Quiz

Let $\mathbf{q}_1, \ldots, \mathbf{q}_n$ be orthonormal vectors in $\mathbb{R}^m$. Let $\mathcal{V} = \text{Span}\ \{\mathbf{q}_1, \ldots, \mathbf{q}_n\}$.

- What does "orthonormal" mean?
- Show:

  There is a matrix M such that, for any vector $\mathbf{b}$ in $\mathbb{R}^m$, the the coordinate representation of $\mathbf{b}^{||\mathcal{V}}$ in terms of $\mathbf{q}_1, \ldots, \mathbf{q}_n$ can be written as $M\mathbf{b}$.

  Be sure to explain.

Projection onto columns of a column-orthogonal matrix

Suppose $\mathbf{q}_1, \ldots, \mathbf{q}_n$ are orthonormal vectors.

Projection of $\mathbf{b}$ onto $\mathbf{q}_j$ is $\mathbf{b}^{\|\mathbf{q}_j} = \sigma_j \mathbf{q}_j$ where $\sigma_j = \dfrac{\langle \mathbf{q}_j, \mathbf{b} \rangle}{\langle \mathbf{q}_j, \mathbf{q}_j \rangle} = \langle \mathbf{q}_j, \mathbf{b} \rangle$

Vector $[\sigma_1, \ldots, \sigma_n]$ can be written using dot-product definition of matrix-vector multiplication:

$$\begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_n \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 \cdot \mathbf{b} \\ \vdots \\ \mathbf{q}_n \cdot \mathbf{b} \end{bmatrix} = \begin{bmatrix} \text{---} \; \mathbf{q}_1^T \; \text{---} \\ \vdots \\ \text{---} \; \mathbf{q}_n^T \; \text{---} \end{bmatrix} \begin{bmatrix} \\ \mathbf{b} \\ \\ \end{bmatrix}$$

and linear combination $\sigma_1 \mathbf{q}_1 + \cdots + \sigma_n \mathbf{q}_n = \left[\begin{array}{c|c|c} & & \\ \mathbf{q}_1 & \cdots & \mathbf{q}_n \\ & & \end{array}\right] \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_n \end{bmatrix}$

Towards QR factorization

Orthogonalization of columns of matrix A gives us a representation of A as product of

- matrix with mutually orthogonal columns
- invertible triangular matrix

$$\begin{bmatrix} | & | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \cdots & \mathbf{v}_n \\ | & | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & | & & | \\ \mathbf{v}_1^* & \mathbf{v}_2^* & \mathbf{v}_3^* & \cdots & \mathbf{v}_n^* \\ | & | & | & & | \end{bmatrix} \begin{bmatrix} 1 & \alpha_{12} & \alpha_{13} & & \alpha_{1n} \\ & 1 & \alpha_{23} & & \alpha_{2n} \\ & & 1 & & \alpha_{3n} \\ & & & \ddots & \\ & & & & \alpha_{n-1,n} \\ & & & & 1 \end{bmatrix}$$

Suppose columns $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are linearly independent. Then $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ are nonzero.

- Normalize $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ (Matrix is called Q)
- To compensate, scale the rows of the triangular matrix. (Matrix is R)

The result is the QR factorization.

Q is a column-orthogonal matrix and R is an upper-triangular matrix.

Towards QR factorization

Orthogonalization of columns of matrix A gives us a representation of A as product of

- matrix with mutually orthogonal columns
- invertible triangular matrix

$$\begin{bmatrix} | & | & | & | \\ \mathbf{v}_2 & \mathbf{v}_3 & \cdots & \mathbf{v}_n \\ | & | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | & | & | \\ \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 & \cdots & \mathbf{q}_n \\ | & | & | & | & | \end{bmatrix} \begin{bmatrix} \|\mathbf{v}_1^*\| & \beta_{12} & \beta_{13} & & \beta_{1n} \\ & \|\mathbf{v}_2^*\| & \beta_{23} & & \beta_{2n} \\ & & \|\mathbf{v}_3^*\| & & \beta_{3n} \\ & & & \ddots & \\ & & & & \beta_{n-1,n} \\ & & & & \|\mathbf{v}_n^*\| \end{bmatrix}$$

Suppose columns $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are linearly independent. Then $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ are nonzero.

- Normalize $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ (Matrix is called Q)
- To compensate, scale the rows of the triangular matrix. (Matrix is R)

The result is the QR factorization.

Q is a column-orthogonal matrix and R is an upper-triangular matrix.

Using the QR factorization to solve a matrix equation $A\mathbf{x} = \mathbf{b}$

First suppose A is square and its columns are linearly independent.
Then A is invertible.
It follows that there is a solution (because we can write $\mathbf{x} = A^{-1}\mathbf{b}$)
QR Solver Algorithm to find the solution in this case:

> Find Q, R such that $A = QR$ and Q is column-orthogonal and R is triangular
> Compute vector $\mathbf{c} = Q^T\mathbf{b}$
> Solve $R\mathbf{x} = \mathbf{c}$ using backward substitution, and return the solution.

Why is this correct?

- Let $\hat{\mathbf{x}}$ be the solution returned by the algorithm.
- We have $R\hat{\mathbf{x}} = Q^T\mathbf{b}$
- Multiply both sides by Q: $Q(R\hat{\mathbf{x}}) = Q(Q^T\mathbf{b})$
- Use associativity: $(QR)\hat{\mathbf{x}} = (QQ^T)\mathbf{b}$
- Substitute A for QR: $A\hat{\mathbf{x}} = (QQ^T)\mathbf{b}$
- Since Q and Q^T are inverses, we know QQ^T is identity matrix: $A\hat{\mathbf{x}} = \mathbb{1}\mathbf{b}$

Thus $A\hat{\mathbf{x}} = \mathbf{b}$.

Solving $A\mathbf{x} = \mathbf{b}$

What if columns of A are not independent?

Let $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$ be columns of A.

Suppose $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$ are linearly dependent.

Then there is a basis consisting of a subset, say $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_4$

$$\left\{ \left[\begin{array}{c|c|c|c} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \ : \ x_1, x_2, x_3, x_4 \in \mathbb{R} \right\} =$$

$$\left\{ \left[\begin{array}{c|c|c} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_4 \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ x_4 \end{bmatrix} \ : \ x_1, x_2, x_4 \in \mathbb{R} \right\}$$

Therefore: if there is a solution to $A\mathbf{x} = \mathbf{b}$ then there is a solution to $A'\mathbf{x}' = \mathbf{b}$ where columns of A' are a subset basis of columns of A (and $\mathbf{x}'$ consists of corresponding variables).
So solve $A'\mathbf{x}' = \mathbf{b}$ instead.

The least squares problem

Suppose A is an $m \times n$ matrix and its columns are linearly independent.

Since each column is an m-vector, dimension of column space is at most m, so $n \leq m$.

What if $n < m$? How can we solve the matrix equation $A\mathbf{x} = \mathbf{b}$?

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \mathbf{b}$$

Remark: There might not be a solution:

- Define $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ by $f(\mathbf{x}) = A\mathbf{x}$
- Dimension of Im f is n
- Dimension of co-domain is m.
- Thus f is not onto.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \mathbf{b}$$

Goal: An algorithm that, given a matrix A whose columns are linearly independent and given $\mathbf{b}$, finds the vector $\hat{\mathbf{x}}$ minimizing $\|\mathbf{b} - A\hat{\mathbf{x}}\|$.

Solution: Same algorithm as we used for square A

The least squares problem

Recall...

High-Dimensional Fire Engine Lemma: The point in a vector space $\mathcal{V}$ closest to $\mathbf{b}$ is $\mathbf{b}^{||\mathcal{V}}$ and the distance is $\|\mathbf{b}^{\perp\mathcal{V}}\|$.

Given equation $A\mathbf{x} = \mathbf{b}$, let $\mathcal{V}$ be the column space of A.

We need to show that the QR Solver Algorithm returns a vector $\hat{\mathbf{x}}$ such that $A\hat{\mathbf{x}} = \mathbf{b}^{||\mathcal{V}}$.

Projection onto columns of a column-orthogonal matrix

Suppose $\mathbf{q}_1, \ldots, \mathbf{q}_n$ are orthonormal vectors.

Projection of $\mathbf{b}$ onto $\mathbf{q}_j$ is $\mathbf{b}^{\|\mathbf{q}_j} = \sigma_j \mathbf{q}_j$ where $\sigma_j = \dfrac{\langle \mathbf{q}_j, \mathbf{b} \rangle}{\langle \mathbf{q}_j, \mathbf{q}_j \rangle} = \langle \mathbf{q}_j, \mathbf{b} \rangle$

Vector $[\sigma_1, \ldots, \sigma_n]$ can be written using dot-product definition of matrix-vector multiplication:

$$\begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_n \end{bmatrix} = \begin{bmatrix} \mathbf{q}_1 \cdot \mathbf{b} \\ \vdots \\ \mathbf{q}_n \cdot \mathbf{b} \end{bmatrix} = \begin{bmatrix} \text{---} \; \mathbf{q}_1^T \; \text{---} \\ \vdots \\ \text{---} \; \mathbf{q}_n^T \; \text{---} \end{bmatrix} \begin{bmatrix} \\ \mathbf{b} \\ \\ \end{bmatrix}$$

and linear combination $\sigma_1 \mathbf{q}_1 + \cdots + \sigma_n \mathbf{q}_n = \left[\begin{array}{c|c|c} & & \\ \mathbf{q}_1 & \cdots & \mathbf{q}_n \\ & & \end{array}\right] \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_n \end{bmatrix}$

QR Solver Algorithm for $A\mathbf{x} \approx \mathbf{b}$

Summary:

- $QQ^T\mathbf{b} = \mathbf{b}^{||}$

Proposed algorithm:

> Find Q, R such that $A = QR$ and Q is column-orthogonal and R is triangular
> Compute vector $\mathbf{c} = Q^T\mathbf{b}$
> Solve $R\mathbf{x} = \mathbf{c}$ using backward substitution, and return the solution $\hat{\mathbf{x}}$.

Goal: To show that the solution $\hat{\mathbf{x}}$ returned is the vector that minimizes $\|\mathbf{b} - A\hat{\mathbf{x}}\|$

Every vector of the form $A\mathbf{x}$ is in Col A (= Col Q)

By the High-Dimensional Fire Engine Lemma, the vector in Col A closest to $\mathbf{b}$ is $\mathbf{b}^{||}$, the projection of $\mathbf{b}$ onto Col A.

Solution $\hat{\mathbf{x}}$ satisfies $R\hat{\mathbf{x}} = Q^T\mathbf{b}$

Multiply by Q: $QR\hat{\mathbf{x}} = QQ^T\mathbf{b}$

Therefore $A\hat{\mathbf{x}} = \mathbf{b}^{||}$.

Least squares when columns are linearly *de*pendent?

This comes up, e.g. ranking sports teams.
Need a more sophisticated algorithm.
We'll see it soon.

The Normal Equations

Let A be a matrix with linearly independent columns. Let QR be its QR factorization. We have given one algorithm for solving the least-squares problem $A\mathbf{x} \approx \mathbf{b}$:

> Find Q, R such that $A = QR$ and Q is column-orthogonal and R is triangular
> Compute vector $\mathbf{c} = Q^T\mathbf{b}$
> Solve $R\mathbf{x} = \mathbf{c}$ using backward substitution, and return the solution $\hat{\mathbf{x}}$.

However, there are other ways to find solution.

Not hard to show that

- A^TA is an invertible matrix
- The solution to the matrix-vector equation $(A^TA)\mathbf{x} = A^T\mathbf{b}$ is the solution to the least-squares problem $A\mathbf{x} \approx \mathbf{b}$
- Can use another method (e.g. Gaussian elimination) to solve $(A^T)\mathbf{x} = A^T\mathbf{b}$

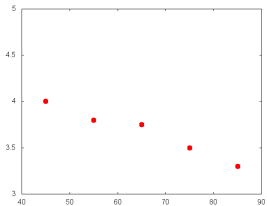
The linear equations making up $A^TA\mathbf{x} = A^T\mathbf{b}$ are called the *normal equations.*

Application of least squares: linear regression

Finding the line that best fits some two-dimensional data.

Data on age versus brain mass from the Bureau of Made-up Numbers:

age	brain mass
45	4 lbs.
55	3.8
65	3.75
75	3.5
85	3.3



Let $f(x)$ be the function that predicts brain mass for someone of age x.

Hypothesis: after age 45, brain mass decreases linearly with age, i.e. that $f(x) = mx + b$ for some numbers m, b.

Goal: find m, b to as to minimize the sum of squares of prediction errors

The observations are $(x_1, y_1) = (45, 4)$, $(x_2, y_2) = (55, 3.8)$, $(x_3, y_3) = (65, 3.75)$,$(x_4, y_4) = (75, 3.5)$, $(x_5, y_5) = (85, 3.3)$.

The prediction error on the i^{th} observation is $|f(x_i) - y_i|$.

The sum of squares of prediction errors is $\sum_i (f(x_i) - y_i)^2$.

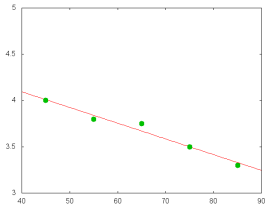
For each observation, measure the difference between the predicted and observed y-value. In this application, this difference is measured in pounds. Measuring the distance from the point to the line wouldn't make sense.

Application of least squares: linear regression

Finding the line that best fits some two-dimensional data.

Data on age versus brain mass from the Bureau of Made-up Numbers:

age	brain mass
45	4 lbs.
55	3.8
65	3.75
75	3.5
85	3.3



Let $f(x)$ be the function that predicts brain mass for someone of age x.

Hypothesis: after age 45, brain mass decreases linearly with age, i.e. that $f(x) = mx + b$ for some numbers m, b.

Goal: find m, b to as to minimize the sum of squares of prediction errors

The observations are $(x_1, y_1) = (45, 4)$, $(x_2, y_2) = (55, 3.8)$, $(x_3, y_3) = (65, 3.75)$,$(x_4, y_4) = (75, 3.5)$, $(x_5, y_5) = (85, 3.3)$.

The prediction error on the i^{th} observation is $|f(x_i) - y_i|$.

The sum of squares of prediction errors is $\sum_i (f(x_i) - y_i)^2$.

For each observation, measure the difference between the predicted and observed y-value. In this application, this difference is measured in pounds. Measuring the distance from the point to the line wouldn't make sense.

Application of least squares: linear regression

Finding the line that best fits some two-dimensional data.

Data on age versus brain mass from the Bureau of Made-up Numbers:

age	brain mass
45	4 lbs.
55	3.8
65	3.75
75	3.5
85	3.3

Let $f(x)$ be the function that predicts brain mass for someone of age x.

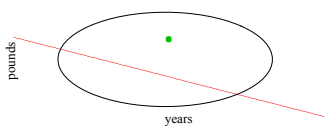
Hypothesis: after age 45, brain mass decreases linearly with age, i.e. that $f(x) = mx + b$ for some numbers m, b.

Goal: find m, b to as to minimize the sum of squares of prediction errors

The observations are $(x_1, y_1) = (45, 4)$, $(x_2, y_2) = (55, 3.8)$, $(x_3, y_3) = (65, 3.75)$,$(x_4, y_4) = (75, 3.5)$, $(x_5, y_5) = (85, 3.3)$.

The prediction error on the i^{th} observation is $|f(x_i) - y_i|$.

The sum of squares of prediction errors is $\sum_i (f(x_i) - y_i)^2$.



For each observation, measure the difference between the predicted and observed y-value. In this application, this difference is measured in pounds. Measuring the distance from the point to the line wouldn't make sense.

Linear regression

To find the best line for given data (x_1, y_1),(x_2, y_2),(x_3, y_3),(x_4, y_4),(x_5, y_5),
solve this least-squares problem

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ x_5 & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

The dot-product of row i with the vector $[m, b]$ is $mx_i + b$, i.e. the value predicted by $f(x) = mx + b$ for the i^{th} observation.

Therefore, the vector of predictions is $A \begin{bmatrix} m \\ b \end{bmatrix}$.

The vector of differences between predictions and observed values is $A \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$, and the sum of squares of differences is the squared norm of this vector.

Therefore the method of least squares can be used to find the pair (m, b) that minimizes the sum of squares, i.e. the line that best fits the data.

Application of least squares: coping with approximate data

Recall the *industrial espionage* problem: finding the number of each product being produced from the amount of each resource being consumed.





Let M =

	metal	concrete	plastic	water	electricity
garden gnome	0	1.3	.2	.8	.4
hula hoop	0	0	1.5	.4	.3
slinky	.25	0	0	.2	.7
silly putty	0	0	.3	.7	.5
salad shooter	.15	0	.5	.4	.8

We solved $\mathbf{u}^T M = \mathbf{b}$ where $\mathbf{b}$ is vector giving amount of each resource consumed:

$\mathbf{b} =$

metal	concrete	plastic	water	electricity
226.25	1300	677	1485	1409.5

`solve(M.transpose(), b)` gives us $\mathbf{u} \approx$

gnome	hoop	slinky	putty	shooter
1000	175	860	590	75

Application of least squares: industrial espionage problem

More realistic scenario: measurement of resources consumed is approximate

True amounts: $\mathbf{b} =$

metal	concrete	plastic	water	electricity
226.25	1300	677	1485	1409.5

Solving with true amounts gives

gnome	hoop	slinky	putty	shooter
1000	175	860	590	75

Measurements: $\tilde{\mathbf{b}} =$

metal	concrete	plastic	water	electricity
223.23	1331.62	679.32	1488.69	1492.64

Solving with measurements gives

gnome	hoop	slinky	putty	shooter
1024.32	28.85	536.32	446.7	594.34

Slight changes in input data leads to pretty big changes in output.

Output data not accurate, perhaps not useful! (see slinky, shooter)

Question: How can we improve accuracy of output without more accurate measurements?

Answer: More measurements!

Application of least squares: industrial espionage problem

Have to measure something else, e.g. amount of waste water produced

	metal	concrete	plastic	water	electricity	waste water
garden gnome	0	1.3	.2	.8	.4	.3
hula hoop	0	0	1.5	.4	.3	.35
slinky	.25	0	0	.2	.7	0
silly putty	0	0	.3	.7	.5	.2
salad shooter	.15	0	.5	.4	.8	.15

Measured: $\tilde{\mathbf{b}} =$

metal	concrete	plastic	water	electricity	waste water
223.23	1331.62	679.32	1488.69	1492.64	489.19

Equation $\mathbf{u} * M = \tilde{\mathbf{b}}$ is more constrained $\Rightarrow$ has no solution

but least-squares solution is

gnome	hoop	slinky	putty	shooter
1022.26	191.8	1005.58	549.63	41.1

True amounts:

gnome	hoop	slinky	putty	shooter
1000	175	860	590	75

Better output accuracy with same input accuracy

Application of least squares: Sensor node problem

Recall *sensor node problem*: estimate current draw for each hardware component

Define `D = {'radio', 'sensor', 'memory', 'CPU'}`.

Goal: Compute a `D`-vector $\mathbf{u}$ that, for each hardware component, gives the current drawn by that component.

Four test periods:

- total mA-seconds in these test periods $\mathbf{b} = [140, 170, 60, 170]$
- for each test period, vector specifying how long each hardware device was operating:

  $\mathbf{duration}_1$ = `Vec(D, 'radio':0.1, 'CPU':0.3)`

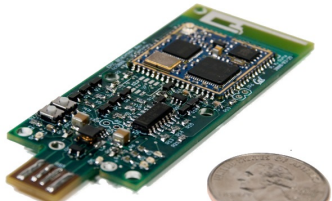
  $\mathbf{duration}_2$ = `Vec(D, 'sensor':0.2, 'CPU':0.4)`

  $\mathbf{duration}_3$ = `Vec(D, 'memory':0.3, 'CPU':0.1)`

  $\mathbf{duration}_4$ = `Vec(D, 'memory':0.5, 'CPU':0.4)`

To get $\mathbf{u}$, solve $A\mathbf{x} = \mathbf{b}$ where

$$A = \begin{bmatrix} \mathbf{duration}_1 \\ \hline \mathbf{duration}_2 \\ \hline \mathbf{duration}_3 \\ \hline \mathbf{duration}_4 \end{bmatrix}$$



Application of least squares: Sensor node problem

If measurement are exact, get back true current draw for each hardware component:

$$\mathbf{b} = [140, 170, 60, 170]$$

solve $A\mathbf{x} = \mathbf{b}$

radio	sensor	CPU	memory
500	250	300	100

More realistic: approximate measurement

$$\tilde{\mathbf{b}} = [141.27, 160.59, 62.47, 181.25]$$

solve $A\mathbf{x} = \tilde{\mathbf{b}}$

radio	sensor	CPU	memory
421	142	331	98.1

How can we get more accurate results?

Solution: Add more test periods and solve least-squares problem

Application of least squares: Sensor node problem

$\mathbf{duration}_1 =$ `Vec(D, 'radio':0.1, 'CPU':0.3)`

$\mathbf{duration}_2 =$ `Vec(D, 'sensor':0.2, 'CPU':0.4)`

$\mathbf{duration}_3 =$ `Vec(D, 'memory':0.3, 'CPU':0.1)`

$\mathbf{duration}_4 =$ `Vec(D, 'memory':0.5, 'CPU':0.4)`

$\mathbf{duration}_5 =$ `Vec(D, 'radio':0.2, 'CPU':0.5)`

$\mathbf{duration}_6 =$ `Vec(D, 'sensor':0.3, 'radio':0.8, 'CPU':0.9, 'memory':0.8)`

$\mathbf{duration}_7 =$ `Vec(D, 'sensor':0.5, 'radio':0.3, 'CPU':0.9, 'memory':0.5)`

$\mathbf{duration}_8 =$ `Vec(D, 'radio':0.2, 'CPU':0.6)`

Let $A = \begin{bmatrix} \mathbf{duration}_1 \\ \mathbf{duration}_2 \\ \mathbf{duration}_3 \\ \mathbf{duration}_4 \\ \mathbf{duration}_5 \\ \mathbf{duration}_6 \\ \mathbf{duration}_7 \\ \mathbf{duration}_8 \end{bmatrix}$

Measurement vector is

$\tilde{\mathbf{b}} = [141.27, 160.59, 62.47, 181.25, 247.74, 804.58, 609.10, 282.09]$

Now $A\mathbf{x} = \tilde{\mathbf{b}}$ has no solution.

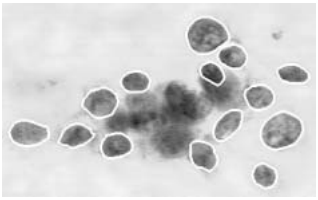
But solution to least-squares problem is

radio	sensor	CPU	memory
451.40	252.07	314.37	111.66

True solution is

radio	sensor	CPU	memory
500	250	300	100

Applications of least squares: breast cancer machine-learning problem



Recall: breast-cancer machine-learning lab

Input: vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m$ giving features of specimen, values $b_1, \ldots, b_m$ specifying $+1$ (malignant) or -1 (benign)

Informal goal: Find vector $\mathbf{w}$ such that sign of $\mathbf{a}_i \cdot \mathbf{w}$ predicts sign of b_i

Formal goal: Find vector $\mathbf{w}$ to minimize sum of squared errors
$(b_1 - \mathbf{a}_1 \cdot \mathbf{w})^2 + \cdots + (b_m - \mathbf{a}_m \cdot \mathbf{w})^2$

Approach: Gradient descent

Results: Took a few minutes to get a solution with error rate around 7%

Can we do better with least squares?

Applications of least squares: breast cancer machine-learning problem

Goal: Find the vector $\mathbf{w}$ that minimizes $(\mathbf{b}[1] - \mathbf{a}_1 \cdot \mathbf{w})^2 + \cdots + (\mathbf{b}[m] - \mathbf{a}_m \cdot \mathbf{w})^2$

Equivalent: Find the vector $\mathbf{w}$ that minimizes $\left\| \begin{bmatrix} \\ \mathbf{b} \\ \\ \end{bmatrix} - \left[\begin{array}{c} \mathbf{a}_1 \\ \hline \vdots \\ \hline \mathbf{a}_m \end{array}\right] \begin{bmatrix} \\ \mathbf{x} \\ \\ \end{bmatrix} \right\|^2$

This is the least-squares problem.

Using the algorithm based on QR factorization takes a fraction of a second and gets a solution with smaller error rate.

Even better solutions using more sophisticated techniques in linear algebra:

- Use an inner product that better reflects the variance of each of the features.
- Use *linear programming*
- Even more general: use *convex programming*

Rating sports teams: Use of least squares when columns are linearly dependent

	Duke	Miami	UNC	UVA	VT
Duke		7-52	21-24	7-38	0-45
Miami	52-7		34-16	25-17	27-7
UNC	24-21	16-34		7-5	3-30
UVA	38-7	17-25	5-7		14-52
VT	45-0	7-27	30-3	52-14	

Compute ratings r_i for teams using Massey's equation: $r_i - r_j$ predicts score difference when team i plays team j.

```
         Duke Miami UNC UVA  VT
       ------------------------
 Duke  |    0   -45  -3 -31 -45
Miami  |   45     0  18   8  20
  UNC  |    3   -18   0   2 -27
  UVA  |   31    -8  -2   0 -38
   VT  |   45   -20  27  38   0
```

Deblurring

Can we reverse the blurring process?

Use of least squares when columns are linearly dependent

Suppose columns are linearly dependent.

Then a given vector in the column space can be represented in multiple ways as a linear combination of columns.

Thus least-squares solution is not unique.

Which solution should we seek?

We'll end up finding the solution with the smallest norm.