Matrix-vector and vector-matrix multiplication

Two ways to multiply a matrix by a vector:

- matrix-vector multiplication
- vector-matrix multiplication

For each of these, two *equivalent definitions*:

- in terms of linear combinations
- in terms of dot-products

Matrix-vector multiplication in terms of linear combinations

Linear-Combinations Definition of matrix-vector multiplication: Let M be an $R \times C$ matrix.

- If $\mathbf{v}$ is a C-vector then

$$M * \mathbf{v} = \sum_{c \in C} \mathbf{v}[c] \text{ (column } c \text{ of } M)$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{bmatrix} * [7, 0, 4] = 7\,[1, 10] + 0\,[2, 20] + 4\,[3, 30]$$

- If $\mathbf{v}$ is *not* a C-vector then

$$M * \mathbf{v} = \text{ERROR!}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{bmatrix} * [7, 0] = \text{ERROR!}$$

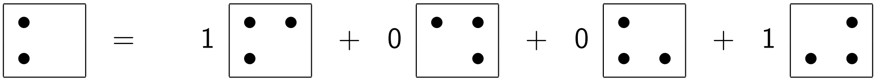
Matrix-vector multiplication in terms of linear combinations

$$\begin{array}{c|ccc} & @ & \# & ? \\ \hline a & 2 & 1 & 3 \\ b & 20 & 10 & 30 \end{array} \quad * \quad \begin{array}{ccc} @ & \# & ? \\ \hline 0.5 & 5 & \text{-1} \end{array} \quad = \quad \begin{array}{c|c} a & 3 \\ b & 30 \end{array}$$

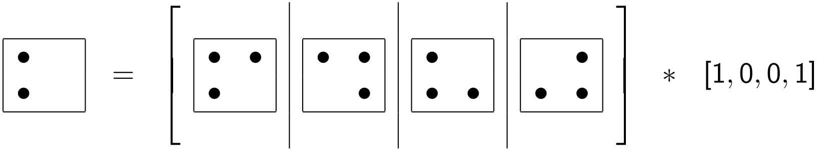
Matrix-vector multiplication in terms of linear combinations: *Lights Out*

A solution to a *Lights Out* configuration is a linear combination of “button vectors.”

For example, the linear combination

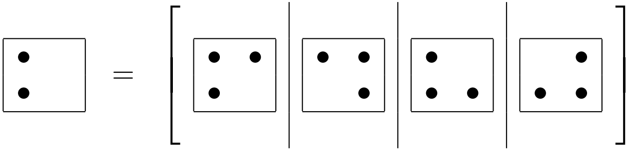


can be written as



Solving a matrix-vector equation: *Lights Out*

$\boxed{\text{Solving an instance of \textit{Lights Out}}} \quad \Rightarrow \quad \boxed{\text{Solving a matrix-vector equation}}$



$* \quad [\alpha_1, \alpha_2, \alpha_3, \alpha_4]$

Solving a matrix-vector equation

Fundamental Computational Problem: *Solving a matrix-vector equation*

- *input:* an $R \times C$ matrix A and an R-vector $\mathbf{b}$
- *output:* the C-vector $\mathbf{x}$ such that $A * \mathbf{x} = \mathbf{b}$

Solving a matrix-vector equation: 2×2 special case

Simple formula to solve

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} * [x, y] = [p, q]$$

if $ad \neq bc$:

$$x = \frac{dp - cq}{ad - bc} \text{ and } y = \frac{aq - bp}{ad - bc}$$

For example, to solve

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * [x, y] = [-1, 1]$$

we set

$$x = \frac{4 \cdot -1 - 2 \cdot 1}{1 \cdot 4 - 2 \cdot 3} = \frac{-6}{-2} = 3$$

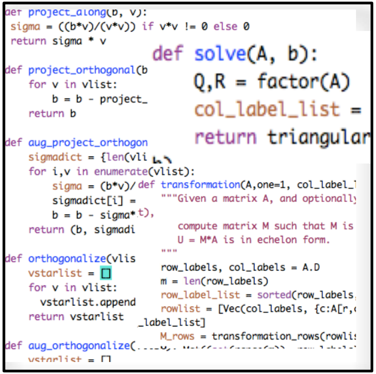
and

$$y = \frac{1 \cdot 1 - 3 \cdot -1}{1 \cdot 4 - 2 \cdot 3} = \frac{4}{-2} = -2$$

Later we study algorithms for more general cases.

The `solver` module

We provide a module `solver` that defines a procedure `solve(A, b)` that tries to find a solution to the matrix-vector equation $A\mathbf{x} = \mathbf{b}$
Currently `solve(A, b)` is a black box



but we will learn how to code it in the coming weeks.

Let's use it to solve this *Lights Out* instance...

Vector-matrix multiplication in terms of linear combinations

Vector-matrix multiplication is different from matrix-vector multiplication:

Let M be an $R \times C$ matrix.
Linear-Combinations Definition of matrix-vector multiplication: If $\mathbf{v}$ is a C-vector then

$$M * \mathbf{v} = \sum_{c \in C} \mathbf{v}[c] \text{ (column } c \text{ of } M)$$

Linear-Combinations Definition of vector-matrix multiplication: If $\mathbf{w}$ is an R-vector then

$$\mathbf{w} * M = \sum_{r \in R} \mathbf{w}[r] \text{ (row } r \text{ of } M)$$

$$[3,4] \quad * \quad \begin{bmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{bmatrix} \quad = \quad 3\,[1,2,3] \quad + \quad 4\,[10,20,30]$$

Vector-matrix multiplication in terms of linear combinations: JunkCo





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

$$\text{total resources used} = [\alpha_{\text{gnome}}, \alpha_{\text{hoop}}, \alpha_{\text{slinky}}, \alpha_{\text{putty}}, \alpha_{\text{shooter}}] * \text{M}$$

Suppose we know total resources used and we know M.

To find the values of $\alpha_{\text{gnome}}, \alpha_{\text{hoop}}, \alpha_{\text{slinky}}, \alpha_{\text{putty}}, \alpha_{\text{shooter}}$,

solve a *vector-matrix* equation $\mathbf{b} = \mathbf{x} * M$
where $\mathbf{b}$ is vector of total resources used.

Solving a matrix-vector equation

Fundamental Computational Problem: *Solving a matrix-vector equation*

- *input:* an $R \times C$ matrix A and an R-vector $\mathbf{b}$
- *output:* the C-vector $\mathbf{x}$ such that $A * \mathbf{x} = \mathbf{b}$

If we had an algorithm for solving a *matrix-vector* equation,
could also use it to solve a *vector-matrix* equation,
using transpose.

The `solver` module, and floating-point arithmetic

For arithmetic over $\mathbb{R}$, Python uses floats, so round-off errors occur:

```
>>> 10.0**16 + 1 == 10.0**16
True
```

Consequently algorithms such as that used in `solve(A, b)` do not find exactly correct solutions.

To see if solution $\mathbf{u}$ obtained is a reasonable solution to $A * \mathbf{x} = \mathbf{b}$, see if the vector $\mathbf{b} - A * \mathbf{u}$ has entries that are close to zero:

```
>>> A = listlist2mat([[1,3],[5,7]])
>>> u = solve(A, b)
>>> b - A*u
Vec({0, 1},{0: -4.440892098500626e-16, 1: -8.881784197001252e-16})
```

The vector $\mathbf{b} - A * \mathbf{u}$ is called the *residual*. Easy way to test if entries of the residual are close to zero: compute the dot-product of the residual with itself:

```
>>> res = b - A*u
>>> res * res
9.860761315262648e-31
```

Checking the output from `solve(A, b)`

For some matrix-vector equations $A * \mathbf{x} = \mathbf{b}$, there is no solution.
In this case, the vector returned by `solve(A, b)` gives rise to a largeish residual:

```
>>> A = listlist2mat([[1,2],[4,5],[-6,1]])
>>> b = list2vec([1,1,1])
>>> u = solve(A, b)
>>> res = b - A*u
>>> res * res
0.24287856071964012
```

Later in the course we will see that the residual is, in a sense, as small as possible.

Some matrix-vector equations are *ill-conditioned*, which can prevent an algorithm using floats from getting even approximate solutions, even when solutions exists:

```
>>> A = listlist2mat([[1e20,1],[1,0]])
>>> b = list2vec([1,1])
>>> u = solve(A, b)
>>> b - A*u
Vec({0, 1},{0: 0.0, 1: 1.0})
```

We will not study conditioning in this course.