Simulations: Approximating π, Predicting the Weather and Winning in Roulette

Dr. Jason Schanker

Empirical Probability

- The probability that an event occurs can be approximated by determining the relative frequency for which the event occurs over a large number of trials; i.e.,

$$Probability\ of\ event \approx \frac{number\ of\ times\ event\ occured}{number\ of\ trials}$$

- **Example:** If you flip a coin 1,000,000 times (trials) and it lands on heads 500,329 times, you'd approximate the probability to be: $\frac{500{,}329}{1{,}000{,}000} = .500329 = 50.0329\%.$

Template for Running Experiment

- Given a function `wasExperimentSuccess` that runs a <u>single trial</u> of the experiment (e.g., flipping a coin) and returns `true` if the event occurred and `false` if it did not, the following function will perform the experiment the specified trials number of times and provide you with the empirical probability, $\frac{number\ of\ times\ event\ occured}{number\ of\ trials}$:

```
let empiricalProbability = (wasExperimentSuccess, trials) => {
  let numberOfTimesEventOccurred = range(0, trials)
    .repeatForEveryItem((numOfSuccessesSoFar, trialNumber) => {
      if(wasExperimentSuccess()) return numOfSuccessesSoFar+1;
      else return numOfSuccessesSoFar;
    }).startingWithAccOf(0);
  return numberOfTimesEventOccurred/trials;
};
```

Template for Running Experiment (cont.)

```
let empiricalProbability = (wasExperimentSuccess, trials) => {
  let numberOfTimesEventOccurred = range(0, trials)
    .repeatForEveryItem((numOfSuccessesSoFar, trialNumber) => {
      if(wasExperimentSuccess()) return numOfSuccessesSoFar+1;
      else return numOfSuccessesSoFar;
    }).startingWithAccOf(0);
  return numberOfTimesEventOccurred/trials;
};
```

- Here, the accumulator (`numOfSuccessesSoFar`) is used to keep track of the number of successes ***before*** `trialNumber`. Because of this, we start it at 0 (0 successes before trial 0).
- The computer performs the experiment using `wasExperimentSuccess` and if it returns true, it bumps up the counter; if not, it leaves it alone. This is done trials number of times: `range(0, trials)`.
- The number of times the event occurred is then divided by trials to get the approximate probability, which is then outputted.

Viewing table with `repeatForEveryItem`

- Every time the accumulator (`numOfSuccessesSoFar`) increases by 1, the experiment must be a success (`wasExperimentSuccess() returns true`)

<u>EXAMPLE OUTPUT – Event occurring 50% of the time – 10 TIMES</u>

```
console.log(empiricalProbability(()=>Math.random()<.5,10)):
ACCUMULATOR | ITEM (trialNumber)
-------------------
0           | 0    => 0 successes to start (before trial 0)
0           | 1    => 0 successes before trial 1 (trial 0 ✗)
1           | 2    => 1 success   before trial 2 (trial 1 ✓)
1           | 3    => 1 success   before trial 3 (trial 2 ✗)
1           | 4    => 1 success   before trial 4 (trial 3 ✗)
2           | 5    => 2 successes before trial 5 (trial 4 ✓)
3           | 6    => 3 successes before trial 6 (trial 5 ✓)
3           | 7    => 3 successes before trial 7 (trial 6 ✗)
3           | 8    => 3 successes before trial 8 (trial 7 ✗)
4           | 9    => 4 successes before trial 9 (trial 8 ✓)
0.5    => FINAL ANSWER: 5/10 (5 successes: trial 9 ✓)
```

Viewing table with `repeatForEveryItem`

- **NOTE**: Need to replace `repeatForEveryItem` with `repeatForEveryItemAndShowSteps` to see table; your output will likely be different.

EXAMPLE OUTPUT - Event occurring 50% of the time - 10 TIMES

```
console.log(empiricalProbability(()=>Math.random()<.5,10)):
ACCUMULATOR | ITEM (trialNumber)
------------------
0           | 0    => 0 successes to start (before trial 0)
0           | 1    => 0 successes before trial 1 (trial 0 ✗)
1           | 2    => 1 success   before trial 2 (trial 1 ✓)
1           | 3    => 1 success   before trial 3 (trial 2 ✗)
1           | 4    => 1 success   before trial 4 (trial 3 ✗)
2           | 5    => 2 successes before trial 5 (trial 4 ✓)
3           | 6    => 3 successes before trial 6 (trial 5 ✓)
3           | 7    => 3 successes before trial 7 (trial 6 ✗)
3           | 8    => 3 successes before trial 8 (trial 7 ✗)
4           | 9    => 4 successes before trial 9 (trial 8 ✓)
0.5   => FINAL ANSWER: 5/10 (5 successes: trial 9 ✓)
```

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).

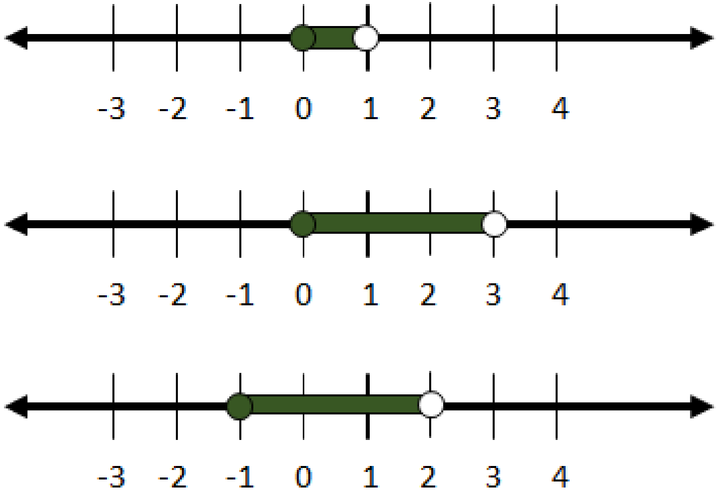
Generating pseudorandom numbers for other ranges

- 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)