Linear quadratic state estimation

## Portfolio allocation weights

- ▶ we invest a total of  $V$  dollars in  $n$  different *assets* (stocks, bonds, ...) over some period (one day, week, month, ...)
- ▶ can include *short positions*, assets you borrow and sell at the beginning, but must return to the borrower at the end of the period
- ▶ *portfolio allocation weight vector*  $w$  gives the fraction of our total portfolio value held in each asset
- ▶  $Vw_j$  is the dollar value of asset  $j$  you hold
- ▶  $\mathbf{1}^T w = 1$ , with negative  $w_i$  meaning a short position
- ▶  $w = (-0.2, 0.0, 1.2)$  means we take a short position of  $0.2V$  in asset 1, don't hold any of asset 2, and hold  $1.2V$  in asset 3

## Leverage, long-only portfolios, and cash

- ▶ leverage is  $L = |w_1| + \cdots + |w_n|$   
 $((L - 1)/2$  is also sometimes used)
- ▶  $L = 1$  when all weights are nonnegative ('long only portfolio')
- ▶  $w = \mathbf{1}/n$  is called the *uniform portfolio*
- ▶ we often assume asset  $n$  is 'risk-free' (or cash or T-bills)
- ▶ so  $w = e_n$  means the portfolio is all cash

## Return over a period

- ▶  $\tilde{r}_j$  is the *return* of asset  $j$  over the period
- ▶  $\tilde{r}_j$  is the fractional increase in price or value (decrease if negative)
- ▶ often expressed as a percentage, like +1.1% or -2.3%
- ▶ full *portfolio return* is

$$\frac{V^+ - V}{V} = \tilde{r}^T w$$

where  $V^+$  is the portfolio value at the end of the period

- ▶ if you hold portfolio for  $t$  periods with returns  $r_1, \dots, r_t$  value is

$$V_{t+1} = V_1(1 + r_1)(1 + r_2) \cdots (1 + r_t)$$

- ▶ portfolio value versus time traditionally plotted using  $V_1 = \$10000$

## Return matrix

- ▶ hold portfolio with weights  $w$  over  $T$  periods
- ▶ define  $T \times n$  (asset) *return matrix*, with  $R_{tj}$  the return of asset  $j$  in period  $t$
- ▶ row  $t$  of  $R$  is  $\tilde{r}_t^T$ , where  $\tilde{r}_t$  is the asset return vector over period  $t$
- ▶ column  $j$  of  $R$  is time series of asset  $j$  returns
- ▶ portfolio returns vector (time series) is  $T$ -vector  $r = Rw$
- ▶ if last asset is risk-free, the last column of  $R$  is  $\mu^{\text{rf}}\mathbf{1}$ , where  $\mu^{\text{rf}}$  is the risk-free per-period interest rate

## Portfolio return and risk

- ▶  $r$  is time series (vector) of portfolio returns
- ▶ *average return* or just *return* is  $\text{avg}(r)$
- ▶ *risk* is  $\text{std}(r)$
- ▶ these are the per-period return and risk
- ▶ for small per-period returns we have

$$\begin{aligned}V_{T+1} &= V_1(1 + r_1) \cdots (1 + r_T) \\&\approx V_1 + V_1(r_1 + \cdots + r_T) \\&= V_1 + T \text{avg}(r)V_1\end{aligned}$$

- ▶ so return approximates the average per-period increase in portfolio value

## Annualized return and risk

- ▶ mean return and risk are often expressed in *annualized form* (*i.e.*, per year)
- ▶ if there are  $P$  trading periods per year

$$\text{annualized return} = P \mathbf{avg}(r), \quad \text{annualized risk} = \sqrt{P} \mathbf{std}(r)$$

(the squareroot in risk annualization comes from the assumption that the fluctuations in return around the mean are independent)

- ▶ if returns are daily, with 250 trading days in a year

$$\text{annualized return} = 250 \mathbf{avg}(r), \quad \text{annualized risk} = \sqrt{250} \mathbf{std}(r)$$

## Portfolio optimization

- ▶ how should we choose the portfolio weight vector  $w$ ?
- ▶ we want high (mean) portfolio return, low portfolio risk
- ▶ we know past *realized asset returns* but not future ones
- ▶ we will choose  $w$  that would have worked well on past returns
- ▶ ... and hope it will work well going forward (just like data fitting)

## Portfolio optimization

$$\begin{aligned} \text{minimize} \quad & \mathbf{std}(Rw)^2 = (1/T) \|Rw - \rho \mathbf{1}\|^2 \\ \text{subject to} \quad & \mathbf{1}^T w = 1 \\ & \mathbf{avg}(Rw) = \rho \end{aligned}$$

- ▶  $w$  is the weight vector we seek
- ▶  $R$  is the returns matrix for *past returns*
- ▶  $Rw$  is the (past) portfolio return time series
- ▶ require mean (past) return  $\rho$
- ▶ we minimize risk for specified value of return
- ▶ solutions  $w$  are *Pareto optimal*
  
- ▶ we are really asking what *would have been* the best constant allocation, had we known future returns

## Portfolio optimization via constrained least squares

$$\begin{aligned} & \text{minimize} && \|Rw - \rho\mathbf{1}\|^2 \\ & \text{subject to} && \begin{bmatrix} \mathbf{1}^T \\ \mu^T \end{bmatrix} w = \begin{bmatrix} 1 \\ \rho \end{bmatrix} \end{aligned}$$

- ▶  $\mu = R^T \mathbf{1}/T$  is  $n$ -vector of (past) asset returns
- ▶  $\rho$  is required (past) portfolio return
- ▶ an equality constrained least squares problem, with solution

$$\begin{bmatrix} w \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 2R^T R & \mathbf{1} & \mu \\ \mathbf{1}^T & 0 & 0 \\ \mu^T & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2\rho T \mu \\ 1 \\ \rho \end{bmatrix}$$

## Optimal portfolios

- ▶ perform significantly better than individual assets
- ▶ risk-return curve forms a straight line
- ▶ one end of the line is the risk-free asset
- ▶ *two-fund theorem*: optimal portfolio  $w$  is an affine function of  $\rho$

$$\begin{bmatrix} w \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 2R^T R & \mathbf{1} & \mu \\ \mathbf{1}^T & 0 & 0 \\ \mu^T & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} R^T \mathbf{1} \\ 1 \\ \rho T \end{bmatrix}$$

## The big assumption

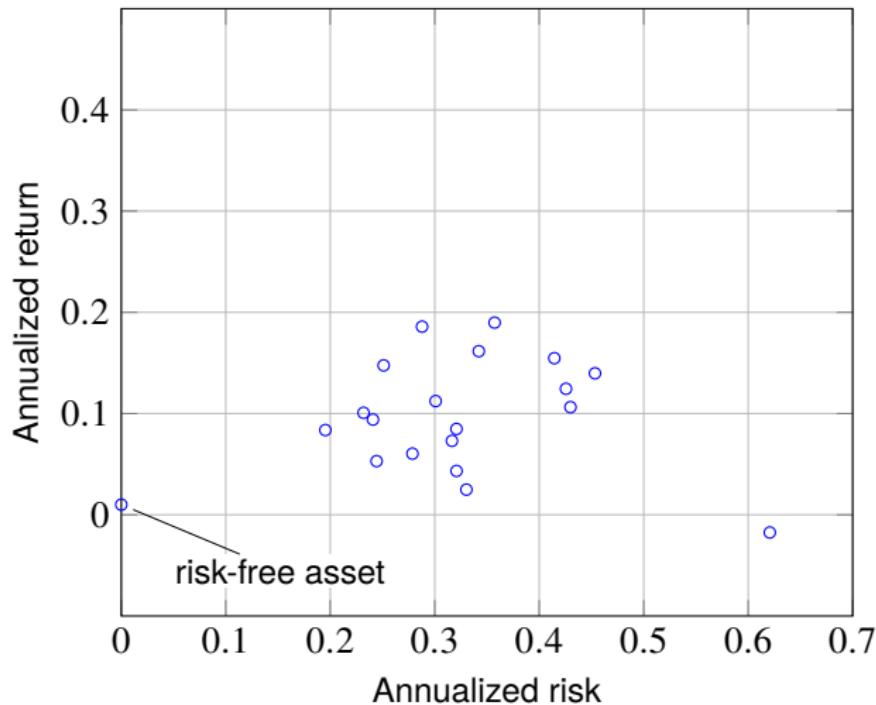
- ▶ now we make the big assumption (BA):

FUTURE RETURNS WILL LOOK SOMETHING LIKE PAST ONES

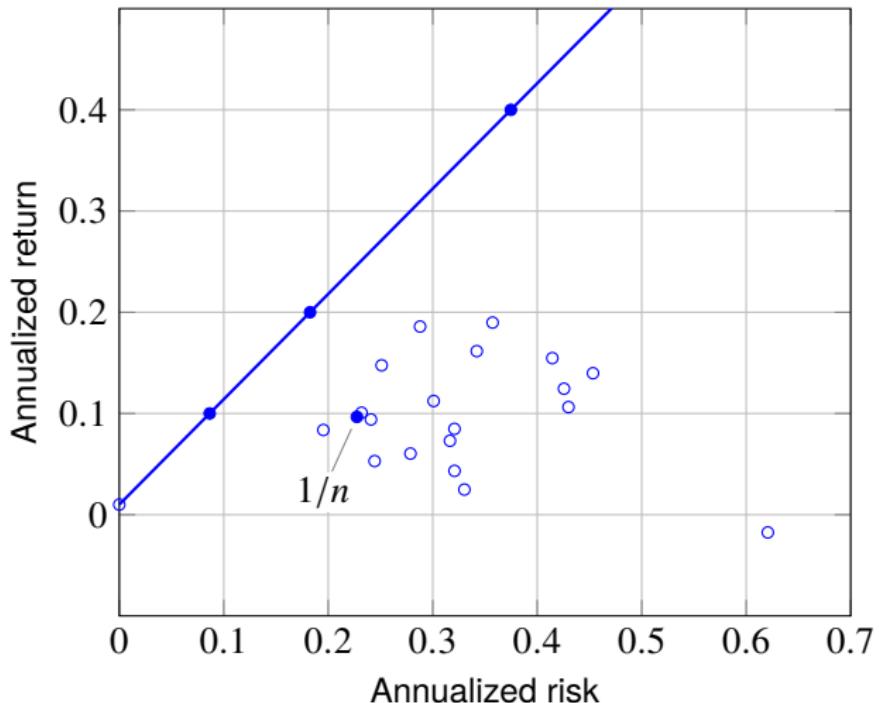
- you are warned this is false, every time you invest
- it is often reasonably true
- in periods of ‘market shift’ it’s much less true
- ▶ if BA holds (even approximately), then a good weight vector for past (realized) returns should be good for future (unknown) returns
- ▶ for example:
  - choose  $w$  based on last 2 years of returns
  - then use  $w$  for next 6 months

## Example

20 assets over 2000 days



## Pareto optimal portfolios

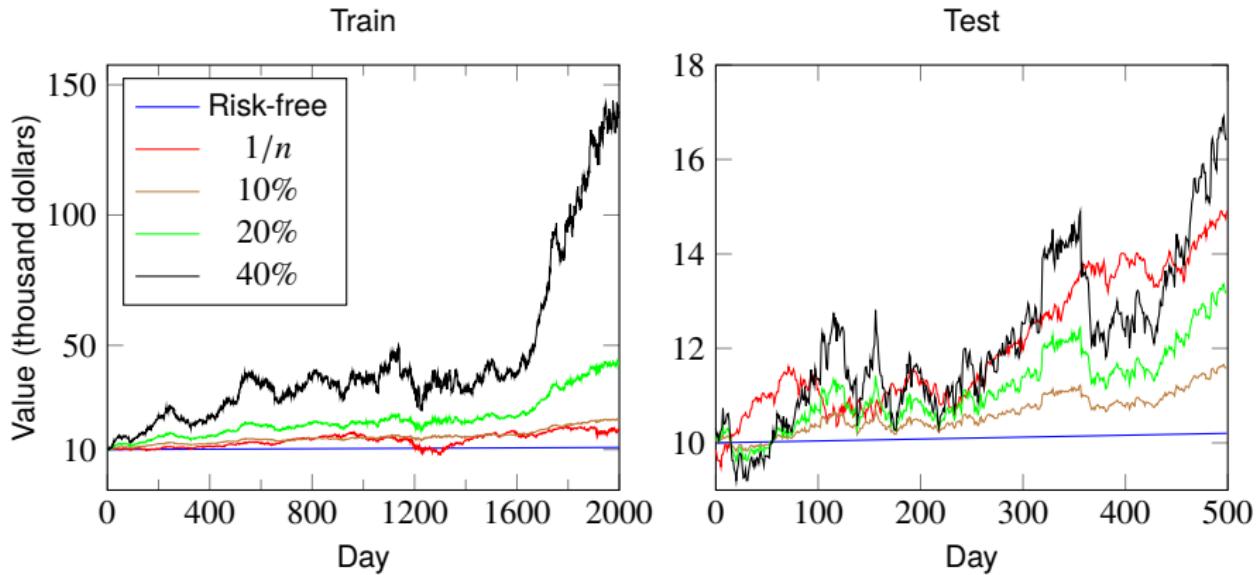


## Five portfolios

Portfolio	Return		Risk		
	Train	Test	Train	Test	Leverage
risk-free	0.01	0.01	0.00	0.00	1.00
$\rho = 10\%$	0.10	0.08	0.09	0.07	1.96
$\rho = 20\%$	0.20	0.15	0.18	0.15	3.03
$\rho = 40\%$	0.40	0.30	0.38	0.31	5.48
1/n (uniform weights)	0.10	0.21	0.23	0.13	1.00

- ▶ train period of 2000 days used to compute optimal portfolio
- ▶ test period is different 500-day period

## Total portfolio value



# Outline

Portfolio optimization

Linear quadratic control

Linear quadratic state estimation

## Linear dynamical system

$$x_{t+1} = A_t x_t + B_t u_t, \quad y_t = C_t x_t, \quad t = 1, 2, \dots$$

- ▶  $n$ -vector  $x_t$  is *state* at time  $t$
- ▶  $m$ -vector  $u_t$  is *input* at time  $t$
- ▶  $p$ -vector  $y_t$  is *output* at time  $t$
- ▶  $n \times n$  matrix  $A_t$  is *dynamics matrix*
- ▶  $n \times m$  matrix  $B_t$  is *input matrix*
- ▶  $p \times n$  matrix  $C_t$  is *output matrix*
- ▶  $x_t, u_t, y_t$  often represent deviations from a standard operating condition

## Linear quadratic control

$$\begin{array}{ll}\text{minimize} & J_{\text{output}} + \rho J_{\text{input}} \\ \text{subject to} & x_{t+1} = A_t x_t + B_t u_t, \quad t = 1, \dots, T-1 \\ & x_1 = x^{\text{init}}, \quad x_T = x^{\text{des}}\end{array}$$

- ▶ variables are state sequence  $x_1, \dots, x_T$  and input sequence  $u_1, \dots, u_{T-1}$
- ▶ two objectives are quadratic functions of state and input sequences:

$$\begin{aligned}J_{\text{output}} &= \|y_1\|^2 + \dots + \|y_T\|^2 = \|C_1 x_1\|^2 + \dots + \|C_T x_T\|^2 \\J_{\text{input}} &= \|u_1\|^2 + \dots + \|u_{T-1}\|^2\end{aligned}$$

- ▶ first constraint imposes the linear dynamics equations
- ▶ second set of constraints specifies the initial and final state
- ▶  $\rho$  is positive parameter used to trade off the two objectives

## Constrained least squares formulation

$$\begin{aligned} & \text{minimize} && \|C_1x_1\|^2 + \dots + \|C_Tx_T\|^2 + \rho\|u_1\|^2 + \dots + \|u_{T-1}\|^2 \\ & \text{subject to} && x_{t+1} = A_t x_t + B_t u_t, \quad t = 1, \dots, T-1 \\ & && x_1 = x^{\text{init}}, \quad x_T = x^{\text{des}} \end{aligned}$$

- ▶ can be written as

$$\begin{aligned} & \text{minimize} && \|\tilde{A}z - \tilde{b}\|^2 \\ & \text{subject to} && \tilde{C}z = \tilde{d} \end{aligned}$$

- ▶ vector  $z$  contains the  $Tn + (T-1)m$  variables:

$$z = (x_1, \dots, x_T, u_1, \dots, u_{T-1})$$

## Constrained least squares formulation

$$\tilde{A} = \left[ \begin{array}{ccc|ccc} C_1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & C_T & 0 & \cdots & 0 \\ \hline 0 & \cdots & 0 & \sqrt{\rho}I & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & \sqrt{\rho}I \end{array} \right], \quad \tilde{b} = 0$$

$$\tilde{C} = \left[ \begin{array}{cccccc|cccccc} A_1 & -I & 0 & \cdots & 0 & 0 & B_1 & 0 & \cdots & 0 \\ 0 & A_2 & -I & \cdots & 0 & 0 & 0 & B_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_{T-1} & -I & 0 & 0 & \cdots & B_{T-1} \\ \hline I & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & I & 0 & 0 & \cdots & 0 \end{array} \right], \quad \tilde{d} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \hline x^{\text{init}} \\ x^{\text{des}} \end{bmatrix}$$

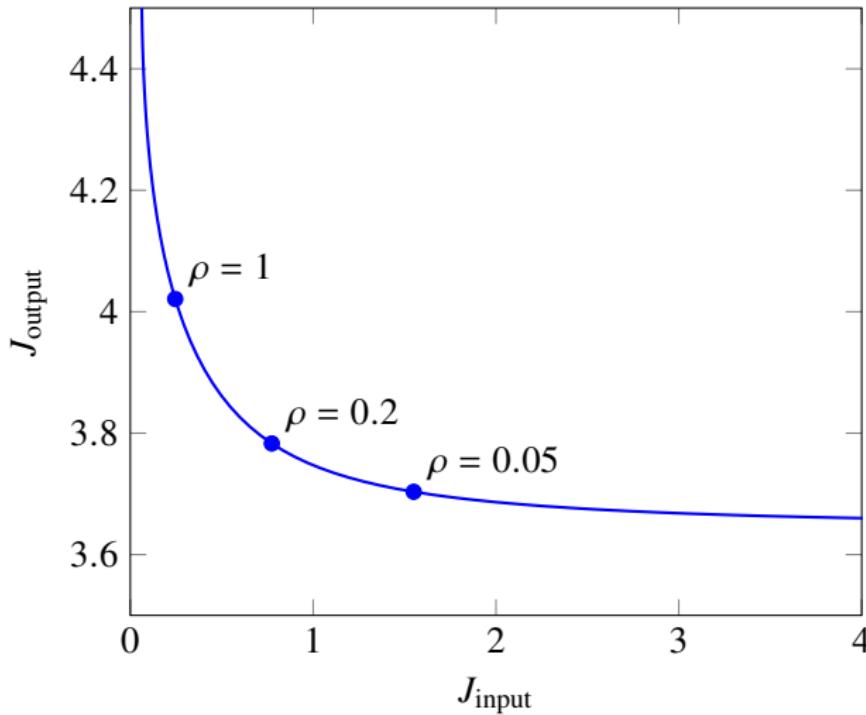
## Example

- ▶ time-invariant system: system matrices are constant

$$A = \begin{bmatrix} 0.855 & 1.161 & 0.667 \\ 0.015 & 1.073 & 0.053 \\ -0.084 & 0.059 & 1.022 \end{bmatrix}, \quad B = \begin{bmatrix} -0.076 \\ -0.139 \\ 0.342 \end{bmatrix},$$
$$C = \begin{bmatrix} 0.218 & -3.597 & -1.683 \end{bmatrix}$$

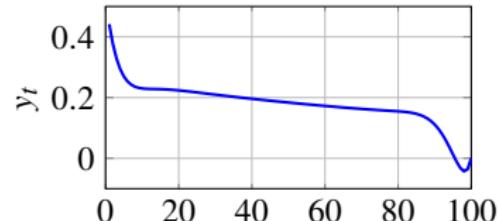
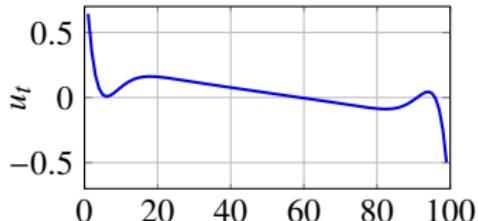
- ▶ initial condition  $x^{\text{init}} = (0.496, -0.745, 1.394)$
- ▶ target or desired final state  $x^{\text{des}} = 0$
- ▶  $T = 100$

## Optimal trade-off curve

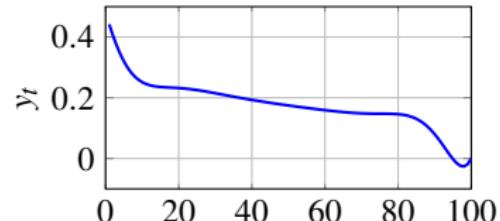
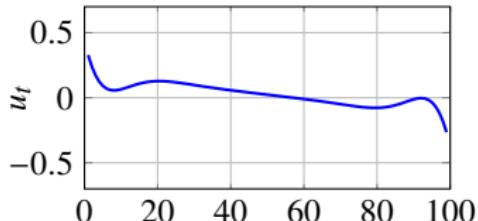


## Three points on the trade-off curve

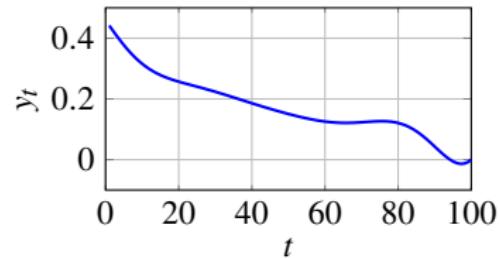
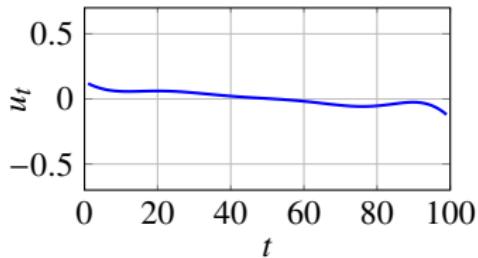
$\rho = 0.05$



$\rho = 0.2$



$\rho = 1$



## Linear state feedback control

- ▶ linear state feedback control uses the input

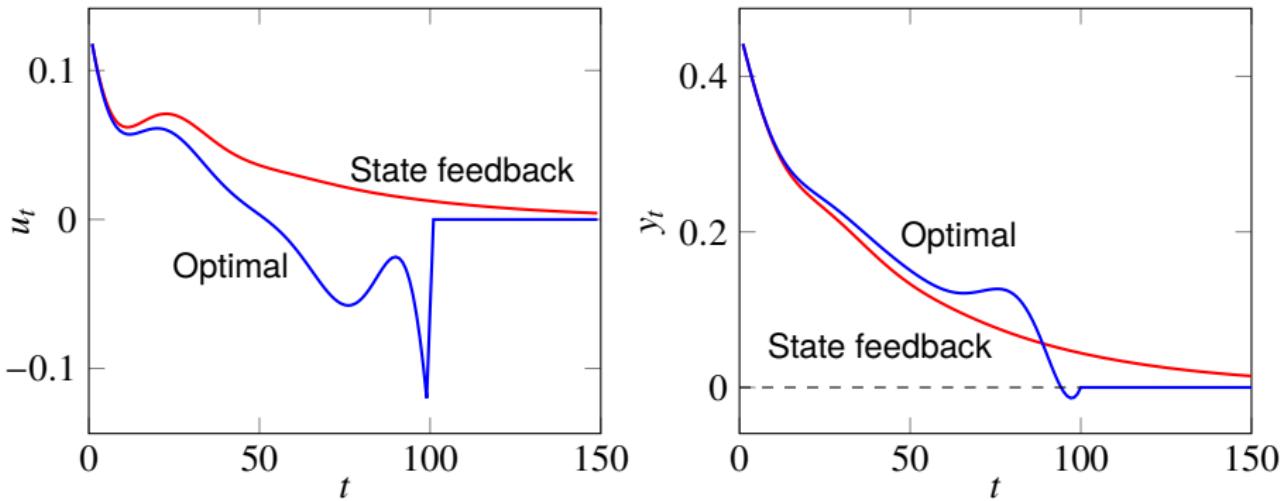
$$u_t = Kx_t, \quad t = 1, 2, \dots$$

- ▶  $K$  is *state feedback gain matrix*
- ▶ widely used, especially when  $x_t$  should converge to zero,  $T$  is not specified
- ▶ one choice for  $K$ : solve linear quadratic control problem with  $x^{\text{des}} = 0$
- ▶ solution  $u_t$  is a linear function of  $x^{\text{init}}$ , hence  $u_1$  can be written as

$$u_1 = Kx^{\text{init}}$$

- ▶ columns of  $K$  can be found by computing  $u_1$  for  $x^{\text{init}} = e_1, \dots, e_n$
- ▶ use this  $K$  as state feedback gain matrix

## Example



- ▶ system matrices of previous example
- ▶ blue curve uses optimal linear quadratic control for  $T = 100$
- ▶ red curve uses simple linear state feedback  $u_t = Kx_t$

# Outline

Portfolio optimization

Linear quadratic control

Linear quadratic state estimation

## State estimation

- ▶ linear dynamical system model:

$$x_{t+1} = A_t x_t + B_t w_t, \quad y_t = C_t x_t + v_t, \quad t = 1, 2, \dots$$

- ▶  $x_t$  is *state* ( $n$ -vector)
- ▶  $y_t$  is *measurement* ( $p$ -vector)
- ▶  $w_t$  is *input* or *process noise* ( $m$ -vector)
- ▶  $v_t$  is *measurement noise* or *measurement residual* ( $p$ -vector)
- ▶ we know  $A_t, B_t, C_t$ , and measurements  $y_1, \dots, y_T$
- ▶  $w_t, v_t$  are unknown, but assumed small
- ▶ *state estimation*: estimate/guess  $x_1, \dots, x_T$

## Least squares state estimation

$$\begin{aligned} & \text{minimize} && J_{\text{meas}} + \lambda J_{\text{proc}} \\ & \text{subject to} && x_{t+1} = A_t x_t + B_t w_t, \quad t = 1, \dots, T-1 \end{aligned}$$

- ▶ variables: states  $x_1, \dots, x_T$  and input noise  $w_1, \dots, w_{T-1}$
- ▶ primary objective  $J_{\text{meas}}$  is sum of squares of measurement residuals:

$$J_{\text{meas}} = \|C_1 x_1 - y_1\|^2 + \dots + \|C_T x_T - y_T\|^2$$

- ▶ secondary objective  $J_{\text{proc}}$  is sum of squares of process noise

$$J_{\text{proc}} = \|w_1\|^2 + \dots + \|w_{T-1}\|^2$$

- ▶  $\lambda > 0$  is a parameter, trades off measurement and process errors

## Constrained least squares formulation

$$\begin{array}{ll}\text{minimize} & \|C_1x_1 - y_1\|^2 + \cdots + \|C_Tx_T - y_T\|^2 + \lambda(\|w_1\|^2 + \cdots + \|w_{T-1}\|^2) \\ \text{subject to} & x_{t+1} = A_t x_t + B_t w_t, \quad t = 1, \dots, T-1\end{array}$$

- ▶ can be written as

$$\begin{array}{ll}\text{minimize} & \|\tilde{A}z - \tilde{b}\|^2 \\ \text{subject to} & \tilde{C}z = \tilde{d}\end{array}$$

- ▶ vector  $z$  contains the  $Tn + (T - 1)m$  variables:

$$z = (x_1, \dots, x_T, w_1, \dots, w_{T-1})$$

## Constrained least squares formulation

$$\tilde{A} = \left[ \begin{array}{cccc|ccc} C_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & C_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & C_T & 0 & \cdots & 0 \\ \hline 0 & 0 & \cdots & 0 & \sqrt{\lambda}I & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & \sqrt{\lambda}I \end{array} \right], \quad \tilde{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\tilde{C} = \left[ \begin{array}{ccccc|ccccc} A_1 & -I & 0 & \cdots & 0 & 0 & B_1 & 0 & \cdots & 0 \\ 0 & A_2 & -I & \cdots & 0 & 0 & 0 & B_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & A_{T-1} & -I & 0 & 0 & \cdots & B_{T-1} \end{array} \right], \quad \tilde{d} = 0$$

## Missing measurements

- ▶ suppose we have measurements  $y_t$  for  $t \in \mathcal{T}$ , a subset of  $\{1, \dots, T\}$
- ▶ measurements for  $t \in \mathcal{T}$  are missing
- ▶ to estimate states, use same formulation but with

$$J_{\text{meas}} = \sum_{t \in \mathcal{T}} \|C_t x_t - y_t\|^2$$

- ▶ from estimated states  $\hat{x}_t$ , can estimate missing measurements

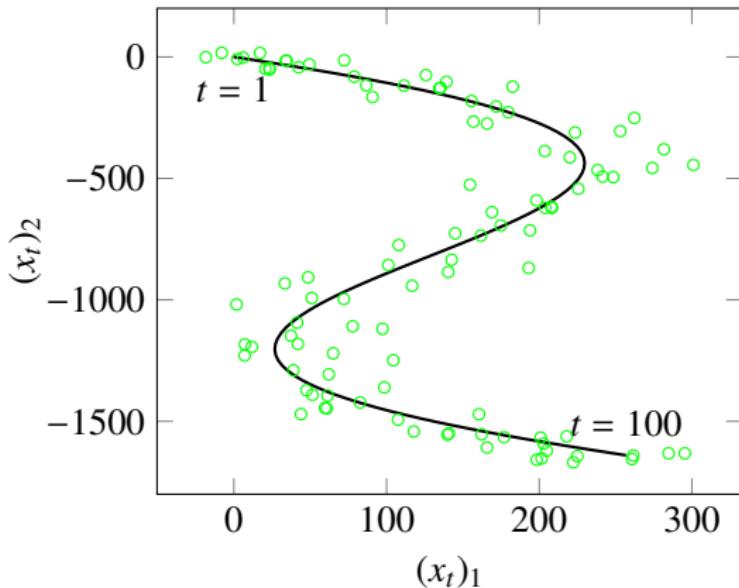
$$\hat{y}_t = C_t \hat{x}_t, \quad t \notin \mathcal{T}$$

## Example

$$A_t = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B_t = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad C_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

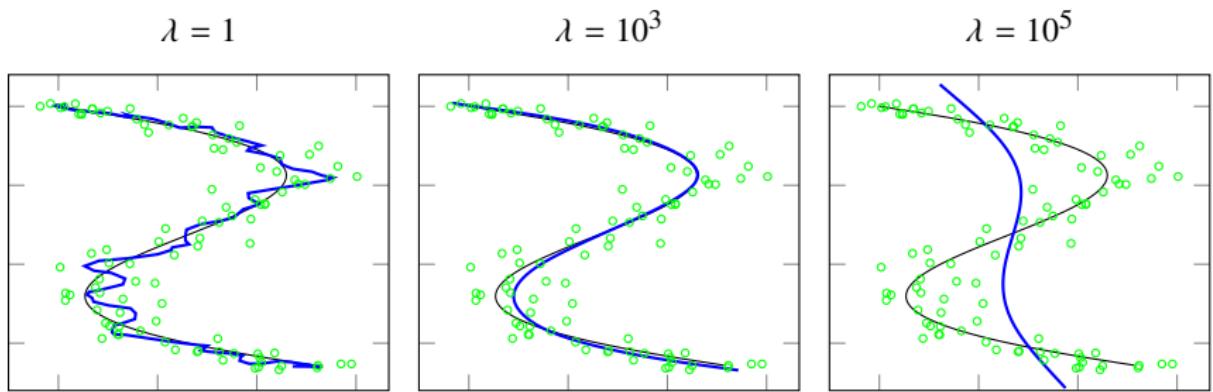
- ▶ simple model of mass moving in a 2-D plane
- ▶  $x_t = (p_t, z_t)$ : 2-vector  $p_t$  is position, 2-vector  $z_t$  is the velocity
- ▶  $y_t = C_t x_t + w_t$  is noisy measurement of position
- ▶  $T = 100$

## Measurements and true positions



- ▶ solid line is exact position  $C_t x_t$
- ▶ 100 noisy measurements  $y_t$  shown as circles

## Position estimates

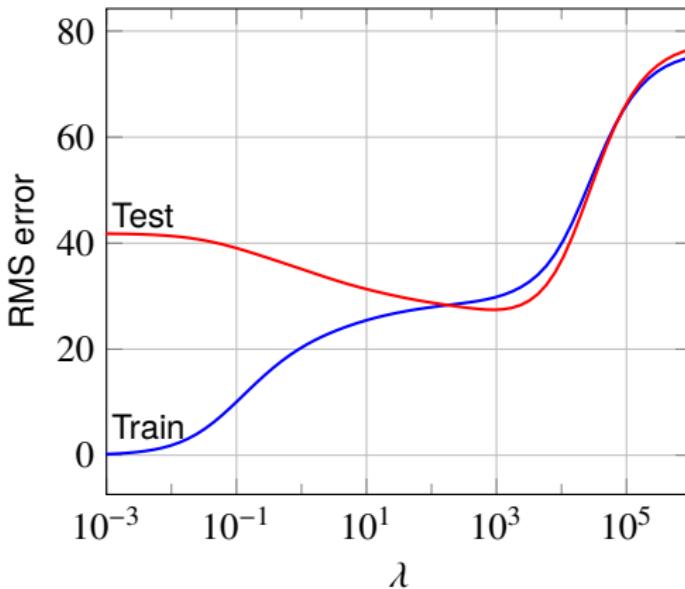


blue lines show position estimates for three values of  $\lambda$

## Cross-validation

- ▶ randomly remove 20% (say) of the measurements and use as test set
- ▶ for many values of  $\lambda$ , estimate states using other (*training*) measurements
- ▶ for each  $\lambda$ , evaluate RMS measurement residuals on test set
- ▶ choose  $\lambda$  to (approximately) minimize the RMS test residuals

## Example



- ▶ cross-validation method applied to previous example
- ▶ remove 20 of the 100 measurements
- ▶ suggests using  $\lambda \approx 10^3$

## 18. Nonlinear least squares

# Outline

Nonlinear equations and least squares

Examples

Levenberg–Marquardt algorithm

Nonlinear model fitting

Nonlinear least squares classification

## Nonlinear equations

- ▶ set of  $m$  nonlinear equations in  $n$  unknowns  $x_1, \dots, x_n$ :

$$f_i(x_1, \dots, x_n) = 0, \quad i = 1, \dots, m$$

- ▶  $f_i(x) = 0$  is the  $i$ th equation;  $f_i(x)$  is the  $i$ th residual
- ▶  $n$ -vector of unknowns  $x = (x_1, \dots, x_n)$
- ▶ write as vector equation  $f(x) = 0$  where  $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ ,

$$f(x) = (f_1(x), \dots, f_m(x))$$

- ▶ when  $f$  is affine, reduces to set of  $m$  linear equations
- ▶ over-determined if  $m > n$ , under-determined if  $m < n$ , square if  $m = n$

## Nonlinear least squares

- ▶ find  $\hat{x}$  that minimizes

$$\|f(x)\|^2 = f_1(x)^2 + \cdots + f_m(x)^2$$

- ▶ includes problem of solving equations  $f(x) = 0$  as special case
- ▶ like (linear) least squares, super useful on its own

## Optimality condition

- ▶ optimality condition:  $\nabla \|f(\hat{x})\|^2 = 0$
- ▶ any optimal point satisfies this
- ▶ points can satisfy this and not be optimal
- ▶ can be expressed as  $2Df(\hat{x})^T f(\hat{x}) = 0$
- ▶  $Df(\hat{x})$  is the  $m \times n$  derivative or Jacobian matrix,

$$Df(\hat{x})_{ij} = \frac{\partial f_i}{\partial x_j}(\hat{x}), \quad i = 1, \dots, m, \quad j = 1, \dots, n$$

- ▶ optimality condition reduces to normal equations when  $f$  is affine

## Difficulty of solving nonlinear least squares problem

- ▶ solving nonlinear equations or nonlinear least squares problem is (in general) *much harder* than solving linear equations
- ▶ even determining if a solution exists is hard
- ▶ so we will use *heuristic* algorithms:
  - not guaranteed to always work
  - but often work well in practice(like  $k$ -means)

# Outline

Nonlinear equations and least squares

## Examples

Levenberg–Marquardt algorithm

Nonlinear model fitting

Nonlinear least squares classification

## Computing equilibrium points

- ▶ *equilibrium prices*: find  $n$ -vector of prices  $p$  for which  $S(p) = D(p)$ 
  - $S(p)$  is supply of  $n$  goods as function of prices
  - $D(p)$  is demand for  $n$  goods as function of prices
  - take  $f(p) = S(p) - D(p)$
- ▶ *chemical equilibrium*: find  $n$ -vector of concentrations  $c$  so  $C(c) = G(c)$ 
  - $C(c)$  is consumption of species as function of  $c$
  - $G(c)$  is generation of species as function of  $c$
  - take  $f(c) = C(c) - G(c)$

## Location from range measurements

- ▶ 3-vector  $x$  is position in 3-D, which we will estimate
- ▶ *range* measurements give (noisy) distance to known locations

$$\rho_i = \|x - a_i\| + v_i, \quad i = 1, \dots, m$$

$a_i$  are known locations,  $v_i$  are noises

- ▶ least squares location estimation: choose  $\hat{x}$  that minimizes

$$\sum_{i=1}^m (\|x - a_i\| - \rho_i)^2$$

- ▶ GPS works like this

# Outline

Nonlinear equations and least squares

Examples

Levenberg–Marquardt algorithm

Nonlinear model fitting

Nonlinear least squares classification

## The basic idea

- ▶ at any point  $z$  we can form the affine approximation

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

- ▶  $\hat{f}(x; z) \approx f(x)$  provided  $x$  is near  $z$
- ▶ we can minimize  $\|\hat{f}(x; z)\|^2$  using linear least squares
- ▶ we'll iterate, with  $z$  the current iterate

## Levenberg–Marquardt algorithm

- ▶ iterates  $x^{(1)}, x^{(2)}, \dots$
- ▶ at iteration  $k$ , form affine approximation of  $f$  at  $x^{(k)}$ :

$$\hat{f}(x; x^{(k)}) = f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})$$

- ▶ choose  $x^{(k+1)}$  as minimizer of

$$\|\hat{f}(x; x^{(k)})\|^2 + \lambda^{(k)} \|x - x^{(k)}\|^2 \quad (\text{where } \lambda^{(k)} > 0)$$

- ▶ we want  $\|\hat{f}(x; x^{(k)})\|^2$  small, but we don't want to move too far from  $x^{(k)}$ , where  $\hat{f}(x; x^{(k)}) \approx f(x)$  no longer holds

## Levenberg–Marquardt iteration

- $x^{(k+1)}$  is solution of least squares problem

$$\text{minimize} \quad \|f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})\|^2 + \lambda^{(k)} \|x - x^{(k)}\|^2$$

- solution is

$$x^{(k+1)} = x^{(k)} - \left( Df(x^{(k)})^T Df(x^{(k)}) + \lambda^{(k)} I \right)^{-1} Df(x^{(k)})^T f(x^{(k)})$$

- inverse always exists (since  $\lambda^{(k)} > 0$ )
- $x^{(k+1)} = x^{(k)}$  only if  $Df(x^{(k)})^T f(x^{(k)}) = 0$ , i.e., optimality condition holds

## Adjusting $\lambda^{(k)}$

idea:

- ▶ if  $\lambda^{(k)}$  is too big,  $x^{(k+1)}$  is too close to  $x^{(k)}$ , and progress is slow
- ▶ if too small,  $x^{(k+1)}$  may be far from  $x^{(k)}$  and affine approximation is poor

update mechanism:

- ▶ if  $\|f(x^{(k+1)})\|^2 < \|f(x^{(k)})\|^2$ , accept new  $x$  and reduce  $\lambda$

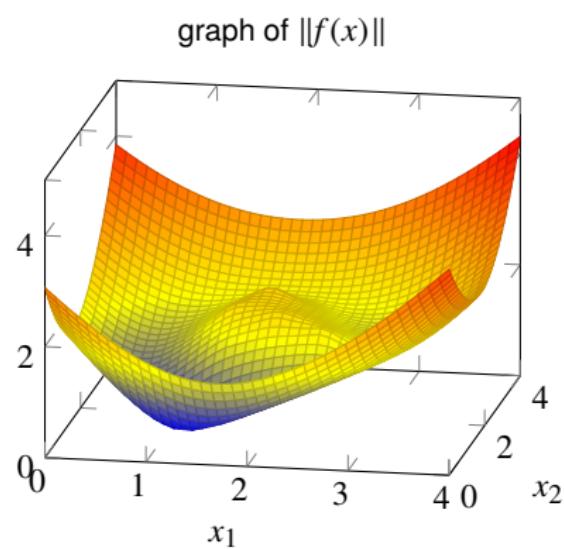
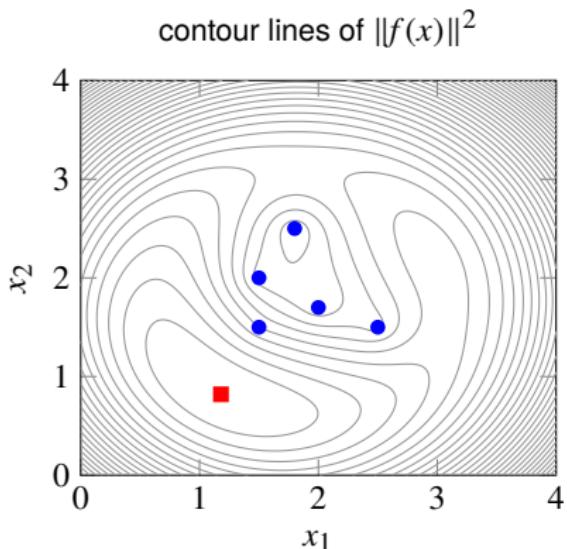
$$\lambda^{(k+1)} = 0.8\lambda^{(k)}$$

- ▶ otherwise, increase  $\lambda$  and do not update  $x$ :

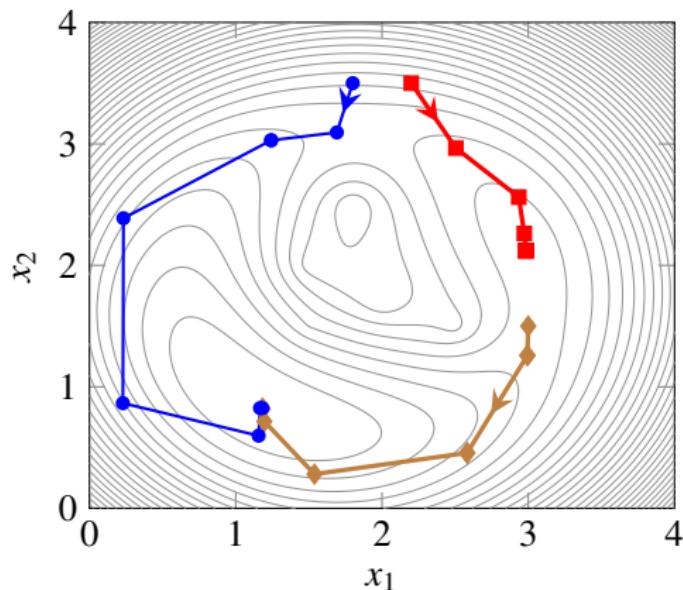
$$\lambda^{(k+1)} = 2\lambda^{(k)}, \quad x^{(k+1)} = x^{(k)}$$

## Example: Location from range measurements

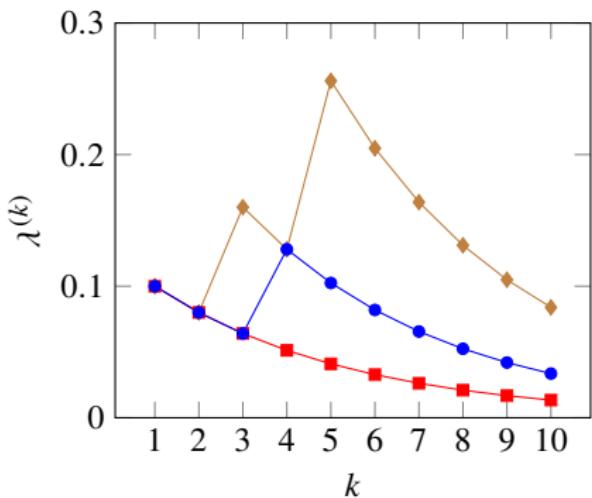
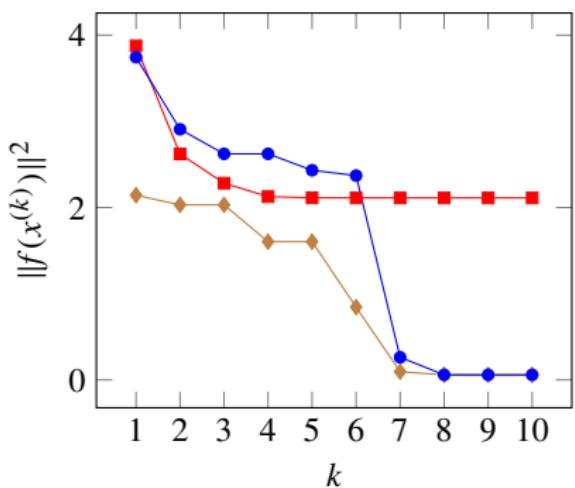
- ▶ range to 5 points (blue circles)
- ▶ red square shows  $\hat{x}$



## Levenberg–Marquardt from three initial points



## Levenberg–Marquardt from three initial points



# Outline

Nonlinear equations and least squares

Examples

Levenberg–Marquardt algorithm

Nonlinear model fitting

Nonlinear least squares classification

## Nonlinear model fitting

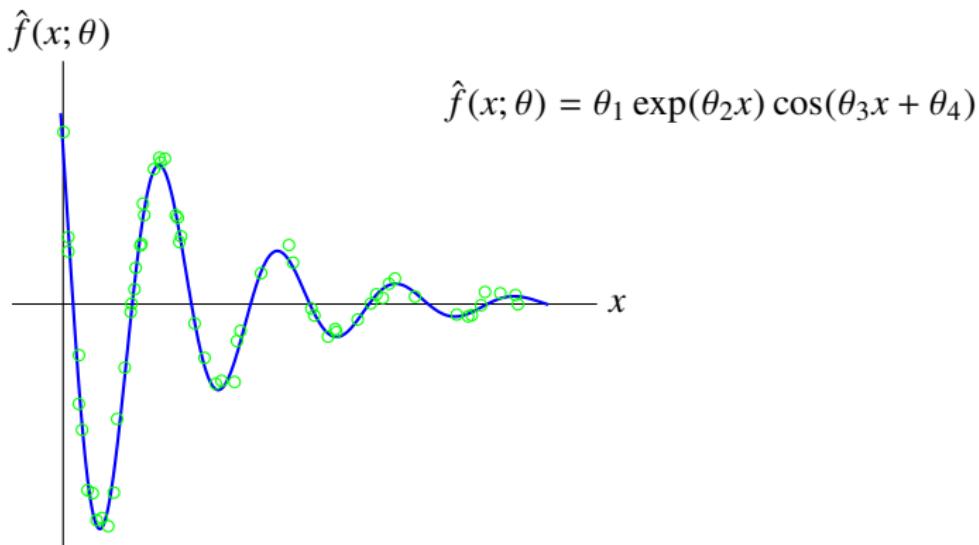
$$\text{minimize} \quad \sum_{i=1}^N (\hat{f}(x^{(i)}; \theta) - y^{(i)})^2$$

- ▶  $x^{(1)}, \dots, x^{(N)}$  are feature vectors
- ▶  $y^{(1)}, \dots, y^{(N)}$  are associated outcomes
- ▶ model  $\hat{f}(x; \theta)$  is parameterized by parameters  $\theta_1, \dots, \theta_p$
- ▶ this generalizes the *linear in parameters model*

$$\hat{f}(x; \theta) = \theta_1 f_1(x) + \dots + \theta_p f_p(x)$$

- ▶ here we allow  $\hat{f}(x, \theta)$  to be a nonlinear function of  $\theta$
- ▶ the minimization is over the model parameters  $\theta$

## Example

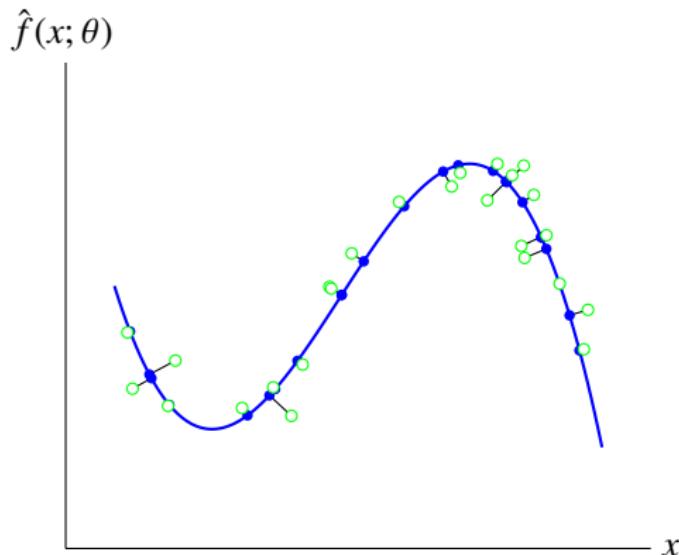


a nonlinear least squares problem with four variables  $\theta_1, \theta_2, \theta_3, \theta_4$ :

$$\text{minimize} \quad \sum_{i=1}^N \left( \theta_1 e^{\theta_2 x^{(i)}} \cos(\theta_3 x^{(i)} + \theta_4) - y^{(i)} \right)^2$$

## Orthogonal distance regression

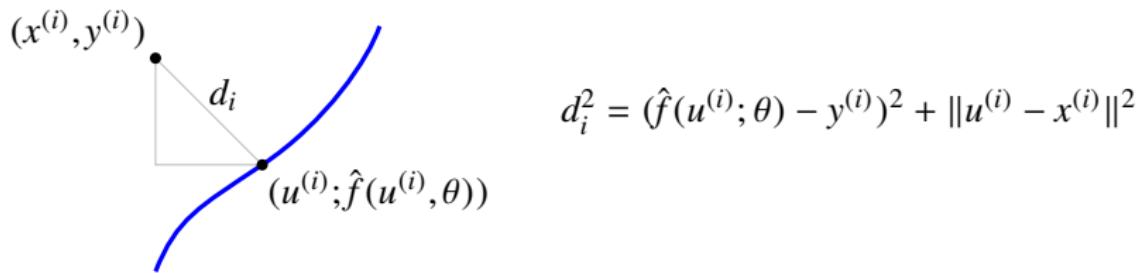
- ▶ to fit model, minimize sum square distance of data points to graph
- ▶ example: orthogonal distance regression to cubic polynomial



## 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 data 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  $u^{(1)}, \dots, u^{(N)}$  and  $\theta$  minimizes the sum square distance

# Outline

Nonlinear equations and least squares

Examples

Levenberg–Marquardt algorithm

Nonlinear model fitting

Nonlinear least squares classification

## Nonlinear least squares classification

linear least squares classifier:

- ▶ classifier is  $\hat{f}(x) = \text{sign}(\tilde{f}(x))$  where  $\tilde{f}(x) = \theta_1 f_1(x) + \dots + \theta_p f_p(x)$
- ▶  $\theta$  is chosen by minimizing  $\sum_{i=1}^N (\tilde{f}(x_i) - y_i)^2$  (plus optionally regularization)

nonlinear least squares classifier:

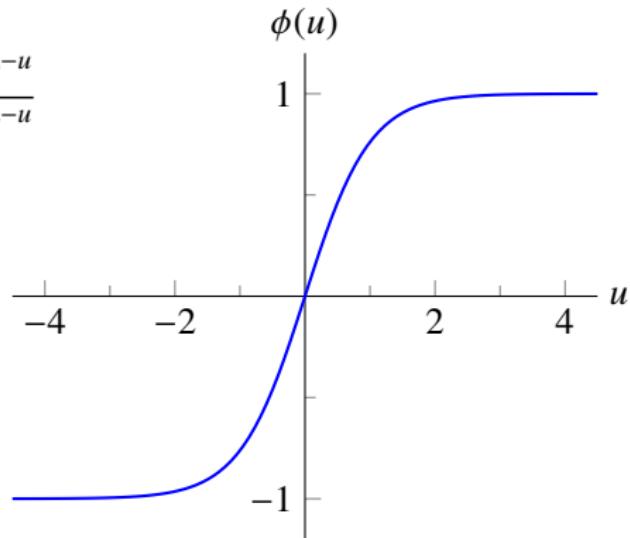
- ▶ choose  $\theta$  to minimize

$$\sum_{i=1}^N (\text{sign}(\tilde{f}(x_i)) - y_i)^2 = 4 \times \text{number of errors}$$

- ▶ replace **sign** function with smooth approximation  $\phi$ , e.g., sigmoid function
- ▶ use Levenberg–Marquardt to minimize  $\sum_{i=1}^N (\phi(\tilde{f}(x_i)) - y_i)^2$

## Sigmoid function

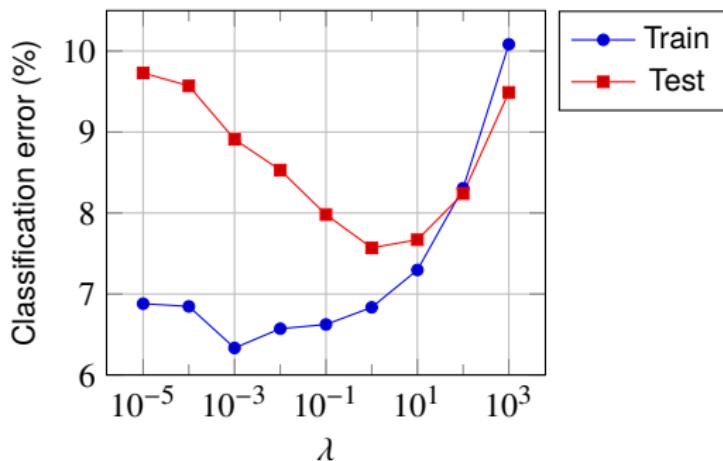
$$\phi(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$



## Example

- ▶ MNIST data set; feature vector  $x$  is 493-vector of pixel intensities
- ▶ nonlinear least squares 10-way multi-class classifier: 7.5% test error
- ▶ Boolean classifiers computed by solving nonlinear least squares problems

$$\text{minimize} \quad \sum_{i=1}^N (\phi((x^{(i)})^T \beta + v) - y^{(i)})^2 + \lambda \|\beta\|^2$$



## Feature engineering

- ▶ add 5000 random features as before
- ▶ test set error drops to 2%
- ▶ this matches human performance
- ▶ with more feature engineering, can substantially beat human performance

## 19. Constrained nonlinear least squares

# Outline

Constrained nonlinear least squares

Penalty method

Augmented Lagrangian method

Nonlinear control example

## Constrained nonlinear least squares

- ▶ add equality constraints to nonlinear least squares problem:

$$\begin{aligned} & \text{minimize} && f_1(x)^2 + \cdots + f_m(x)^2 \\ & \text{subject to} && g_1(x) = 0, \dots, g_p(x) = 0 \end{aligned}$$

- ▶  $f_i(x)$  is *i*th (scalar) *residual*;  $g_i(x) = 0$  is *i*th (scalar) equality constraint
- ▶ with vector notation  $f(x) = (f_1(x), \dots, f_m(x))$ ,  $g(x) = (g_1(x), \dots, g_p(x))$

$$\begin{aligned} & \text{minimize} && \|f(x)\|^2 \\ & \text{subject to} && g(x) = 0 \end{aligned}$$

- ▶  $x$  is *feasible* if it satisfies the constraints  $g(x) = 0$
- ▶  $\hat{x}$  is a solution if it is feasible and  $\|f(x)\|^2 \geq \|f(\hat{x})\|^2$  for all feasible  $x$
- ▶ problem is difficult to solve in general, but useful heuristics exist

## Lagrange multipliers

- ▶ the *Lagrangian* of the problem is the function

$$\begin{aligned} L(x, z) &= \|f(x)\|^2 + z_1 g_1(x) + \cdots + z_m g_m(x) \\ &= \|f(x)\|^2 + g(x)^T z \end{aligned}$$

- ▶  $p$ -vector  $z = (z_1, \dots, z_p)$  is vector of *Lagrange multipliers*
- ▶ method of Lagrange multipliers: if  $\hat{x}$  is a solution, then there exists  $\hat{z}$  with

$$\frac{\partial L}{\partial x_i}(\hat{x}, \hat{z}) = 0, \quad i = 1, \dots, n. \quad \frac{\partial L}{\partial z_i}(\hat{x}, \hat{z}) = 0, \quad i = 1, \dots, p$$

(provided the gradients  $\nabla g_1(\hat{x}), \dots, \nabla g_p(\hat{x})$  are linearly independent)

- ▶  $\hat{z}$  is called an *optimal Lagrange multiplier*

## Optimality condition

- ▶ gradient of Lagrangian with respect to  $x$  is

$$\nabla_x L(\hat{x}, \hat{z}) = 2Df(\hat{x})^T f(\hat{x}) + Dg(\hat{x})^T \hat{z}$$

- ▶ gradient with respect to  $z$  is

$$\nabla_z L(\hat{x}, \hat{z}) = g(\hat{x})$$

- ▶ optimality condition: if  $\hat{x}$  is optimal, then there exists  $\hat{z}$  such that

$$2Df(\hat{x})^T f(\hat{x}) + Dg(\hat{x})^T \hat{z} = 0, \quad g(\hat{x}) = 0$$

(provided the rows of  $Dg(\hat{x})$  are linearly independent)

- ▶ this condition is necessary for optimality but not sufficient

## Constrained (linear) least squares

- ▶ recall constrained least squares problem

$$\begin{array}{ll}\text{minimize} & \|Ax - b\|^2 \\ \text{subject to} & Cx = d\end{array}$$

- ▶ a special case of the nonlinear problem with  $f(x) = Ax - b$ ,  $g(x) = Cx - d$
- ▶ apply general optimality condition:

$$2Df(\hat{x})^T f(\hat{x}) + Dg(\hat{x})^T \hat{z} = 2A^T(A\hat{x} - b) + C^T\hat{z} = 0, \quad g(\hat{x}) = C\hat{x} - d = 0$$

- ▶ these are the KKT equations

$$\left[ \begin{array}{cc} 2A^T A & C^T \\ C & 0 \end{array} \right] \left[ \begin{array}{c} \hat{x} \\ \hat{z} \end{array} \right] = \left[ \begin{array}{c} 2A^T b \\ d \end{array} \right]$$

# Outline

Constrained nonlinear least squares

Penalty method

Augmented Lagrangian method

Nonlinear control example

## Penalty method

- ▶ solve sequence of (unconstrained) nonlinear least squares problems

$$\text{minimize} \quad \|f(x)\|^2 + \mu\|g(x)\|^2 = \left\| \begin{bmatrix} f(x) \\ \sqrt{\mu}g(x) \end{bmatrix} \right\|^2$$

- ▶  $\mu$  is a positive *penalty parameter*
- ▶ instead of insisting on  $g(x) = 0$  we assign a penalty to deviations from zero
- ▶ for increasing sequence  $\mu^{(1)}, \mu^{(2)}, \dots$ , compute  $x^{(k+1)}$  by minimizing

$$\|f(x)\|^2 + \mu^{(k)}\|g(x)\|^2$$

- ▶  $x^{(k+1)}$  is computed by Levenberg–Marquardt algorithm started at  $x^{(k)}$

## Termination

- ▶ recall optimality condition

$$2Df(\hat{x})^T f(\hat{x}) + Dg(\hat{x})^T \hat{z} = 0, \quad g(\hat{x}) = 0$$

- ▶  $x^{(k)}$  satisfies normal equations for linear least squares problem:

$$2Df(x^{(k)})^T f(x^{(k)}) + 2\mu^{(k-1)} Dg(x^{(k)})^T g(x^{(k)}) = 0$$

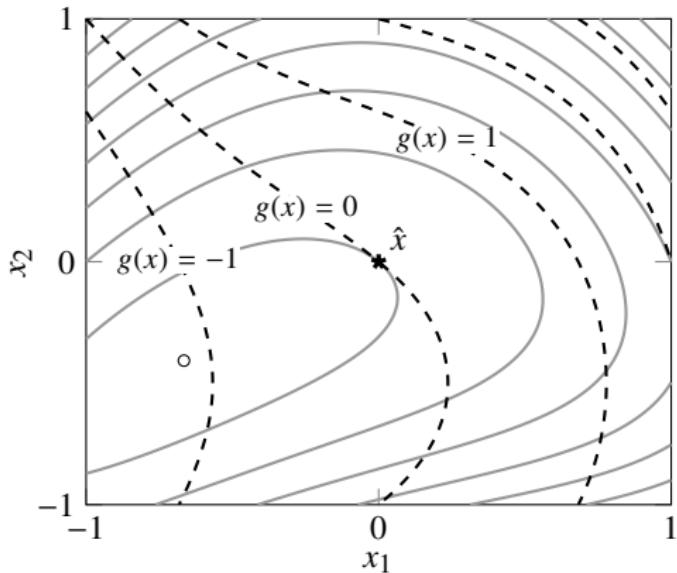
- ▶ if we define  $z^{(k)} = 2\mu^{(k-1)} g(x^{(k)})$ , this can be written as

$$2Df(x^{(k)})^T f(x^{(k)}) + Dg(x^{(k)})^T z^{(k)} = 0$$

- ▶ we see that  $x^{(k)}, z^{(k)}$  satisfy first equation in optimality condition
- ▶ feasibility  $g(x^{(k)}) = 0$  is only satisfied approximately for  $\mu^{(k-1)}$  large enough
- ▶ penalty method is terminated when  $\|g(x^{(k)})\|$  becomes sufficiently small

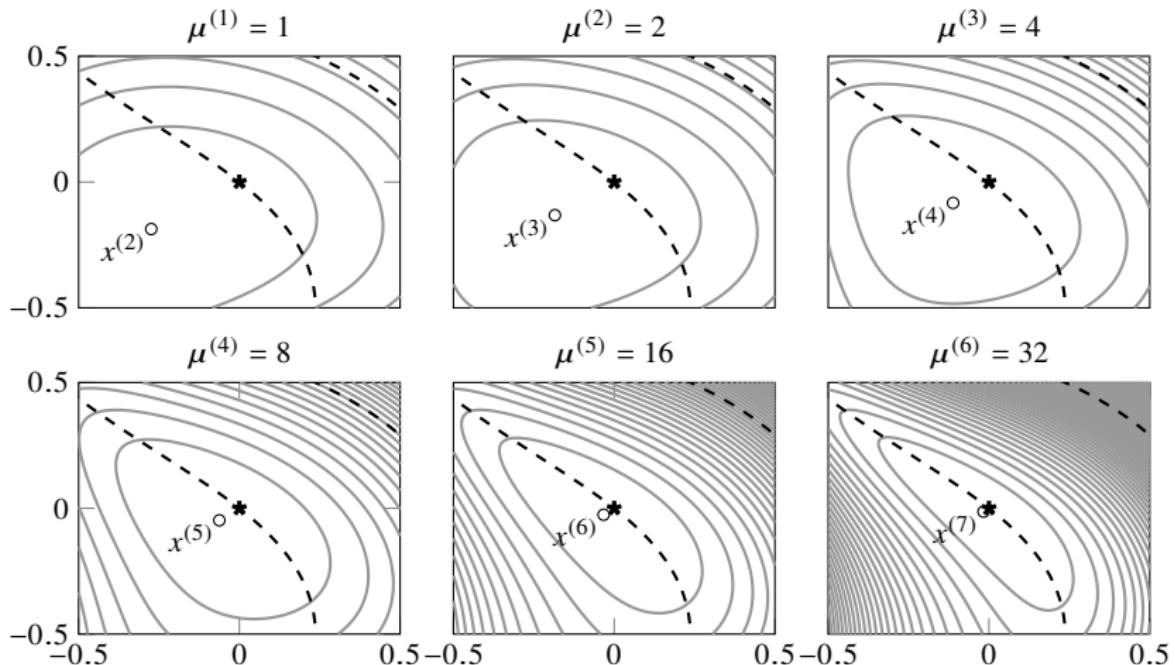
## Example

$$f(x_1, x_2) = \begin{bmatrix} x_1 + \exp(-x_2) \\ x_1^2 + 2x_2 + 1 \end{bmatrix}, \quad g(x_1, x_2) = x_1 + x_1^3 + x_2 + x_2^2$$



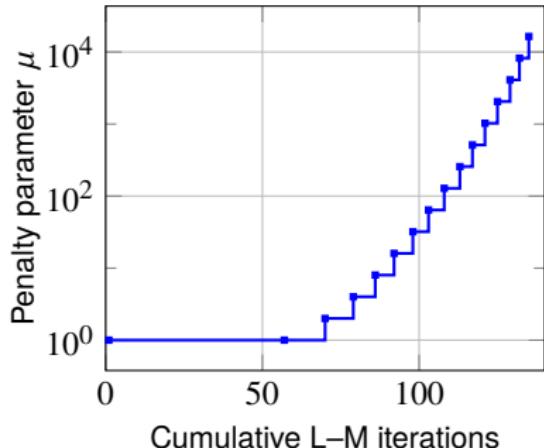
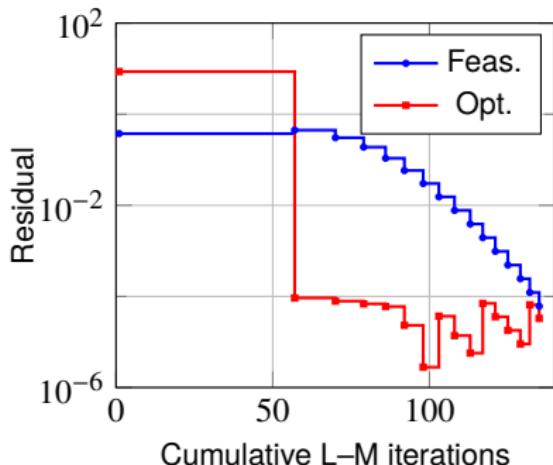
- ▶ solid: contour lines of  $\|f(x)\|^2$
- ▶ dashed: contour lines of  $g(x)$
- ▶  $\hat{x}$  is solution

## First six iterations



solid lines are contour lines of  $\|f(x)\|^2 + \mu^{(k)} \|g(x)\|^2$

## Convergence



- ▶ figure on the left shows residuals in optimality condition
- ▶ blue curve is norm of  $g(x^{(k)})$
- ▶ red curve is norm of  $2Df(x^{(k)})^T f(x^{(k)}) + Dg(x^{(k)})^T z^{(k)}$

# Outline

Constrained nonlinear least squares

Penalty method

Augmented Lagrangian method

Nonlinear control example

## Drawback of penalty method

- ▶  $\mu^{(k)}$  increases rapidly and must become large to drive  $g(x)$  to (near) zero
- ▶ for large  $\mu^{(k)}$ , nonlinear least squares subproblem becomes harder
- ▶ for large  $\mu^{(k)}$ , Levenberg–Marquardt method can take a large number of iterations, or fail

## Augmented Lagrangian

- ▶ the *augmented Lagrangian* for the constrained NLLS problem is

$$\begin{aligned} L_\mu(x, z) &= L(x, z) + \mu \|g(x)\|^2 \\ &= \|f(x)\|^2 + g(x)^T z + \mu \|g(x)\|^2 \end{aligned}$$

- ▶ this is the Lagrangian  $L(x, z)$  augmented with a quadratic penalty
- ▶  $\mu$  is a positive penalty parameter
- ▶ augmented Lagrangian is the Lagrangian of the equivalent problem

$$\begin{array}{ll} \text{minimize} & \|f(x)\|^2 + \mu \|g(x)\|^2 \\ \text{subject to} & g(x) = 0 \end{array}$$

## Minimizing augmented Lagrangian

- ▶ equivalent expressions for augmented Lagrangian

$$\begin{aligned} L_\mu(x, z) &= \|f(x)\|^2 + g(x)^T z + \mu\|g(x)\|^2 \\ &= \|f(x)\|^2 + \mu\|g(x)\|^2 + \frac{1}{2\mu}\|z\|^2 - \frac{1}{2\mu}\|z\|^2 \\ &= \left\| \begin{bmatrix} f(x) \\ \sqrt{\mu}g(x) + z/(2\sqrt{\mu}) \end{bmatrix} \right\|^2 - \frac{1}{2\mu}\|z\|^2 \end{aligned}$$

- ▶ can be minimized over  $x$  (for fixed  $\mu, z$ ) by Levenberg–Marquardt method:

$$\text{minimize } \left\| \begin{bmatrix} f(x) \\ \sqrt{\mu}g(x) + z/(2\sqrt{\mu}) \end{bmatrix} \right\|^2$$

## Lagrange multiplier update

- minimizer  $\tilde{x}$  of augmented Lagrangian  $L_\mu(x, z)$  satisfies

$$2Df(\tilde{x})^T f(\tilde{x}) + Dg(\tilde{x})^T (2\mu g(\tilde{x}) + z) = 0$$

- if we define  $\tilde{z} = z + 2\mu g(\tilde{x})$  this can be written as

$$2Df(\tilde{x})^T f(\tilde{x}) + Dg(\tilde{x})^T \tilde{z} = 0$$

- this is the first equation in the optimality conditions

$$2Df(\hat{x})^T f(\hat{x}) + Dg(\hat{x})^T \hat{z} = 0, \quad g(\hat{x}) = 0$$

- shows that if  $g(\tilde{x}) = 0$ , then  $\tilde{x}$  is optimal
- if  $g(\tilde{x})$  is not small, suggests  $\tilde{z}$  is a good update for  $z$

## Augmented Lagrangian algorithm

1. set  $x^{(k+1)}$  to be the (approximate) minimizer of

$$\|f(x)\|^2 + \mu^{(k)} \|g(x) + z^{(k)} / (2\mu^{(k)})\|^2$$

using Levenberg–Marquardt algorithm, starting from initial point  $x^{(k)}$

2. *multiplier update:*

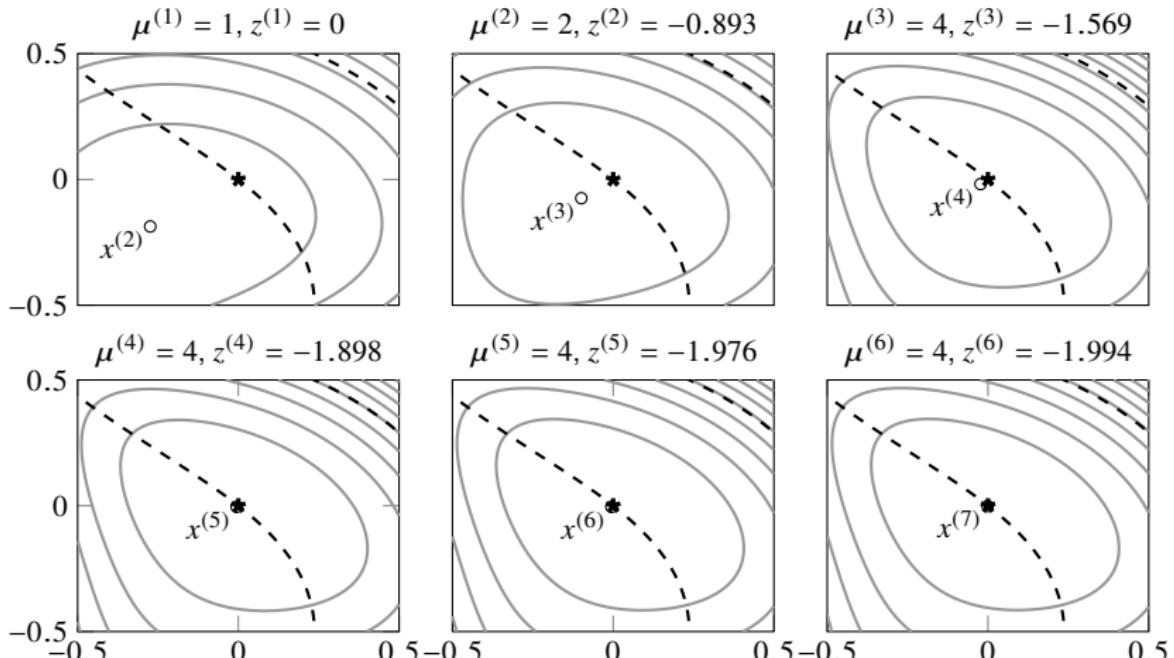
$$z^{(k+1)} = z^{(k)} + 2\mu^{(k)} g(x^{(k+1)}).$$

3. *penalty parameter update:*

$$\mu^{(k+1)} = \mu^{(k)} \quad \text{if } \|g(x^{(k+1)})\| < 0.25 \|g(x^{(k)})\|, \quad \mu^{(k+1)} = 2\mu^{(k)} \quad \text{otherwise}$$

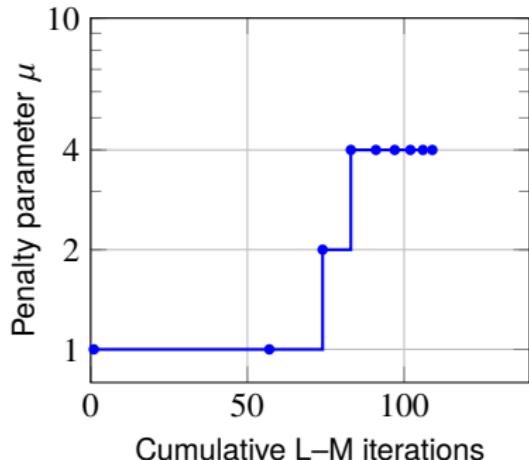
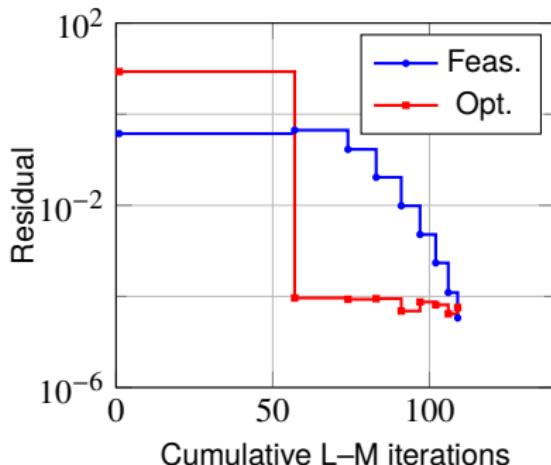
- ▶ iteration starts at  $z^{(1)} = 0$ ,  $\mu^{(1)} = 1$ , some initial  $x^{(1)}$
- ▶  $\mu$  is increased only when needed, more slowly than in penalty method
- ▶ continues until  $g(x^{(k)})$  is sufficiently small (or iteration limit is reached)

## Example of slide 19.9



solid lines are contour lines of  $L_{\mu^{(k)}}(x, z^{(k)})$

## Convergence



- ▶ figure on the left shows residuals in optimality condition
- ▶ blue curve is norm of  $g(x^{(k)})$
- ▶ red curve is norm of  $2Df(x^{(k)})^T f(x^{(k)}) + Dg(x^{(k)})^T z^{(k)}$

# Outline

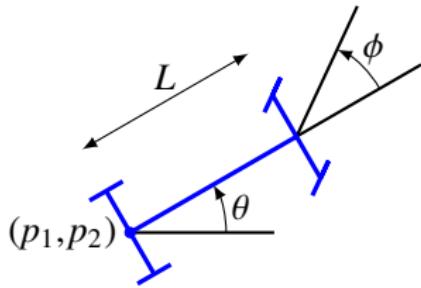
Constrained nonlinear least squares

Penalty method

Augmented Lagrangian method

Nonlinear control example

## Simple model of a car



$$\begin{aligned}\frac{dp_1}{dt} &= s(t) \cos \theta(t) \\ \frac{dp_2}{dt} &= s(t) \sin \theta(t) \\ \frac{d\theta}{dt} &= \frac{s(t)}{L} \tan \phi(t)\end{aligned}$$

- $s(t)$  is speed of vehicle,  $\phi(t)$  is steering angle
- $p(t)$  is position,  $\theta(t)$  is orientation

## Discretized model

- discretized model (for small time interval  $h$ ):

$$p_1(t+h) \approx p_1(t) + hs(t) \cos(\theta(t))$$

$$p_2(t+h) \approx p_2(t) + hs(t) \sin(\theta(t))$$

$$\theta(t+h) \approx \theta(t) + h \frac{s(t)}{L} \tan(\phi(t))$$

- define input vector  $u_k = (s(kh), \phi(kh))$
- define state vector  $x_k = (p_1(kh), p_2(kh), \theta(kh))$
- discretized model is  $x_{k+1} = f(x_k, u_k)$  with

$$f(x_k, u_k) = \begin{bmatrix} (x_k)_1 + h(u_k)_1 \cos((x_k)_3) \\ (x_k)_2 + h(u_k)_1 \sin((x_k)_3) \\ (x_k)_3 + h(u_k)_1 \tan((u_k)_2)/L \end{bmatrix}$$

## Control problem

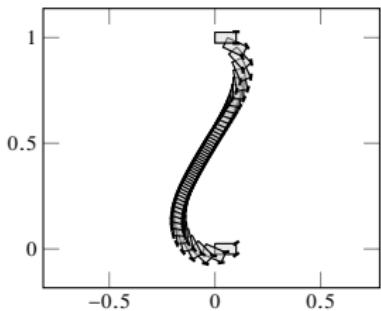
- ▶ move car from given initial to desired final position and orientation
- ▶ using a small and slowly varying input sequence
- ▶ this is a constrained nonlinear least squares problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{k=1}^N \|u_k\|^2 + \gamma \sum_{k=1}^{N-1} \|u_{k+1} - u_k\|^2 \\ \text{subject to} \quad & x_2 = f(0, u_1) \\ & x_{k+1} = f(x_k, u_k), \quad k = 2, \dots, N-1 \\ & x_{\text{final}} = f(x_N, u_N) \end{aligned}$$

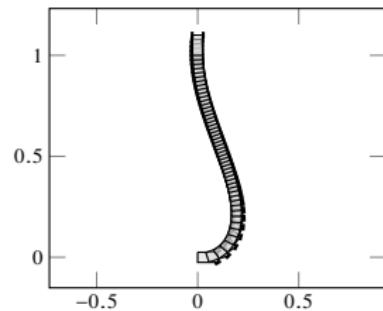
- ▶ variables are  $u_1, \dots, u_N, x_2, \dots, x_N$

## Four solution trajectories

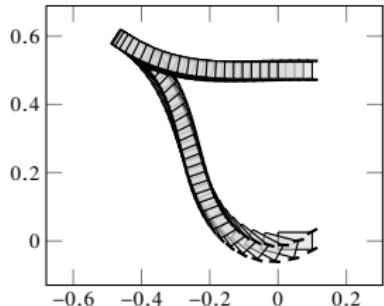
$$x_{\text{final}} = (0, 1, 0)$$



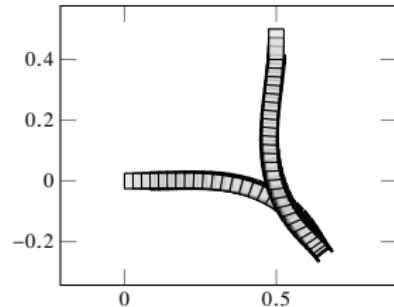
$$x_{\text{final}} = (0, 1, \pi/2)$$



$$x_{\text{final}} = (0, 0.5, 0)$$



$$x_{\text{final}} = (0.5, 0.5, -\pi/2)$$



## Inputs for four trajectories

