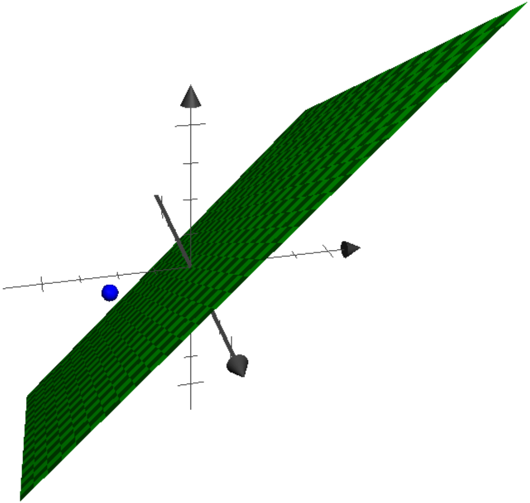
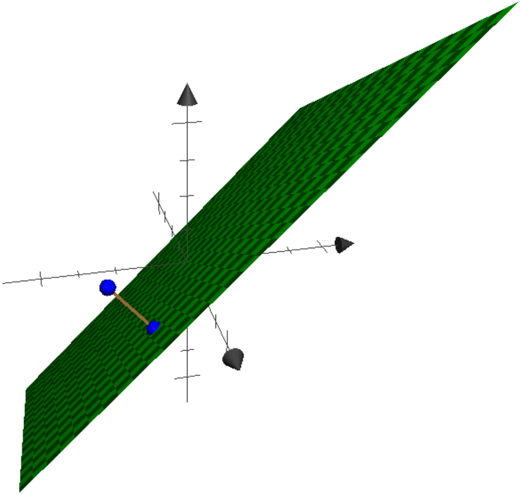
[9] Orthogonalization

Finding the closest point in a plane

Goal: Given a point **b** and a plane, find the point in the plane closest to **b**.





Finding the closest point in a plane

Goal: Given a point **b** and a plane, find the point in the plane closest to **b**.

By translation, we can assume the plane includes the origin.

The plane is a vector space $\mathcal{V}$. Let $\{\mathbf{v}_1, \mathbf{v}_2\}$ be a basis for $\mathcal{V}$.

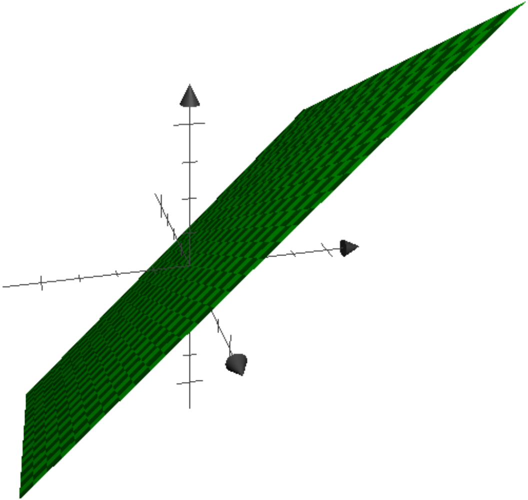
Goal: Given a point **b**, find the point in Span $\{\mathbf{v}_1, \mathbf{v}_2\}$ closest to **b**.

Example:

$$\mathbf{v}_1 = [8, -2, 2] \text{ and } \mathbf{v}_2 = [4, 2, 4]$$

$\mathbf{b} = [5, -5, 2]$

point in plane closest to **b**: $[6, -3, 0]$.



Closest-point problem in in higher dimensions

Goal: An algorithm that, given a vector $\mathbf{b}$ and vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, finds the vector in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ that is closest to $\mathbf{b}$.

Special case: We can use the algorithm to determine whether $\mathbf{b}$ lies in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$:
If the vector in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ closest to $\mathbf{b}$ is $\mathbf{b}$ itself then clearly $\mathbf{b}$ is in the span; if not, then $\mathbf{b}$ is not in the span.

Let $A = \left[\begin{array}{c|c|c} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{array}\right]$.

Using the linear-combinations interpretation of matrix-vector multiplication, a vector in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ can be written $A\mathbf{x}$.

Thus testing if $\mathbf{b}$ is in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is equivalent to testing if the equation $A\mathbf{x} = \mathbf{b}$ has a solution.

More generally:
Even if $A\mathbf{x} = \mathbf{b}$ has no solution, we can use the algorithm to find the point in $\{A\mathbf{x} \ : \ \mathbf{x} \in \mathbb{R}^n\}$ closest to $\mathbf{b}$.

Moreover: We hope to extend the algorithm to also find the best solution $\mathbf{x}$.

Closest point and coefficients

Not enough to find the point p in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ closest to $\mathbf{b}$....
We need an algorithm to find the representation of p in terms of $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

Goal: find the coefficients $x_1, \ldots, x_n$ so that $x_1\,\mathbf{v}_1 + \cdots + x_n\,\mathbf{v}_n$ is the vector in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ closest to $\mathbf{b}$.

Equivalent: Find the vector $\mathbf{x}$ that minimizes $\left\| \begin{bmatrix} \mathbf{b} \end{bmatrix} - \left[\begin{array}{c|c|c} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{array}\right] \begin{bmatrix} \mathbf{x} \end{bmatrix} \right\|$

Equivalent: Find the vector $\mathbf{x}$ that minimizes $\left\| \begin{bmatrix} \mathbf{b} \end{bmatrix} - \left[\begin{array}{c|c|c} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{array}\right] \begin{bmatrix} \mathbf{x} \end{bmatrix} \right\|^2$

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

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

This last problem was addressed using gradient descent in Machine Learning lab.

Closest point and least squares

Find the vector $\mathbf{x}$ that minimizes $\left\| \begin{bmatrix} \\ \mathbf{b} \\ \\ \end{bmatrix} - \left[\begin{array}{c|c|c} \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ \\ \end{array} \right] \begin{bmatrix} \\ \mathbf{x} \\ \\ \end{bmatrix} \right\|^2$

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

This problem is called *least squares* ("méthode des moindres carrés", due to Adrien-Marie Legendre but often attributed to Gauss)

Equivalent: Given a matrix equation $A\mathbf{x} = \mathbf{b}$ that might have no solution, find the best solution available in the sense that the norm of the error $\mathbf{b} - A\mathbf{x}$ is as small as possible.

- There is an algorithm based on Gaussian elimination.
- We will develop an algorithm based on orthogonality (used in `solver`)

  Much faster and more reliable than gradient descent.

High-dimensional projection onto/orthogonal to

For any vector $\mathbf{b}$ and any vector $\mathbf{a}$, define vectors $\mathbf{b}^{||\mathbf{a}}$ and $\mathbf{b}^{\perp\mathbf{a}}$ so that

$$\mathbf{b} = \mathbf{b}^{||\mathbf{a}} + \mathbf{b}^{\perp\mathbf{a}}$$

and there is a scalar $\sigma \in R$ such that

$$\mathbf{b}^{||\mathbf{a}} = \sigma\, \mathbf{a}$$

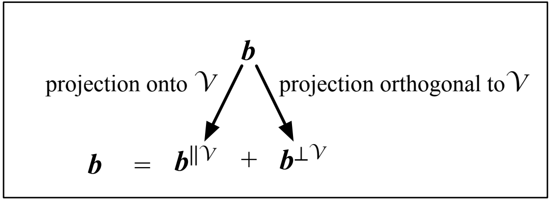
and

$$\mathbf{b}^{\perp\mathbf{a}} \text{ is orthogonal to } \mathbf{a}$$

Definition: For a vector $\mathbf{b}$ and a vector space $\mathcal{V}$, we define the projection of $\mathbf{b}$ onto $\mathcal{V}$ (written $\mathbf{b}^{||\mathcal{V}}$) and the projection of $\mathbf{b}$ orthogonal to $\mathcal{V}$ (written $\mathbf{b}^{\perp\mathcal{V}}$) so that

$$\mathbf{b} = \mathbf{b}^{||\mathcal{V}} + \mathbf{b}^{\perp\mathcal{V}}$$

and $\mathbf{b}^{||\mathcal{V}}$ is in $\mathcal{V}$, and $\mathbf{b}^{\perp\mathcal{V}}$ is orthogonal to every vector in $\mathcal{V}$.



High-Dimensional Fire Engine Lemma

Definition: For a vector $\mathbf{b}$ and a vector space $\mathcal{V}$, we define the projection of $\mathbf{b}$ onto $\mathcal{V}$ (written $\mathbf{b}^{||\mathcal{V}}$) and the projection of $\mathbf{b}$ orthogonal to $\mathcal{V}$ (written $\mathbf{b}^{\perp\mathcal{V}}$) so that

$$\mathbf{b} = \mathbf{b}^{||\mathcal{V}} + \mathbf{b}^{\perp\mathcal{V}}$$

and $\mathbf{b}^{||\mathcal{V}}$ is in $\mathcal{V}$, and $\mathbf{b}^{\perp\mathcal{V}}$ is orthogonal to every vector in $\mathcal{V}$.

One-dimensional Fire Engine Lemma: The point in Span $\{\mathbf{a}\}$ closest to $\mathbf{b}$ is $\mathbf{b}^{||\mathbf{a}}$ and the distance is $\|\mathbf{b}^{\perp\mathbf{a}}\|$.

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 $\|\mathbf{b}^{\perp\mathcal{V}}\|$.

Finding the projection of $\mathbf{b}$ orthogonal to Span $\{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$

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

Suppose $\mathcal{V}$ is specified by generators $\mathbf{v}_1, \ldots, \mathbf{v}_n$

Goal: An algorithm for computing $\mathbf{b}^{\perp\mathcal{V}}$ in this case.

- *input:* vector $\mathbf{b}$, vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$
- *output:* projection of $\mathbf{b}$ orthogonal to $\mathcal{V} = \text{Span}\ \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$

We already know how to solve this when $n = 1$:

```
def project_orthogonal_1(b, v):
 return b - project_along(b, v)
```

Let's try to generalize....

project_orthogonal(b, vlist)

```
def project_orthogonal_1(b, v):
 return b - project_along(b, v)
```

$\Downarrow$

```
def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

Reviews are in....

"Short, elegant, ... and flawed"

"Beautiful—if only it worked!"

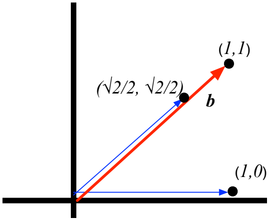
"A tragic failure."

project_orthogonal(b, vlist) doesn't work

```
def project_orthogonal(b, vlist):
  for v in vlist:
    b = b - project_along(b, v)
  return b
```

$\mathbf{b} = [1,1]$

vlist $=[\ [1,0],$
$[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]\ ]$



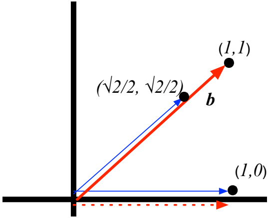
Let $\mathbf{b}_i$ be value of the variable $\mathbf{b}$ after i iterations.

$$
\begin{aligned}
\mathbf{b}_1 &= \mathbf{b}_0 - (\text{projection of } [1,1] \text{ along } [1,0]) \\
&= \mathbf{b}_0 - [1,0] \\
&= [0,1] \\
\mathbf{b}_2 &= \mathbf{b}_1 - (\text{projection of } [0,1] \text{ along } [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]) \\
&= \mathbf{b}_1 - [\frac{1}{2}, \frac{1}{2}] \\
&= [-\frac{1}{2}, \frac{1}{2}] \text{ which is not orthogonal to } [1,0]
\end{aligned}
$$

`project_orthogonal(b, vlist)` doesn't work

```
def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

$\mathbf{b} = [1,1]$
`vlist` $=[\ [1,0],$
$[\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]\ ]$



Let $\mathbf{b}_i$ be value of the variable $\mathbf{b}$ after i iterations.

$$\begin{aligned}
\mathbf{b}_1 &= \mathbf{b}_0 - (\text{projection of } [1,1] \text{ along } [1,0]) \\
&= \mathbf{b}_0 - [1,0] \\
&= [0,1]
\end{aligned}$$

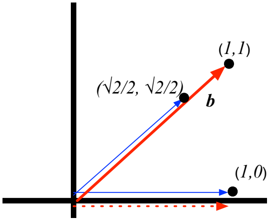
$$\begin{aligned}
\mathbf{b}_2 &= \mathbf{b}_1 - (\text{projection of } [0,1] \text{ along } [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]) \\
&= \mathbf{b}_1 - [\frac{1}{2}, \frac{1}{2}] \\
&= [-\frac{1}{2}, \frac{1}{2}] \ \text{which is not orthogonal to } [1,0]
\end{aligned}$$

project_orthogonal(b, vlist) doesn't work

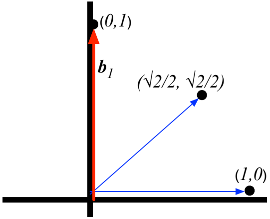
```
def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

$\mathbf{b} = [1,1]$

vlist $= [\ [1,0],$
$\quad [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]\ ]$



Let $\mathbf{b}_i$ be value of the variable $\mathbf{b}$ after i iterations.



$$\begin{aligned}
\mathbf{b}_1 &= \mathbf{b}_0 - (\text{projection of } [1,1] \text{ along } [1,0]) \\
&= \mathbf{b}_0 - [1,0] \\
&= [0,1]
\end{aligned}$$

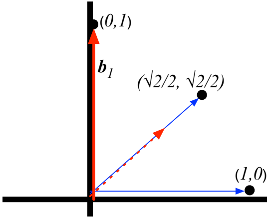
$$\begin{aligned}
\mathbf{b}_2 &= \mathbf{b}_1 - (\text{projection of } [0,1] \text{ along } [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]) \\
&= \mathbf{b}_1 - [\frac{1}{2}, \frac{1}{2}] \\
&= [-\frac{1}{2}, \frac{1}{2}] \text{ which is not orthogonal to } [1,0]
\end{aligned}$$

`project_orthogonal(b, vlist)` doesn't work

```
def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

$\mathbf{b} = [1,1]$

`vlist` $=[\ [1,0], [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]\ ]$



Let $\mathbf{b}_i$ be value of the variable $\mathbf{b}$ after i iterations.

$$
\begin{aligned}
\mathbf{b}_1 &= \mathbf{b}_0 - (\text{projection of } [1,1] \text{ along } [1,0]) \\
&= \mathbf{b}_0 - [1,0] \\
&= [0,1] \\
\mathbf{b}_2 &= \mathbf{b}_1 - (\text{projection of } [0,1] \text{ along } [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]) \\
&= \mathbf{b}_1 - [\frac{1}{2}, \frac{1}{2}] \\
&= [-\frac{1}{2}, \frac{1}{2}] \ \text{which is not orthogonal to } [1,0]
\end{aligned}
$$

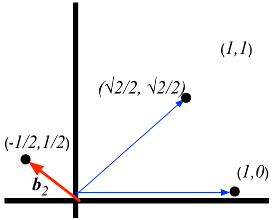
`project_orthogonal(b, vlist)` doesn't work

```
def project_orthogonal(b, vlist):
  for v in vlist:
    b = b - project_along(b, v)
  return b
```

$\mathbf{b} = [1,1]$

$\texttt{vlist} = [\ [1,0],\ [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]\ ]$

Let $\mathbf{b}_i$ be value of the variable $\mathbf{b}$ after i iterations.



$$
\begin{aligned}
\mathbf{b}_1 &= \mathbf{b}_0 - (\text{projection of } [1,1] \text{ along } [1,0]) \\
&= \mathbf{b}_0 - [1,0] \\
&= [0,1] \\
\mathbf{b}_2 &= \mathbf{b}_1 - (\text{projection of } [0,1] \text{ along } [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]) \\
&= \mathbf{b}_1 - [\frac{1}{2}, \frac{1}{2}] \\
&= [-\frac{1}{2}, \frac{1}{2}] \text{ which is not orthogonal to } [1,0]
\end{aligned}
$$

How to repair project_orthogonal(b, vlist)?

```
def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

$\mathbf{b} = [1,1]$

vlist $= [\ [1,0], [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]\ ]$

Final vector is not orthogonal to $[1,0]$

Maybe the problem will go away if the algorithm

- first finds the projection of **b** along each of the vectors in vlist, and
- only afterwards subtracts all these projections from **b**.

```
def classical_project_orthogonal(b, vlist):
  w = all-zeroes-vector
  for v in vlist:
    w = w + project_along(b, v)
  return b - w
```

Alas, this procedure also does not work. For the inputs

$$\mathbf{b} = [1,1], \texttt{vlist} = [\ [1,0], [\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]\ ]$$

the output vector is $[-1,0]$

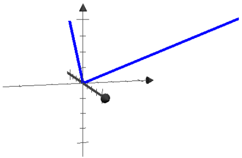
which is orthogonal to neither of the two vectors in vlist.

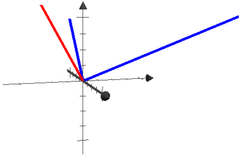
What to do with `project_orthogonal(b, vlist)`?

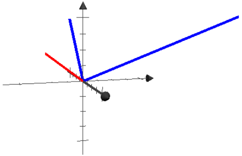
Try it with two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ that are orthogonal...

$$
\begin{aligned}
\mathbf{v}_1 &= [1, 2, 1] \\
\mathbf{v}_2 &= [-1, 2, -1] \\
\mathbf{b} &= [1, 1, 2] \\
\mathbf{b}_1 &= \mathbf{b}_0 - \frac{\mathbf{b}_0 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1} \mathbf{v}_1 \\
&= [1, 1, 2] - \frac{5}{6} [1, 2, 1] \\
&= \left[\frac{1}{6}, -\frac{4}{6}, \frac{7}{6}\right] \\
\mathbf{b}_2 &= \mathbf{b}_1 - \frac{\mathbf{b}_1 \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2} \mathbf{v}_2 \\
&= \left[\frac{1}{6}, -\frac{4}{6}, \frac{7}{6}\right] - \frac{1}{2} [-1, 0, 1] \\
&= \left[\frac{2}{3}, -\frac{2}{3}, \frac{2}{3}\right]
\end{aligned}
$$

and note $\mathbf{b}_2$ is orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$.





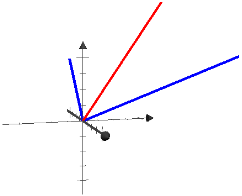


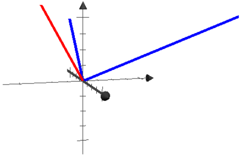
What to do with `project_orthogonal(b, vlist)`?

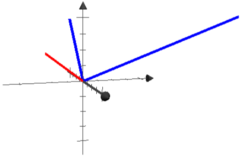
Try it with two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ that are orthogonal...

$$
\begin{aligned}
\mathbf{v}_1 &= [1, 2, 1] \\
\mathbf{v}_2 &= [-1, 2, -1] \\
\mathbf{b} &= [1, 1, 2] \\
\mathbf{b}_1 &= \mathbf{b}_0 - \frac{\mathbf{b}_0 \cdot \mathbf{v}_1}{\mathbf{v}_1 \cdot \mathbf{v}_1}\,\mathbf{v}_1 \\
&= [1, 1, 2] - \frac{5}{6}\,[1, 2, 1] \\
&= \left[\frac{1}{6}, -\frac{4}{6}, \frac{7}{6}\right] \\
\mathbf{b}_2 &= \mathbf{b}_1 - \frac{\mathbf{b}_1 \cdot \mathbf{v}_2}{\mathbf{v}_2 \cdot \mathbf{v}_2}\,\mathbf{v}_2 \\
&= \left[\frac{1}{6}, -\frac{4}{6}, \frac{7}{6}\right] - \frac{1}{2}\,[-1, 0, 1] \\
&= \left[\frac{2}{3}, -\frac{2}{3}, \frac{2}{3}\right]
\end{aligned}
$$

and note $\mathbf{b}_2$ is orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$.







Maybe `project_orthogonal(b, vlist)` works with $\mathbf{v}_1, \mathbf{v}_2$ orthogonal?

Assume $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle = 0$.
Remember: $\mathbf{b}_i$ is value of `b` after i iterations

First iteration:

$$\mathbf{b}_1 = \mathbf{b}_0 - \sigma_1 \, \mathbf{v}_1$$

gives $\mathbf{b}_1$ such that $\langle \mathbf{v}_1, \mathbf{b}_1 \rangle = 0$.

Second iteration:

$$\mathbf{b}_2 = \mathbf{b}_1 - \sigma_1 \, \mathbf{v}_2$$

gives $\mathbf{b}_2$ such that $\langle \mathbf{v}_2, \mathbf{b}_2 \rangle = 0$

But what about $\langle \mathbf{v}_1, \mathbf{b}_2 \rangle$?

$$\begin{aligned}
\langle \mathbf{v}_1, \mathbf{b}_2 \rangle &= \langle \mathbf{v}_1, \mathbf{b}_1 - \sigma \ \mathbf{v}_2 \rangle \\
&= \langle \mathbf{v}_1, \mathbf{b}_1 \rangle - \langle \mathbf{v}_1, \sigma \ \mathbf{v}_2 \rangle \\
&= \langle \mathbf{v}_1, \mathbf{b}_1 \rangle - \sigma \ \langle \mathbf{v}_1, \mathbf{v}_2 \rangle \\
&= 0 + 0
\end{aligned}$$

Thus $\mathbf{b}_2$ is orthogonal to $\mathbf{v}_1$ and $\mathbf{v}_2$

Don't fix `project_orthogonal(b, vlist)`. Fix the spec.

```
def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

Instead of trying to fix the flaw by changing the procedure, we will change the spec we expect the procedure to fulfill.

Require that `vlist` consists of **mutually orthogonal** vectors:
the i^{th} vector in the list is orthogonal to the j^{th} vector in the list for every $i \neq j$.

New spec:

- *input:* a vector $\mathbf{b}$, and a list `vlist` of *mutually orthogonal* vectors
- *output:* the projection $\mathbf{b}^{\perp}$ of $\mathbf{b}$ orthogonal to the vectors in `vlist`

Loop invariant of project_orthogonal(b, vlist)

```
def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

Loop invariant: Let vlist $= [\mathbf{v}_1, \ldots, \mathbf{v}_n]$
For $i = 0, \ldots, n$, let $\mathbf{b}_i$ be the value of the variable b after i iterations. Then $\mathbf{b}_i$ is the projection of $\mathbf{b}$ orthogonal to Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_i\}$. That is,

- $\mathbf{b}_i$ is orthogonal to the first i vectors of vlist, and
- $\mathbf{b} - \mathbf{b}_i$ is in the span of the first i vectors of vlist

We use induction to prove the invariant holds.

For $i = 0$, the invariant is trivially true:

- $\mathbf{b}_0$ is orthogonal to each of the first 0 vectors (every vector is), and
- $\mathbf{b} - \mathbf{b}_0$ is in the span of the first 0 vectors (because $\mathbf{b} - \mathbf{b}_0$ is the zero vector).

Proof of loop invariant of `project_orthogonal(b, `$[\mathbf{v}_1, \ldots, \mathbf{v}_n]$` )`

$\mathbf{b}_i$ = projection of $\mathbf{b}$ orthogonal to Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_i\}$:

- $\mathbf{b}_i$ is orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_i$, and
- $\mathbf{b} - \mathbf{b}_i$ is in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_i\}$

```
for v in vlist:
  b = b - project_along(b, v)
```

Assume invariant holds for $i = k-1$ iterations, and prove it for $i = k$ iterations.

In k^{th} iteration, algorithm computes $\mathbf{b}_k = \mathbf{b}_{k-1} - \sigma_k \, \mathbf{v}_k$
By induction hypothesis, $\mathbf{b}_{k-1}$ is the projection of $\mathbf{b}$ orthogonal to Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_{k-1}\}$
Must prove

- $\mathbf{b}_k$ is orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_k$, $\checkmark$
- and $\mathbf{b} - \mathbf{b}_k$ is in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ $\checkmark$

Choice of σ_k ensures that $\mathbf{b}_k$ is orthogonal to $\mathbf{v}_k$.
Must show $\mathbf{b}_k$ also orthogonal to $\mathbf{v}_j$ for $j = 1, \ldots, k-1$

$$
\begin{aligned}
\langle \mathbf{b}_k, \mathbf{v}_j \rangle &= \langle \mathbf{b}_{k-1} - \sigma_k \mathbf{v}_k, \mathbf{v}_j \rangle \\
&= \langle \mathbf{b}_{k-1}, \mathbf{v}_j \rangle - \sigma_k \, \langle \mathbf{v}_k, \mathbf{v}_j \rangle \\
&= 0 - \sigma_k \, \langle \mathbf{v}_k, \mathbf{v}_j \rangle && \text{by the inductive hypothesis} \\
&= 0 - \sigma_k 0 && \text{by mutual orthogonality}
\end{aligned}
$$

Shows $\mathbf{b}_k$ orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_k$

Correctness of `project_orthogonal(b, vlist)`

```
def project_orthogonal(b, vlist):
    for v in vlist:
        b = b - project_along(b, v)
    return b
```

We have proved:

If $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are mutually orthogonal then
output of $\texttt{project\_orthogonal}(\mathbf{b}, [\mathbf{v}_1, \ldots, \mathbf{v}_n])$ is the vector $\mathbf{b}^{\perp}$ such that

- $\mathbf{b} = \mathbf{b}^{||} + \mathbf{b}^{\perp}$
- $\mathbf{b}^{||}$ is in Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$
- $\mathbf{b}^{\perp}$ is orthogonal to $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

Change to zero-based indexing::

If $\mathbf{v}_0, \ldots, \mathbf{v}_n$ are mutually orthogonal then
output of $\texttt{project\_orthogonal}(\mathbf{b}, [\mathbf{v}_0, \ldots, \mathbf{v}_n])$ is the vector $\mathbf{b}^{\perp}$ such that

- $\mathbf{b} = \mathbf{b}^{||} + \mathbf{b}^{\perp}$
- $\mathbf{b}^{||}$ is in Span $\{\mathbf{v}_0, \ldots, \mathbf{v}_n\}$
- $\mathbf{b}^{\perp}$ is orthogonal to $\mathbf{v}_0, \ldots, \mathbf{v}_n$.

Augmenting `project_orthogonal`

Since $\mathbf{b}^{||} = \mathbf{b} - \mathbf{b}^{\perp}$ is in Span $\{\mathbf{v}_0, \ldots, \mathbf{v}_n\}$, there are coefficients $\alpha_0, \ldots, \alpha_n$ such that

$$\mathbf{b} - \mathbf{b}^{\perp} = \alpha_0\,\mathbf{v}_0 + \cdots + \alpha_n\,\mathbf{v}_n$$

$$\mathbf{b} = \alpha_0\,\mathbf{v}_0 + \cdots + \alpha_n\,\mathbf{v}_n + 1\,\mathbf{b}^{\perp}$$

Write as

$$\left[\begin{array}{c} \mathbf{b} \end{array}\right] = \left[\begin{array}{c|c|c|c} \mathbf{v}_0 & \cdots & \mathbf{v}_n & \mathbf{b}^{\perp} \end{array}\right] \left[\begin{array}{c} \alpha_0 \\ \vdots \\ \alpha_n \\ 1 \end{array}\right]$$

The procedure `project_orthogonal(b, vlist)` can be augmented to output the vector of coefficients.

For technical reasons, we will represent the vector of coefficents as a dictionary, not a `Vec`.

Augmenting `project_orthogonal`

$$\begin{bmatrix} \mathbf{b} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_0 & \cdots & \mathbf{v}_n & \mathbf{b}^\perp \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_n \\ 1 \end{bmatrix}$$

We reuse code from two prior procedures.

```
def project_along(b, v):
 sigma = ((b*v)/(v*v)) \
      if v*v != 0 else 0
 return sigma * v

def project_orthogonal(b, vlist):
  for v in vlist:
     b = b - project_along(b, v)
  return b
```

Must create and populate a dictionary.

- One entry for each vector in `vlist`
- One additional entry, 1, for $\mathbf{b}^\perp$

Initialize dictionary with the additonal entry.

```
def aug_project_orthogonal(b, vlist):
 alphadict = {len(vlist):1}
 for i in range(len(vlist)):
   v = vlist[i]
   sigma = (b*v)/(v*v) \
           if v*v > 0 else 0
   alphadict[i] = sigma
   b = b - sigma*v
 return (b, alphadict)
```

Building an orthogonal set of generators

Original stated goal:

Find the projection of $\mathbf{b}$ orthogonal to the space $\mathcal{V}$ spanned by arbitrary vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

So far we know how to find the projection of $\mathbf{b}$ orthogonal to the space spanned by mutually orthogonal vectors.

This would suffice if we had a procedure that, given arbitrary vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, computed mutually orthogonal vectors $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ that span the same space.

We consider a new problem: *orthogonalization*:

- *input:* A list $[\mathbf{v}_1, \ldots, \mathbf{v}_n]$ of vectors over the reals
- *output:* A list of mutually orthogonal vectors $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ such that

$$\text{Span } \{\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*\} = \text{Span } \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$$

How can we solve this problem?

The `orthogonalize` procedure

Idea: Use `project_orthogonal` iteratively to make a longer and longer list of mutually orthogonal vectors.

- First consider $\mathbf{v}_1$. Define $\mathbf{v}_1^* := \mathbf{v}_1$ since the set $\{\mathbf{v}_1^*\}$ is trivially a set of mutually orthogonal vectors.
- Next, define $\mathbf{v}_2^*$ to be the projection of $\mathbf{v}_2$ orthogonal to $\mathbf{v}_1^*$.
- Now $\{\mathbf{v}_1^*, \mathbf{v}_2^*\}$ is a set of mutually orthogonal vectors.
- Next, define $\mathbf{v}_3^*$ to be the projection of $\mathbf{v}_3$ orthogonal to $\mathbf{v}_1^*$ and $\mathbf{v}_2^*$, so $\{\mathbf{v}_1^*, \mathbf{v}_2^*, \mathbf{v}_3^*\}$ is a set of mutually orthogonal vectors....

In each step, we use `project_orthogonal` to find the next orthogonal vector.

In the i^{th} iteration, we project $\mathbf{v}_i$ orthogonal to $\mathbf{v}_1^*, \ldots, \mathbf{v}_{i-1}^*$ to find $\mathbf{v}_i^*$.

```
def orthogonalize(vlist):
  vstarlist = []
  for v in vlist:
    vstarlist.append(project_orthogonal(v, vstarlist))
  return vstarlist
```

Correctness of the `orthogonalize` procedure, Part I

```
def orthogonalize(vlist):
  vstarlist = []
  for v in vlist:
    vstarlist.append(project_orthogonal(v, vstarlist))
  return vstarlist
```

Lemma: Throughout the execution of `orthogonalize`, the vectors in `vstarlist` are mutually orthogonal.

In particular, the list `vstarlist` at the end of the execution, which is the list returned, consists of mutually orthogonal vectors.

Proof: by induction, using the fact that each vector added to `vstarlist` is orthogonal to all the vectors already in the list. QED

Example of `orthogonalize`

Example: When `orthogonalize` is called on a `vlist` consisting of vectors

$$\mathbf{v}_1 = [2, 0, 0], \mathbf{v}_2 = [1, 2, 2], \mathbf{v}_3 = [1, 0, 2]$$

it returns the list `vstarlist` consisting of

$$\mathbf{v}_1^* = [2, 0, 0], \mathbf{v}_2^* = [0, 2, 2], \mathbf{v}_3^* = [0, -1, 1]$$

(1) In the first iteration, when `v` is $\mathbf{v}_1$, `vstarlist` is empty, so the first vector $\mathbf{v}_1^*$ added to `vstarlist` is $\mathbf{v}_1$ itself.

(2) In the second iteration, when `v` is $\mathbf{v}_2$, `vstarlist` consists only of $\mathbf{v}_1^*$. The projection of $\mathbf{v}_2$ orthogonal to $\mathbf{v}_1^*$ is

$$\begin{aligned} \mathbf{v}_2 - \frac{\langle \mathbf{v}_2, \mathbf{v}_1^* \rangle}{\langle \mathbf{v}_1^*, \mathbf{v}_1^* \rangle} \mathbf{v}_1^* &= [1, 2, 2] - \frac{2}{4}[2, 0, 0] \\ &= [0, 2, 2] \end{aligned}$$

so $\mathbf{v}_2^* = [0, 2, 2]$ is added to `vstarlist`.

(3) In the third iteration, when `v` is $\mathbf{v}_3$, `vstarlist` consists of $\mathbf{v}_1^*$ and $\mathbf{v}_2^*$. The projection of $\mathbf{v}_3$ orthogonal to $\mathbf{v}_1^*$ is $[0, 0, 2]$, and the projection of $[0, 0, 2]$ orthogonal to $\mathbf{v}_2^*$ is

$$[0, 0, 2] - \frac{1}{2}[0, 2, 2] = [0, -1, 1]$$

so $\mathbf{v}_3^* = [0, -1, 1]$ is added to `vstarlist`

Correctness of the orthogonalize procedure, Part II

Lemma: Consider orthogonalize applied to an n-element list $[\mathbf{v}_1, \ldots, \mathbf{v}_n]$. After i iterations of the algorithm, Span vstarlist $=$ Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_i\}$.

Proof: by induction on i.
The case $i = 0$ is trivial.
After $i-1$ iterations, vstarlist consists of vectors $\mathbf{v}_1^*, \ldots, \mathbf{v}_{i-1}^*$.

Assume the lemma holds at this point. This means that

$$\text{Span }\{\mathbf{v}_1^*, \ldots, \mathbf{v}_{i-1}^*\} = \text{Span }\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}\}$$

By adding the vector $\mathbf{v}_i$ to sets on both sides, we obtain

$$\text{Span }\{\mathbf{v}_1^*, \ldots, \mathbf{v}_{i-1}^*, \mathbf{v}_i\} = \text{Span }\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}, \mathbf{v}_i\}$$

... It therefore remains only to show that

Span $\{\mathbf{v}_1^*, \ldots, \mathbf{v}_{i-1}^*, \mathbf{v}_i^*\} =$ Span $\{\mathbf{v}_1^*, \ldots, \mathbf{v}_{i-1}^*, \mathbf{v}_i\}$.

The i^{th} iteration computes $\mathbf{v}_i^*$ using project_orthogonal$(\mathbf{v}_i, [\mathbf{v}_1^*, \ldots, \mathbf{v}_{i-1}^*])$.
There are scalars $\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{i,i-1}$ such that

$$\mathbf{v}_i = \alpha_{1i}\mathbf{v}_1^* + \cdots + \alpha_{i-1,i}\mathbf{v}_{i-1}^* + \mathbf{v}_i^*$$

This equation shows that any linear combination of

Order in `orthogonalize`

Order matters!

Suppose you run the procedure `orthogonalize` twice, once with a list of vectors and once with the reverse of that list.

The output lists will **not** be the reverses of each other.

Contrast with `project_orthogonal(b, vlist)`.

The projection of a vector **b** orthogonal to a vector space is unique,
so in principle the order of vectors in `vlist` doesn't affect the output of `project_orthogonal(b, vlist)`.

Matrix form for `orthogonalize`

For `project_orthogonal`, we had $\left[\begin{array}{c} \mathbf{b} \end{array}\right] = \left[\begin{array}{c|c|c|c} \mathbf{v}_0 & \cdots & \mathbf{v}_n & \mathbf{b}^\perp \end{array}\right] \left[\begin{array}{c} \alpha_0 \\ \vdots \\ \alpha_n \\ 1 \end{array}\right]$

For `orthogonalize`, we have

$$\left[\begin{array}{c} \mathbf{v}_0 \end{array}\right] = \left[\begin{array}{c} \mathbf{v}_0^* \end{array}\right] \left[\begin{array}{c} 1 \end{array}\right] \qquad \left[\begin{array}{c|c|c|c} \mathbf{v}_0 & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \end{array}\right] = \left[\begin{array}{c|c|c|c} \mathbf{v}_0^* & \mathbf{v}_1^* & \mathbf{v}_2^* & \mathbf{v}_3^* \end{array}\right] \left[\begin{array}{cccc} 1 & \alpha_{01} & \alpha_{02} & \alpha_{03} \\ & 1 & \alpha_{12} & \alpha_{13} \\ & & 1 & \alpha_{23} \\ & & & 1 \end{array}\right]$$

$$\left[\begin{array}{c} \mathbf{v}_1 \end{array}\right] = \left[\begin{array}{c|c} \mathbf{v}_0^* & \mathbf{v}_1^* \end{array}\right] \left[\begin{array}{c} \alpha_{01} \\ 1 \end{array}\right]$$

$$\left[\begin{array}{c} \mathbf{v}_2 \end{array}\right] = \left[\begin{array}{c|c|c} \mathbf{v}_0^* & \mathbf{v}_1^* & \mathbf{v}_2^* \end{array}\right] \left[\begin{array}{c} \alpha_{02} \\ \alpha_{12} \\ 1 \end{array}\right]$$

$$\left[\begin{array}{c} \mathbf{v}_3 \end{array}\right] = \left[\begin{array}{c|c|c|c} \mathbf{v}_0^* & \mathbf{v}_1^* & \mathbf{v}_2^* & \mathbf{v}_3^* \end{array}\right] \left[\begin{array}{c} \alpha_{03} \\ \alpha_{13} \\ \alpha_{23} \\ 1 \end{array}\right]$$

Example of matrix form for `orthogonalize`

Example: for `vlist` consisting of vectors

$$\mathbf{v}_0 = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \mathbf{v}_1 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$$

we saw that the output list `vstarlist` of orthogonal vectors consists of

$$\mathbf{v}_0^* = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \mathbf{v}_1^* = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}, \mathbf{v}_2^* = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$$

The corresponding matrix equation is

$$\left[\begin{array}{c|c|c} \mathbf{v}_0 & \mathbf{v}_1 & \mathbf{v}_2 \end{array}\right] = \left[\begin{array}{c|c|c} 2 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & 2 & 1 \end{array}\right] \begin{bmatrix} 1 & 0.5 & 0.5 \\ & 1 & 0.5 \\ & & 1 \end{bmatrix}$$

Solving *closest point in the span of many vectors*

Let $\mathcal{V} = \text{Span }\{\mathbf{v}_0, \ldots, \mathbf{v}_n\}$.

The vector in $\mathcal{V}$ closest to $\mathbf{b}$ is $\mathbf{b}^{||\mathcal{V}}$, which is $\mathbf{b} - \mathbf{b}^{\perp\mathcal{V}}$.

There are two equivalent ways to find $\mathbf{b}^{\perp\mathcal{V}}$,

- *One method:*

  Step 1: Apply `orthogonalize` to $\mathbf{v}_0, \ldots, \mathbf{v}_n$, and obtain $\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*$.
  (Now $\mathcal{V} = \text{Span }\{\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*\}$)

  Step 2: Call `project_orthogonal`$(\mathbf{b}, [\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*])$
  and obtain $\mathbf{b}^{\perp}$ as the result.

- *Another method:* Exactly the same computations take place when `orthogonalize` is applied to $[\mathbf{v}_0, \ldots, \mathbf{v}_n, \mathbf{b}]$ to obtain $[\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*, \mathbf{b}^*]$.

  In the last iteration of `orthogonalize`, the vector $\mathbf{b}^*$ is obtained by projecting $\mathbf{b}$ orthogonal to $\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*$. Thus $\mathbf{b}^* = \mathbf{b}^{\perp}$.

Mutually orthogonal nonzero vectors are linearly independent

Proposition: Mutually orthogonal nonzero vectors are linearly independent.

Proof: Let $\mathbf{v}_0^*, \mathbf{v}_2^*, \ldots, \mathbf{v}_n^*$ be mutually orthogonal nonzero vectors.
Suppose $\alpha_0, \alpha_1, \ldots, \alpha_n$ are coefficients such that

$$\mathbf{0} = \alpha_0 \, \mathbf{v}_0^* + \alpha_1 \, \mathbf{v}_1^* + \cdots + \alpha_n \, \mathbf{v}_n^*$$

We must show that therefore the coefficients are all zero.
To show that α_0 is zero, take inner product with $\mathbf{v}_0^*$ on both sides:

$$\begin{aligned}
\langle \mathbf{v}_0^*, \mathbf{0} \rangle &= \langle \mathbf{v}_0^*, \alpha_0 \, \mathbf{v}_0^* + \alpha_1 \, \mathbf{v}_1^* + \cdots + \alpha_n \, \mathbf{v}_n^* \rangle \\
&= \alpha_0 \, \langle \mathbf{v}_0^*, \mathbf{v}_0^* \rangle + \alpha_1 \, \langle \mathbf{v}_0^*, \mathbf{v}_1^* \rangle + \cdots + \alpha_n \, \langle \mathbf{v}_0^*, \mathbf{v}_n^* \rangle \\
&= \alpha_0 \|\mathbf{v}_0^*\|^2 + \alpha_1 \, 0 + \cdots + \alpha_n \, 0 \\
&= \alpha_0 \|\mathbf{v}_0^*\|^2
\end{aligned}$$

The inner product $\langle \mathbf{v}_0^*, 0 \rangle$ is zero, so $\alpha_0 \, \|\mathbf{v}_0^*\|^2 = 0$. Since $\mathbf{v}_0^*$ is nonzero, its norm is nonzero, so the only solution is $\alpha_0 = 0$.
Can similarly show that $\alpha_1 = \cdots = \alpha_n = 0$. QED

Computing a basis

Proposition: Mutually orthogonal nonzero vectors are linearly independent.

What happens if we call the `orthogonalize` procedure on a list `vlist`$=[\mathbf{v}_0, \ldots, \mathbf{v}_n]$ of vectors that are linearly dependent?

$\dim \text{Span } \{\mathbf{v}_0, \ldots, \mathbf{v}_n\} < n+1$.

`orthogonalize(`$[\mathbf{v}_0, \ldots, \mathbf{v}_n]$`)` returns $[\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*]$

The vectors $\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*$ are mutually orthogonal.

They can't be linearly independent since they span a space of dimension less than $n+1$.

Therefore some of them must be zero vectors.

Leaving out the zero vectors does not change the space spanned...

Let S be the subset of $\{\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*\}$ consisting of nonzero vectors.

Span $S =$ Span $\{\mathbf{v}_0^*, \ldots, \mathbf{v}_n^*\} =$ Span $\{\mathbf{v}_0, \ldots, \mathbf{v}_n\}$

Proposition implies that S is linearly independent.

Thus S is a basis for Span $\{\mathbf{v}_0, \ldots, \mathbf{v}_n\}$.

Orthogonal complement

Let $\mathcal{U}$ be a subspace of $\mathcal{W}$.
For each vector $\mathbf{b}$ in $\mathcal{W}$, we can write $\mathbf{b} = \mathbf{b}^{||\mathcal{U}} + \mathbf{b}^{\perp\mathcal{U}}$ where

- $\mathbf{b}^{||\mathcal{U}}$ is in $\mathcal{U}$, and
- $\mathbf{b}^{\perp\mathcal{U}}$ is orthogonal to every vector in $\mathcal{U}$.

Let $\mathcal{V}$ be the set $\{\mathbf{b}^{\perp\mathcal{U}} \;:\; \mathbf{b} \in \mathcal{W}\}$.
Definition: We call $\mathcal{V}$ the *orthogonal complement* of $\mathcal{U}$ in $\mathcal{W}$

Easy observations:

- Every vector in $\mathcal{V}$ is orthogonal to every vector in $\mathcal{U}$.
- Every vector $\mathbf{b}$ in $\mathcal{W}$ can be written as the sum of a vector in $\mathcal{U}$ and a vector in $\mathcal{V}$.

Maybe $\mathcal{U} \oplus \mathcal{V} = \mathcal{W}$? To show direct sum of $\mathcal{U}$ and $\mathcal{V}$ is defined, we need to show that the only in vector that is in both $\mathcal{U}$ and $\mathcal{V}$ is the zero vector.

Any vector $\mathbf{w}$ in both $\mathcal{U}$ and $\mathcal{V}$ is orthogonal to itself.
Thus $0 = \langle \mathbf{w}, \mathbf{w} \rangle = \|\mathbf{w}\|^2$.
By Property N2 of norms, that means $\mathbf{w} = \mathbf{0}$.

Therefore $\mathcal{U} \oplus \mathcal{V} = \mathcal{W}$. **Recall:** $\dim \mathcal{U} + \dim \mathcal{V} = \dim \mathcal{U} \oplus \mathcal{V}$

Orthogonal complement: example

Example: Let $\mathcal{U} = \text{Span}\ \{[1,1,0,0],[0,0,1,1]\}$. Let $\mathcal{V}$ denote the orthogonal complement of $\mathcal{U}$ in $\mathbb{R}^4$. What vectors form a basis for $\mathcal{V}$?

Every vector in $\mathcal{U}$ has the form $[a, a, b, b]$.

Therefore any vector of the form $[c, -c, d, -d]$ is orthogonal to every vector in $\mathcal{U}$.

Every vector in Span $\{[1,-1,0,0],[0,0,1,-1]\}$ is orthogonal to every vector in $\mathcal{U}$....
... so Span $\{[1,-1,0,0],[0,0,1,-1]\}$ is a subspace of $\mathcal{V}$, the orthogonal complement of $\mathcal{U}$ in $\mathbb{R}^4$.

Is it the whole thing?

$\mathcal{U} \oplus \mathcal{V} = \mathbb{R}^4$ so $\dim \mathcal{U} + \dim \mathcal{V} = 4$.

$\{[1,1,0,0],[0,0,1,1]\}$ is linearly independent so $\dim \mathcal{U} = 2$... so $\dim \mathcal{V} = 2$

$\{[1,-1,0,0],[0,0,1,-1]\}$ is linearly independent
so dim Span $\{[1,-1,0,0],[0,0,1,-1]\}$ is also 2....
so Span $\{[1,-1,0,0],[0,0,1,-1]\} = \mathcal{V}$.

Computing the orthogonal complement

Suppose we have a basis $\mathbf{u}_1, \ldots, \mathbf{u}_k$ for $\mathcal{U}$ and a basis $\mathbf{w}_1, \ldots, \mathbf{w}_n$ for $\mathcal{W}$. How can we compute a basis for the orthogonal complement of $\mathcal{U}$ in $\mathcal{W}$?

One way: use `orthogonalize(vlist)` with

$$\text{vlist} = [\mathbf{u}_1, \ldots, \mathbf{u}_k, \mathbf{w}_1, \ldots, \mathbf{w}_n]$$

Write list returned as $[\mathbf{u}_1^*, \ldots, \mathbf{u}_k^*, \mathbf{w}_1^*, \ldots, \mathbf{w}_n^*]$

These span the same space as input vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k, \mathbf{w}_1, \ldots, \mathbf{w}_n^*$, namely $\mathcal{W}$, which has dimension n.

Therefore exactly n of the output vectors $\mathbf{u}_1^*, \ldots, \mathbf{u}_k^*, \mathbf{w}_1^*, \ldots, \mathbf{w}_n^*$ are nonzero.

The vectors $\mathbf{u}_1^*, \ldots, \mathbf{u}_k^*$ have same span as $\mathbf{u}_1, \ldots, \mathbf{u}_k$ and are all nonzero since $\mathbf{u}_1, \ldots, \mathbf{u}_k$ are linearly independent.

Therefore exactly $n - k$ of the remaining vectors $\mathbf{w}_1^*, \ldots, \mathbf{w}_n^*$ are nonzero.

Every one of them is orthogonal to $\mathbf{u}_1, \ldots, \mathbf{u}_n$...
so they are orthogonal to every vector in $\mathcal{U}$...
so they lie in the orthogonal complement of $\mathcal{U}$.
By Direct-Sum Dimension Lemma, orthogonal complement has dimension $n - k$, so the remaining nonzero vectors are a basis for the orthogonal complement.

Augmenting orthogonalize(vlist)

We will write a procedure aug_orthogonalize(vlist) with the following spec:

- *input:* a list $[\mathbf{v}_1, \ldots, \mathbf{v}_n]$ of vectors
- *output:* the pair $([\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*], [\mathbf{r}_1, \ldots, \mathbf{r}_n])$ of lists of vectors such that $\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*$ are mutually orthogonal vectors whose span equals Span $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$, and

$$\left[\begin{array}{c|c|c} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{array}\right] = \left[\begin{array}{c|c|c} \mathbf{v}_1^* & \cdots & \mathbf{v}_n^* \end{array}\right] \left[\begin{array}{c|c|c} \mathbf{r}_1 & \cdots & \mathbf{r}_n \end{array}\right]$$

```
def orthogonalize(vlist):
  vstarlist = []
  for v in vlist:
    vstarlist.append(
      project_orthogonal(v, vstarlist))
  return vstarlist
```

```
def aug_orthogonalize(vlist):
  vstarlist = []
  r_vecs = []
  D = set(range(len(vlist)))
  for v in vlist:
    (vstar, alphadict) =
       aug_project_orthogonal(v, vstarlist)
    vstarlist.append(vstar)
    r_vecs.append(Vec(D, alphadict))
  return vstarlist, r_vecs
```

Towards QR factorization

We will now develop the *QR factorization*. We will show that certain matrices can be written as the product of matrices in special form.
Matrix factorizations are useful mathematically and computationally:

- *Mathematical:* They provide insight into the nature of matrices—each factorization gives us a new way to think about a matrix.
- *Computational:* They give us ways to compute solutions to fundamental computational problems involving matrices.

Matrices with mutually orthogonal columns

$$\begin{bmatrix} \mathbf{v}_1^{*\,T} \\ \hline \vdots \\ \hline \mathbf{v}_n^{*\,T} \end{bmatrix} \left[\begin{array}{c|c|c} \\ \mathbf{v}_1^* & \cdots & \mathbf{v}_n^* \\ \\ \end{array}\right] = \begin{bmatrix} \|\mathbf{v}_1\|^2 & & \\ & \ddots & \\ & & \|\mathbf{v}_n\|^2 \end{bmatrix}$$

Cross-terms are zero because of mutual orthogonality.
To make the product into the identity matrix, can *normalize* the columns.

Normalizing a vector means scaling it to make its norm 1.

Just divide it by its norm.

```
>>> def normalize(v): return v/sqrt(v*v)
>>> q = normalize(list2vec[1,1,1])
>>> q * q
1.0000000000000002
>>> print(q)
     0     1     2
------------------
 0.577 0.577 0.577
```

Matrices with mutually orthogonal columns

$$\left[\begin{array}{c} {\mathbf{v}_1^*}^T \\ \hline \vdots \\ \hline {\mathbf{v}_n^*}^T \end{array}\right] \left[\begin{array}{c|c|c} & & \\ \mathbf{v}_1^* & \cdots & \mathbf{v}_n^* \\ & & \end{array}\right] = \begin{bmatrix} \|\mathbf{v}_1\|^2 & & \\ & \ddots & \\ & & \|\mathbf{v}_n\|^2 \end{bmatrix}$$

Cross-terms are zero because of mutual orthogonality.

To make the product into the identity matrix, can *normalize* the columns.

Normalize columns $\left[\begin{array}{c|c|c} & & \\ \mathbf{v}_1^* & \cdots & \mathbf{v}_n^* \\ & & \end{array}\right] \Rightarrow \left[\begin{array}{c|c|c} & & \\ \mathbf{q}_1 & \cdots & \mathbf{q}_n \\ & & \end{array}\right]$

Matrices with mutually orthogonal columns

$$\begin{bmatrix} \mathbf{q}_1^T \\ \hline \vdots \\ \hline \mathbf{q}_n^T \end{bmatrix} \left[\begin{array}{c|c|c} & & \\ \mathbf{q}_1 & \cdots & \mathbf{q}_n \\ & & \end{array}\right] = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$$

Normalize columns $\left[\begin{array}{c|c|c} & & \\ \mathbf{v}_1^* & \cdots & \mathbf{v}_n^* \\ & & \end{array}\right] \Rightarrow \left[\begin{array}{c|c|c} & & \\ \mathbf{q}_1 & \cdots & \mathbf{q}_n \\ & & \end{array}\right]$

Matrices with mutually orthogonal columns

$$\begin{bmatrix} \mathbf{q}_1^T \\ \hline \vdots \\ \hline \mathbf{q}_n^T \end{bmatrix} \left[\begin{array}{c|c|c} & & \\ \mathbf{q}_1 & \cdots & \mathbf{q}_n \\ & & \end{array}\right] = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$$

Proposition: If columns of Q are mutually orthogonal with norm 1 then $Q^T Q$ is identity matrix.

Definition: Vectors that are mutually orthogonal and have norm 1 are *orthonormal*.

Definition: If columns of Q are orthonormal then we call Q a *column-orthogonal* matrix.
should be called *orthonormal* but oh well

Definition: If Q is square and column-orthogonal, we call Q an *orthogonal* matrix.

Proposition: If Q is an orthogonal matrix then its inverse is Q^T.

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

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

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

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

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 $R = \mathbf{c}$ 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).

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

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

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

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.

Goal: An algorithm that, given equation $A\mathbf{x} = \mathbf{b}$, where columns are linearly independent, 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 the vector $\hat{\mathbf{x}}$ such that $A\hat{\mathbf{x}} = \mathbf{b}^{||\mathcal{V}}$.

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

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

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

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.

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 $\mathbf{b}^{||\mathcal{V}}$.

Representation of $\mathbf{b}^{||}$ in terms of columns of Q

Let Q be a column-orthogonal matrix. Let $\mathbf{b}$ be a vector, and write $\mathbf{b} = \mathbf{b}^{||} + \mathbf{b}^{\perp}$ where $\mathbf{b}^{||}$ is projection of $\mathbf{b}$ onto Col Q and $\mathbf{b}^{\perp}$ is projection orthogonal to Col Q.

Let $\mathbf{u}$ be the coordinate representation of $\mathbf{b}^{||}$ in terms of columns of Q.

By linear-combinations definition of matrix-vector multiplication,

$$\left[\begin{array}{c} \\ \mathbf{b}^{||} \\ \\ \end{array}\right] = \left[\begin{array}{ccc} & & \\ & Q & \\ & & \end{array}\right] \left[\begin{array}{c} \mathbf{u} \end{array}\right]$$

Multiply both sides on the left by Q^T:

$$\left[\begin{array}{ccc} & Q^T & \end{array}\right] \left[\begin{array}{c} \\ \mathbf{b}^{||} \\ \\ \end{array}\right] = \left[\begin{array}{ccc} & Q^T & \end{array}\right] \left[\begin{array}{ccc} & & \\ & Q & \\ & & \end{array}\right] \left[\begin{array}{c} \mathbf{u} \end{array}\right]$$

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

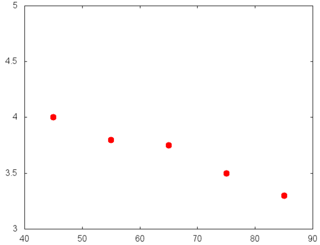
Application of least squares: linear regression

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 $(x_1, y_1) = (45, 4)$, $(x_2, y_2) = (55, 3.8)$, $(x_3, y_3) = (64, 3.75)$,$(x_4, y_4) = (75, 3.5)$, $(x_5, y_5) = (85, 3.3)$.
The prediction error on the 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.

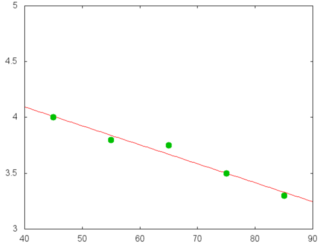
Application of least squares: linear regression

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 $(x_1, y_1) = (45, 4)$, $(x_2, y_2) = (55, 3.8)$, $(x_3, y_3) = (64, 3.75)$,$(x_4, y_4) = (75, 3.5)$, $(x_5, y_5) = (85, 3.3)$.
The prediction error on the 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.

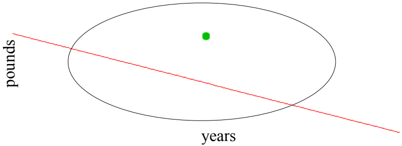
Hypothesis: after age 45, brain mass decreases linearly with age, i.e. that $f(x) = mx + b$ for some numbers m, b.

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

The observations are $(x_1, y_1) = (45, 4)$, $(x_2, y_2) = (55, 3.8)$, $(x_3, y_3) = (64, 3.75)$,$(x_4, y_4) = (75, 3.5)$, $(x_5, y_5) = (85, 3.3)$.

The prediction error on the 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.

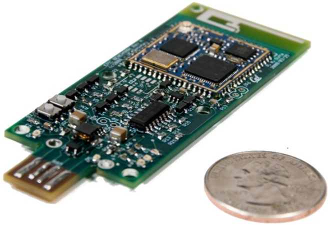
Four test periods:

- 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)`

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

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

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

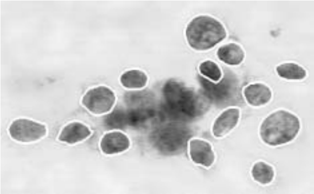
But solution to least-squares problem is

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 $\left\| \begin{bmatrix} \mathbf{b} \end{bmatrix} - \begin{bmatrix} \mathbf{a}_1 \\ \hline \vdots \\ \hline \mathbf{a}_m \end{bmatrix} \begin{bmatrix} \mathbf{x} \end{bmatrix} \right\|^2$

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*