Simulations (cont.), Nearest Neighbor, Map-Reduce, Predicting Gender based on weight and height, Decolorizing Photo, Old School Graphics, and AI for Video Game

Dr. Jason Schanker

Review: Pseudorandom number generator

- JavaScript and most other programming languages have a way of generating pseudo-random numbers. In JavaScript, `Math.random()` is used to generate a pseudo-random number from 0 to 1 (including 0 and excluding 1). Despite the function name, `Math.random()` does not actually produce random numbers. It is completely determined by a seed and the number of times it is called. Think of a book of random numbers in which the seed is the page in which the number produced when calling it for the nth time is the nth number appearing on the page: It never changes!
- More formally, we can think of `Math.random()` as $f^n(seed)$ where n is the number of times it's been used since the program started and `seed` is some starting value, which is often gotten from the current state of the computer (e.g., the time).

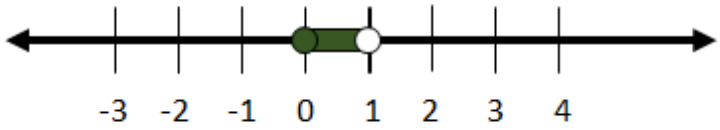
Generating pseudorandom numbers for other ranges

- To generate a random number from a to b instead (interval [a, b)):
  1. Multiply by `Math.random()` by `b-a` (this "stretches" the length of the interval from 1 (0 to 1) to a length of b-a) to get a number in [0, b-a).
  2. Add `a` (shifts the interval by a (+ => Right, - => Left)) to get a number in [a, b).
- Example (a = -1, b = 2):
  1. Multiply `Math.random()` by b - a = 2-(-1) = 3 to get `3*Math.random()`. This stretches the interval to have a length of 3, giving you a number in [0, 3)
  2. Then add a=-1 to get `3*Math.random()-1`. This shifts the interval 1 to the left, giving you a number in [-1, 2)
- See picture on next slide.

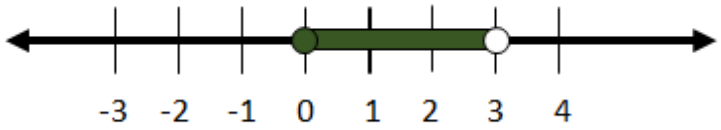
Generating pseudorandom numbers for other ranges

From [0, 1) to other interval

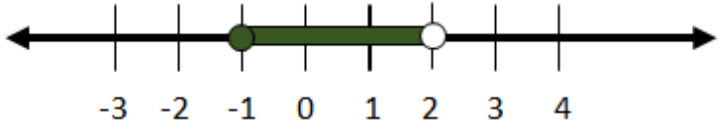
- What if you want a random number in [-1, 2) instead?




```
Math.random();
```




```
3*Math.random();
```




```
3*Math.random()-1;
```

- Interval length: 3 (2 – (-1): Stretch x 3) Smallest number: -1 (Shift left 1)

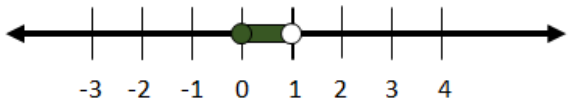
Generating pseudorandom integers in range

- If you instead want to simulate an experiment with only integer outcomes from a to b-1 instead (e.g., to simulate the possibilities 1, 2, 3, 4, 5, 6 resulting from a roll of a die):
  1. Multiply by `Math.random()` by `b-a` (this "stretches" the length of the interval from 1 (0 to 1) to a length of b-a, which is the number of possibilities) to get a number in [0, b-a).
  2. Add `a` (shifts the interval by a (+ => Right, - => Left)) to get a number in [a, b).
  3. Enclose your answer to (2) in `Math.floor(…)`; this will round **DOWN** to the greatest integer greater than or equal to the input (e.g., Math.floor(3.75) = 3; Math.floor(-3.25) = -4; Math.floor(3) = 3). This gives you the possible outcomes a, a+1, …, b-1
- Example (Possibilities of -1, 0, 1):
  1. Multiply `Math.random()` by b - a = 2-(-1) = 3 to get `3*Math.random()`. This stretches the interval to have a length of 3, which is the number of integers from -1 to 2, including -1 and excluding 2, giving you a number in [0, 3).
  2. Then add a=-1 to get `3*Math.random()-1`. This shifts the interval 1 to the left, giving you a number in [-1, 2)
  3. Surround `3*Math.random()-1` by `Math.floor` to get `Math.floor(3*Math.random()-1)`; this gives you the possible numbers -1, 0, or 1.
- See picture on next slide.

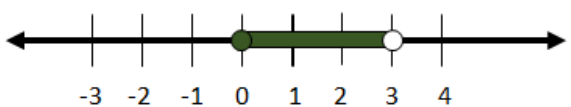
Generating pseudorandom numbers for other ranges

From [0, 1) to other interval

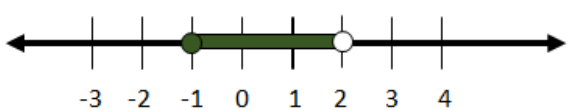
- What if you want a random number in [-1, 2) instead?




```
Math.random();
```

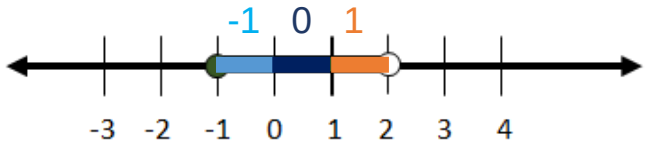



```
3*Math.random();
```




```
3*Math.random()-1;
```

- Interval length: 3 (2 – (-1): Stretch x 3) Smallest number: -1 (Shift left 1)

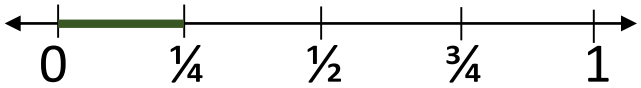



```
Math.floor(3*Math.random()-1)
```

[-1, 0) => -1 ; [0, 1) => 0; [1, 2) => 1

Using `Math.random()` to simulate experiments

- The probability that a randomly chosen number from 0 to 1 will be between 0 and p is equal to p!
  - For example, for p = 1/4, the probability that a randomly chosen number will be less than or equal to ¼ is ¼, as depicted below (green shaded region is ¼ of the length of the interval [0, 1]):



  - Since `Math.random()` produces a number between 0 and 1 (0 is technically possible but is basically impossible; 1 will never occur), we can simulate an experiment with probability ¼ with the inequality `Math.random() < 1/4`; `true` indicates the outcome with ¼ probability happened, while `false` indicates that the outcome did **NOT** happen, which will occur roughly ¾ of the time.
  - **NOTE**: This information should help you with the `rainsOneWeekFromNow` and `winsOneHundred` functions.

Markov Chains (Linear Algebra Application)

- Using the transition matrix *P* to the right, you can calculate the probability of winning using this strategy by determining

$$\lim_{n\to\infty} P^n$$

Ending

Starting	$0	$100	$200	$300	$400	$500
$0	1	0	0	0	0	0
$100	$\frac{10}{19}$	0	$\frac{9}{19}$	0	0	0
$200	0	$\frac{10}{19}$	0	$\frac{9}{19}$	0	0
$300	0	0	$\frac{10}{19}$	0	$\frac{9}{19}$	0
$400	0	0	0	$\frac{10}{19}$	0	$\frac{9}{19}$
$500	0	0	0	0	0	1

(the probability will be in row 5, column 6 (start with $400, end with $500). You'll learn how to do this in Linear Algebra, if you haven't already. (Think Eigenvalues and Eigenvectors and $S\Lambda S^{-1}$ decomposition.

- Probability of going from $400 - $500 using this strategy: about --ANSWER TO-HOMEWORK--**.58%! (Whole number part)**
- Probability of going from $300 - $500 using this strategy: about 53.6%

WARNING: It's called Gambler's Ruin for a Reason!

- Although you may be tempted to go to Vegas to try this strategy out, remember in the long term you will lose money!
- Remember that when you win, you only win $100, but when you lose, you lose $400!
- To see your average earnings per game (so-called expected value), multiply $100 by the probability that you win with this strategy and subtract 400*probability you lose, you'll see that you get a negative number, a loss! ☹

Starting \ Ending	$0	$100	$200	$300	$400	$500
$0	1	0	0	0	0	0
$100	$\frac{10}{19}$	0	$\frac{9}{19}$	0	0	0
$200	0	$\frac{10}{19}$	0	$\frac{9}{19}$	0	0
$300	0	0	$\frac{10}{19}$	0	$\frac{9}{19}$	0
$400	0	0	0	$\frac{10}{19}$	0	$\frac{9}{19}$
$500	0	0	0	0	0	1

Rains one week from now

➢Using the transition matrix *P* to the right, you can calculate the probability of rain one week from now given rain today (and the given probabilities) by determining P^7 (probability will be in row 1, column 1).

- Probability of rain one week from now given rain today is about answer from homework, which is actually the same as rain one week from now given no rain today.

TODAY \ TOMORROW	RAIN	NO RAIN
RAIN	0.3	0.7
NO RAIN	0.4	0.6

Machine Learning Classification Algorithm

- A common problem in machine learning is correctly classifying data by a label (e.g., an e-mail by spam or ham label – ham means NOT spam; web page text by its language – e.g., English, Hindi, Japanese, Spanish, etc. label)
- A **supervised** machine learning classification algorithm attempts to **predict** the label of data it hasn't seen before (e.g., a new e-mail or web page) by using a large amount of **labelled training data** (e.g., a large number of e-mails with spam or ham classifications, a large number of web pages with English, Hindi, Japanese, Spanish, etc. classifications).

Nearest Neighbor Classification Algorithm

- A Nearest neighbor algorithm is an example of a simple supervised machine learning classification algorithm that can be used in cases where there's a sensible concept of distances between possible data with smaller distances indicating a higher probability of similarity.
- A 1-nearest neighbor algorithm predicts the label of a data point by using the label of the closest data point from its training data.
- A 5-nearest neighbor algorithm locates the 5 closest points and outputs the label of the most frequently occurring of the 5 (e.g., if 3 of the closest points had label A, 1 of the closest points had label B, and the final 1 had label C, label A would be the chosen one to classify the new data point).
- A k-nearest algorithm could be used for any k to take the most frequently occurring label of the k nearest neighbors where k is a most equal to the number of data points but typically is a small number such as 5.

Predicting Gender by Weight/Height

- As an example, we could classify a large number of data points representing people's weights and heights by gender. The data points would be (weight, height) pairs and the labels would be their genders.
- The distance between weight-height data points (w_1, h_1) and (w_2, h_2) can be defined by the ordinary (Euclidean) distance between 2 points given by the Pythagorean Theorem $d = \sqrt{(w_2 - w_1)^2 + (h_2 - h_1)^2}$.
- Then the 1-nearest neighbor algorithm would assign a new weight/height to be the label (either male or female) of the (weight, height) point from the training data that's closest to it.

Decolorizing Images

- As another example, recall that pixel colors can be represented by triples (red, green, blue) where red, green, and blue are all integers from 0 to 255, the higher the number the brighter the color.
- With this color scheme, there are $256^3 = (2^8)^3 = 2^{24}$ =16,777,216 possible colors!
- If we want to convert an image to a more limited color palette using only 16 possible colors or to grayscale using the 256 possible colors (red, green, blue all same numbers), we can use the 1-nearest neighbor algorithm!
- Here, we use the 3D-Euclidean Distance between two rgb colors (r_1, g_1, b_1) and (r_2, g_2, b_2) by $\sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2}$ and find the closest allowed color.
- Here, the training data is the r-g-b triples of allowed colors and we're trying to predict the colors of each pixel data from the original photo to best represent the original photo with this smaller color palette.

Implementations with `map`/`reduce`

- We can implement a 1-Nearest-Neighbor algorithm by first using `map` on the Array of training data to get an Array of distances, where the item at `index` is the distance that the new point we're trying to predict is from the training data point located at `index`.
- Next, on the resulting distance Array, we get the `index` of the minimum distance using the built-in JavaScript Array method `reduce`. This is replicated by the `repeatForEveryItem(…).startingWithAccOf(…)` from the Text2Code library.
- Finally, we output the label of this index to get the predicted label for the new data point.

Parallelizing with `map`/`reduce`

- Many problems can be solved by using a combination of `map` and `reduce`. What's great about the `map` part is that it can be computed in **parallel**, meaning multiple processors can work on different parts at the **same** time! This is because the map only uses individual items from the Array so we can split the Arrays into parts.
- For example, if there are four processors available and we want to compute `[1,2,3,4,5,6,7,8].map(x => 2*x)`, i.e., double every element in the Array, processor 1 can compute `[1,2].map(x => 2*x)`, processor 2 can compute `[3,4].map(x => 2*x)`, processor 3 can compute `[5,6].map(x => 2*x)`, and processor 4 can compute `[7,8].map(x => 2*x)` to get the results [2, 4], [6, 8], [10, 12], and [14, 16], which can then be joined together to get the resulting Array [2, 4, 6, 8, 10, 12, 14, 16].
- **Although this won't be what happens even when a computer has 4 processors available (most modern computers have at least 2), we can write code so this does happen and it's important to be able to do so!**