Vector addition: The zero vector

The D-vector whose entries are all zero is the *zero vector*, written $\mathbf{0}_D$ or just $\mathbf{0}$

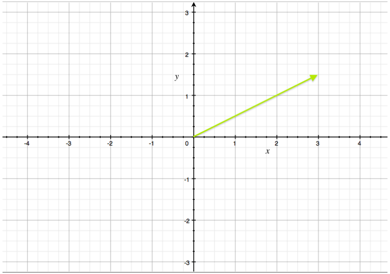
$$\mathbf{v} + \mathbf{0} = \mathbf{v}$$

Vector addition: Vector addition is associative and commutative

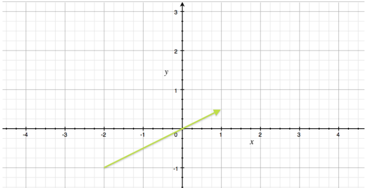
- *Associativity*
$$(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$$
- *Commutativity*
$$\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$$

Vector addition: Vectors as arrows

Like complex numbers in the plane, *n*-vectors over $\mathbb{R}$ can be visualized as *arrows* in $\mathbb{R}^n$.



The 2-vector $[3, 1.5]$ can be represented by an arrow with its tail at the origin and its head at $(3, 1.5)$.



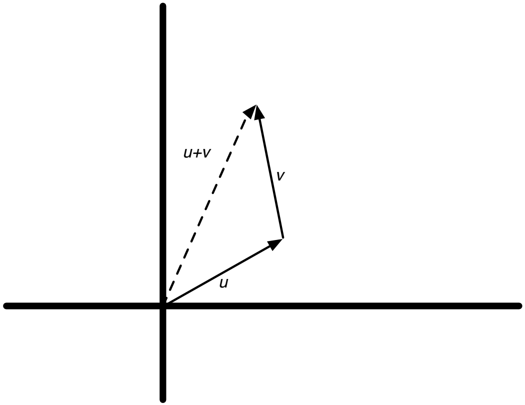
or, equivalently, by an arrow whose tail is at $(-2, -1)$ and whose head is at $(1, 0.5)$.

Vector addition: Vectors as arrows

Like complex numbers, addition of vectors over $\mathbb{R}$ can be visualized using arrows.

To add **u** and **v**:

- place tail of **v**'s arrow on head of **u**'s arrow;
- draw a new arrow from tail of **u** to head of **v**.



Scalar-vector multiplication

With complex numbers, *scaling* was multiplication by a real number $f(z) = r\,z$

For vectors,

- we refer to field elements as *scalars*;
- we use them to scale vectors:

$$\alpha\,\mathbf{v}$$

  Greek letters (e.g. α, β, γ) denote scalars.

Scalar-vector multiplication

Definition: Multiplying a vector $\mathbf{v}$ by a scalar α is defined as multiplying each entry of $\mathbf{v}$ by α:

$$\alpha\,[v_1, v_2, \ldots, v_n] = [\alpha\, v_1, \alpha\, v_2, \ldots, \alpha\, v_n]$$

Example: $2\,[5, 4, 10] = [2 \cdot 5, 2 \cdot 4, 2 \cdot 10] = [10, 8, 20]$

Scalar-vector multiplication

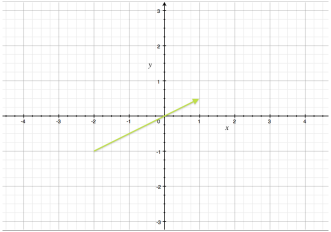
Quiz: Suppose we represent n-vectors by n-element lists. Write a procedure `scalar_vector_mult(alpha, v)` that multiplies the vector `v` by the scalar `alpha`.

Answer:

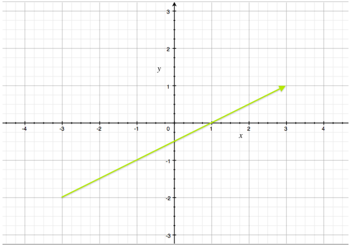
```
def scalar_vector_mult(alpha, v): return [alpha*x for x in v]
```

Scalar-vector multiplication: Scaling arrows

An arrow representing the vector $[3, 1.5]$ is this:



and an arrow representing two times this vector is this:



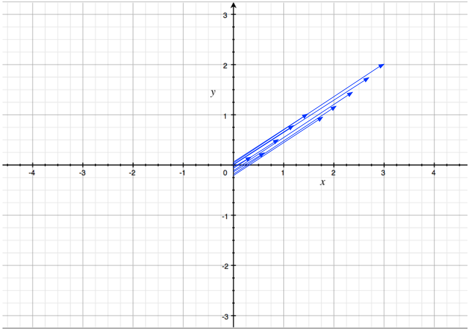
Scalar-vector multiplication: Associativity of scalar-vector multiplication

Associativity: $\alpha(\beta \mathbf{v}) = (\alpha\beta)\mathbf{v}$

Scalar-vector multiplication: Line segments through the origin

Consider scalar multiples of $\mathbf{v} = [3, 2]$:
$\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$

For each value of α in this set,
$\alpha\,\mathbf{v}$ is shorter than $\mathbf{v}$ but in same direction.



Scalar-vector multiplication: Line segments through the origin

Conclusion: The set of points

$$\{\alpha\,\mathbf{v} \;:\; \alpha \in \mathbb{R}, 0 \leq \alpha \leq 1\}$$

forms the line segment between the origin and $\mathbf{v}$

Scalar-vector multiplication: Lines through the origin

What if we let α range over all real numbers?

- Scalars bigger than 1 give rise to somewhat larger copies
- Negative scalars give rise to vectors pointing in the opposite direction

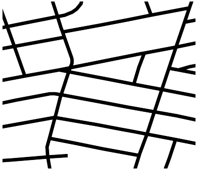
The set of points

$$\{\alpha\,\mathbf{v} \,:\, \alpha \in \mathbb{R}\}$$

forms the line through the origin and **v**

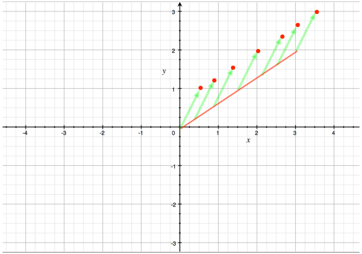
Combining vector addition and scalar multiplication

We want to describe the set of points forming an arbitrary line segment (not necessarily through the origin).
Idea: Use translation.



Start with line segment from $[0, 0]$ to $[3, 2]$:

$$\{\alpha\,[3, 2]\ :\ 0 \leq \alpha \leq 1\}$$



Translate it by adding $[0.5, 1]$ to every point:

$$\{[0.5, 1] + \alpha\,[3, 2]\ :\ 0 \leq \alpha \leq 1\}$$

Get line segment from $[0, 0] + [0.5, 1]$ to $[3, 2] + [0.5, 1]$

Combining vector addition and scalar multiplication: Distributive laws for scalar-vector multiplication and vector addition

Scalar-vector multiplication distributes over vector addition:

$$\alpha(\mathbf{u} + \mathbf{v}) = \alpha\mathbf{u} + \alpha\mathbf{v}$$

Example:

- On the one hand,

$$2\,([1,2,3] + [3,4,4]) = 2\,[4,6,7] = [8,12,14]$$

- On the other hand,

$$2\,([1,2,3] + [3,4,4]) = 2\,[1,2,3] + 2\,[3,4,4] = [2,4,6] + [6,8,8] = [8,12,14]$$

Combining vector addition and scalar multiplication: First look at convex combinations

Set of points making up the the $[0.5, 1]$-to-$[3.5, 3]$ segment:

$$\{\alpha\,[3,2] + [0.5,1] \;:\; \alpha \in \mathbb{R}, 0 \le \alpha \le 1\}$$

Not symmetric with respect to endpoints ☹

Use distributivity:

$$\begin{aligned}
\alpha\,[3,2] + [0.5,1] &= \alpha\,([3.5,3] - [0.5,1]) + [0.5,1] \\
&= \alpha\,[3.5,3] - \alpha\,[0.5,1] + [0.5,1] \\
&= \alpha\,[3.5,3] + (1-\alpha)\,[0.5,1] \\
&= \alpha\,[3.5,3] + \beta\,[0.5,1]
\end{aligned}$$

where $\beta = 1 - \alpha$

New formulation:

$$\{\alpha\,[3.5,3] + \beta\,[0.5,1] \;:\; \alpha, \beta \in \mathbb{R}, \alpha, \beta \ge 0, \alpha + \beta = 1\}$$

Symmetric with respect to endpoints ☺

Combining vector addition and scalar multiplication: First look at convex combinations

New formulation:

$$\{\alpha\,[3.5, 3] + \beta\,[0.5, 1] \;:\; \alpha, \beta \in \mathbb{R}, \alpha, \beta \geq 0, \alpha + \beta = 1\}$$

Symmetric with respect to endpoints ☺

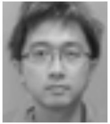
An expression of the form

$$\alpha\,\mathbf{u} + \beta\,\mathbf{v}$$

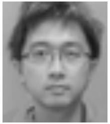
where $0 \leq \alpha \leq 1, 0 \leq \beta \leq 1$, and $\alpha + \beta = 1$ is called a *convex combination* of **u** and **v**

The **u**-to-**v** line segment consists of the set of convex combinations of **u** and **v**.

Combining vector addition and scalar multiplication: First look at convex combinations

$\mathbf{u} =$  and $\mathbf{v} =$ 

Use scalars $\alpha = \frac{1}{2}$ and $\beta = \frac{1}{2}$:

$\frac{1}{2}$  $+$ $\frac{1}{2}$  $=$ 

"Line segment" between two faces:


$1\mathbf{u} + 0\mathbf{v}$


$\frac{7}{8}\mathbf{u} + \frac{1}{8}\mathbf{v}$

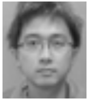

$\frac{6}{8}\mathbf{u} + \frac{2}{8}\mathbf{v}$

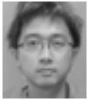

$\frac{5}{8}\mathbf{u} + \frac{3}{8}\mathbf{v}$


$\frac{4}{8}\mathbf{u} + \frac{4}{8}\mathbf{v}$


$\frac{3}{8}\mathbf{u} + \frac{5}{8}\mathbf{v}$


$\frac{2}{8}\mathbf{u} + \frac{6}{8}\mathbf{v}$


$\frac{1}{8}\mathbf{u} + \frac{7}{8}\mathbf{v}$


$0\mathbf{u} + 1\mathbf{v}$

Combining vector addition and scalar multiplication: First look at convex combinations

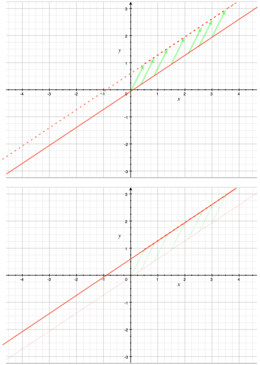
Line segments not necessarily through the origin

How to write the (infinite) line through $[0.5, 1]$ and $[3.5, 3]$?

Start with the line through the origin and $[3, 2]$, and translate it by adding $[0.5, 1]$ to each point.

The untranslated line is $\{\alpha\,[3, 2]\ :\ \alpha \in \mathbb{R}\}$.

so the translated line is $\{[0.5, 1] + \alpha\,[3, 2]\ :\ \alpha \in \mathbb{R}\}$



Combining vector addition and scalar multiplication: First look at affine combinations

Infinite line through $[0.5, 1]$ and $[3.5, 3]$?
Our formulation so far ☹

$$\{[0.5, 1] + \alpha\,[3, 2] \;:\; \alpha \in \mathbb{R}\}$$

Nicer formulation ☺:

$$\{\alpha\,[3.5, 3] + \beta\,[0.5, 1] \;:\; \alpha \in \mathbb{R}, \beta \in \mathbb{R}, \alpha + \beta = 1\}$$

An expression of the form $\alpha\,\mathbf{u} + \beta\,\mathbf{v}$ where $\alpha + \beta = 1$ is called an *affine* combination of $\mathbf{u}$ and $\mathbf{v}$.

> The line through **u** and **v** consists of the set of affine combinations of **u** and **v**.

Vectors over $GF(2)$

Addition of vectors over $GF(2)$:

$$\begin{array}{rccccc} & 1 & 1 & 1 & 1 & 1 \\ + & 1 & 0 & 1 & 0 & 1 \\ \hline & 0 & 1 & 0 & 1 & 0 \end{array}$$

For brevity, in doing $GF(2)$, we often write 1101 instead of [1,1,0,1].

Example: Over $GF(2)$, what is $1101 + 0111$?
Answer: 1010

Vectors over $GF(2)$: Perfect secrecy

Represent encryption of n bits by addition of n-vectors over $GF(2)$.

Example:

Alice and Bob agree on the following 10-vector as a key:

$$\mathbf{k} = [0, 1, 1, 0, 1, 0, 0, 0, 0, 1]$$

Alice wants to send this message to Bob:

$$\mathbf{p} = [0, 0, 0, 1, 1, 1, 0, 1, 0, 1]$$

She encrypts it by adding $\mathbf{p}$ to $\mathbf{k}$:

$$\mathbf{c} = \mathbf{k} + \mathbf{p} = [0, 1, 1, 0, 1, 0, 0, 0, 0, 1] + [0, 0, 0, 1, 1, 1, 0, 1, 0, 1] = [0, 1, 1, 1, 0, 1, 0, 1, 0, 0]$$

When Bob receives $\mathbf{c}$, he decrypts it by adding $\mathbf{k}$:

$$\mathbf{c} + \mathbf{k} = [0, 1, 1, 1, 0, 1, 0, 1, 0, 0] + [0, 1, 1, 0, 1, 0, 0, 0, 0, 1] = [0, 0, 0, 1, 1, 1, 0, 1, 0, 1]$$

which is the original message.

Vectors over $GF(2)$: Perfect secrecy

If the key is chosen according to the uniform distribution, encryption by addition of vectors over $GF(2)$ achieves *perfect secrecy*.
For each plaintext $\mathbf{p}$, the function that maps the key to the cyphertext

$$\mathbf{k} \mapsto \mathbf{k} + \mathbf{p}$$

is invertible

Since the key $\mathbf{k}$ has the uniform distribution,
the cyphertext $\mathbf{c}$ also has the uniform distribution.

Vectors over $GF(2)$: All-or-nothing secret-sharing using $GF(2)$

- I have a secret: the midterm exam.
- I've represented it as an n-vector $\mathbf{v}$ over $GF(2)$.
- I want to provide it to my TAs Alice and Bob (A and B) so they can administer the midterm while I take vacation.
- One TA might be bribed by a student into giving out the exam ahead of time, so I don't want to simply provide each TA with the exam.
- **Idea:** Provide pieces to the TAs:
  - the two TAs can jointly reconstruct the secret, but
  - neither of the TAs all alone gains any information whatsoever.
- **Here's how:**
  - I choose a random n-vector $\mathbf{v}_A$ over $GF(2)$ randomly according to the uniform distribution.
  - I then compute

$$\mathbf{v}_B := \mathbf{v} - \mathbf{v}_A$$

  - I provide Alice with $\mathbf{v}_A$ and Bob with $\mathbf{v}_B$, and I leave for vacation.

Vectors over $GF(2)$: All-or-nothing secret-sharing using $GF(2)$

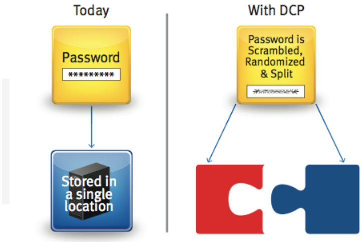
- What can Alice learn without Bob?
- All she receives is a random n-vector.
- What about Bob?
- He receives the output of $f(\mathbf{x}) = \mathbf{v} - \mathbf{x}$ where the input is random and uniform.
- Since $f(\mathbf{x})$ is invertible, the output is also random and uniform.

Vectors over $GF(2)$: All-or-nothing secret-sharing using $GF(2)$

Too simple to be useful, right?
RSA just introduced a product based on this idea:

RSA® DISTRIBUTED CREDENTIAL PROTECTION

Scramble, randomize and split credentials



- Split each password into two parts.
- Store the two parts on two separate servers.

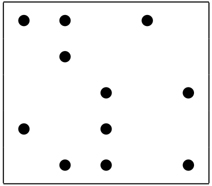
Vectors over $GF(2)$: *Lights Out*

- *input:* Configuration of lights
- *output:* Which buttons to press in order to turn off all lights?

Computational Problem: $\boxed{\text{Solve an instance of } \textit{Lights Out}}$

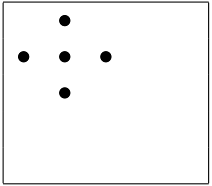
Represent state using `range(5)`$\times$`range(5)`-vector over $GF(2)$.

Example state vector:



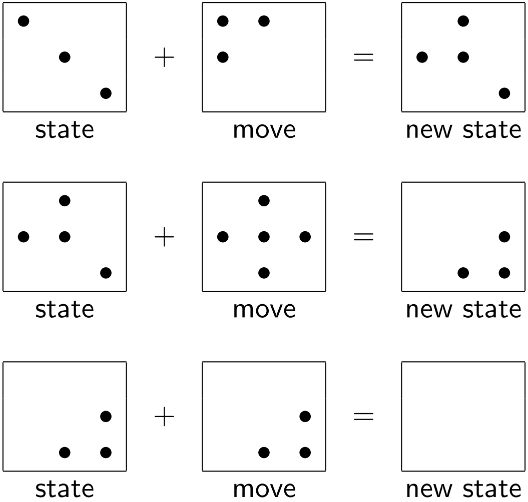
Represent each button as a vector (with ones in positions that the button toggles)

Example button vector:

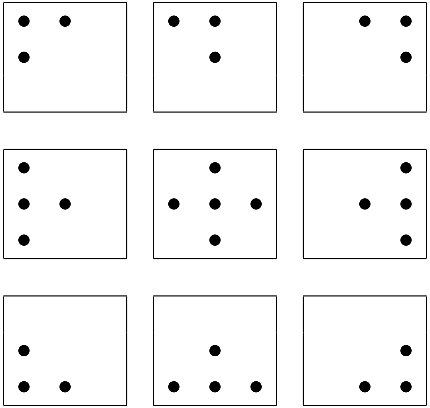


Vectors over $GF(2)$: *Lights Out*

Look at 3×3 case.



Vectors over $GF(2)$: 3×3 *Lights Out* button vectors



Computational Problem: $\boxed{\text{Which sequence of button vectors plus } \mathbf{s} \text{ sums to } \mathbf{0}?}$

$\Longrightarrow$ $\boxed{\text{Which sequence of button vectors sum to } \mathbf{s}?}$

Vectors over $GF(2)$: *Lights Out*

Computational Problem: $\boxed{\text{Which sequence of button vectors sums to } \mathbf{s}\text{?}}$

Observations:

- By commutative property of vector addition, order doesn't matter.
- A button vector occuring twice cancels out.

Replace Computational Problem with: $\boxed{\text{Which set of button vectors sums to } \mathbf{s}\text{?}}$

Vectors over $GF(2)$: *Lights Out*

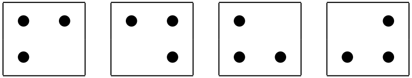
Replace our original Computational Problem with a more general one:

$\boxed{\text{Solve an instance of } \textit{Lights Out}}$ $\Rightarrow$ $\boxed{\text{Which set of button vectors sum to } \mathbf{s}\text{?}}$

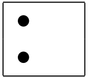
$\Rightarrow$ $\boxed{\begin{array}{l}\text{Find subset of } GF(2) \text{ vectors} \\ \mathbf{v}_1, \ldots, \mathbf{v}_n \text{ whose sum equals } \mathbf{s}\end{array}}$

Vectors over $GF(2)$: *Lights Out*

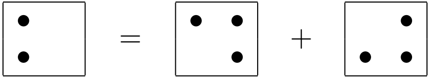
Button vectors for 2×2 version:



where the black dots represent ones.

Quiz: Find the subset of the button vectors whose sum is 

Answer:



Dot-product

Dot-product of two D-vectors is sum of product of corresponding entries:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{k \in D} \mathbf{u}[k]\ \mathbf{v}[k]$$

Example: For traditional vectors $\mathbf{u} = [u_1, \ldots, u_n]$ and $\mathbf{v} = [v_1, \ldots, v_n]$,

$$\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

Output is a scalar, not a vector
Dot-product sometimes called *scalar product*.

Dot-product

Example: Dot-product of $[1, 1, 1, 1, 1]$ and $[10, 20, 0, 40, -100]$:

$$
\begin{array}{ccccccccccc}
 & 1 & & 1 & & 1 & & 1 & & 1 & & \\
\bullet & 10 & & 20 & & 0 & & 40 & & \text{-100} & & \\
\hline
 & 10 & + & 20 & + & 0 & + & 40 & + & (\text{-100}) & = & \text{-30}
\end{array}
$$

Dot-product: Total cost or benefit

Suppose D consists of four main ingredients of beer:

$$D = \{\text{malt}, \text{hops}, \text{yeast}, \text{water}\}$$

A *cost* vector maps each food to a price per unit amount:

$$cost = \{\text{hops} : \$2.50/ounce, \text{malt} : \$1.50/pound, \text{water} : \$0.06/gallon, \text{yeast} : \$.45/g\}$$

A *quantity* vector maps each food to an amount (e.g. measured in pounds).
$quantity = \{$hops:6 oz, malt:14 pounds, water:7 gallons, yeast:11 grams$\}$

The total cost is the dot-product of *cost* with *quantity*:

$$cost \cdot quantity = \$2.50 \cdot 6 + \$1.50 \cdot 14 + \$0.006 \cdot 7 + \$0.45 \cdot 11 = \$40.992$$

A *value* vector maps each food to its caloric content per pound:

$$value = \{\text{hops} : 0, \text{malt} : 960, \text{water} : 0, \text{yeast} : 3.25\}$$

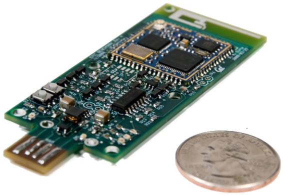
The total calories represented by a pint is the dot-product of *value* with *quantity*:

$$value \cdot quantity = 0 \cdot 6 + 960 \cdot 14 + 7 \cdot 0 + 3.25 \cdot 11 = 13475.75$$

Dot-product: Linear equations

Example: A sensor node consist of hardware components, e.g.

- CPU
- radio
- temperature sensor
- memory



Battery-driven and remotely located so we care about energy usage.

Suppose we know the power consumption for each hardware component. Represent it as a D-vector with $D = \{radio, sensor, memory, CPU\}$

$$\mathbf{rate} = \text{Vec}(D, \{memory : 0.06\text{W}, radio : 0.06\text{W}, sensor : 0.004\text{W}, CPU : 0.0025\text{W}\})$$

Have a test period during which we know how long each component was working. Represent as another D vector:

$$\mathbf{duration} = \text{Vec}(D, \{memory : 1.0\text{s}, radio : 0.2\text{s}, sensor : 0.5\text{s}, CPU : 1.0\text{s}\})$$

Total energy consumed (in Joules): $\mathbf{duration} \cdot \mathbf{rate}$