Probability and Computing

Fall 2009 version of Course 15-359,

Computer Science Department,

Carnegie Mellon University.

Lecture notes by Ryan O'Donnell

Acknowledgments:

CMU's course 15-359, Probability and Computing, was originally conceived and designed by Mor Harchol-Balter and John Lafferty. The choice, order, and presentation of topics in the latter half of the course is strongly informed by the work of Mor Harchol-Balter. Indeed, you might like to buy her book! *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*, http://www.cs.cmu.edu/~harchol/PerformanceModeling/book.html
The choice, order, and presentation of topics in the earlier half of the course is informed by the work of John Lafferty: http://galton.uchicago.edu/~lafferty/
Further, a very great deal of material in these lecture notes was strongly informed by the outstanding book *Probability and Computing* by Michael Mitzenmacher and Eli Upfal, http://www.cambridge.org/us/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/probability-and-computing-randomized-algorithms-and-probabilistic-analysis
Many thanks to Mor Harchol-Balter, John Lafferty, Michael Mitzenmacher, Eli Upfal (and many other web sources from which I borrowed)!

15-359: Probability and Computing
Fall 2008

Lecture 1: Probability in Computing; Verifying matrix multiplication

1 Administrivia

The course web site is: `http://15-359.blogspot.com`. Please read carefully the Policies document there, which will also be handed out in class. This is also a blog, so please add it to your RSS reader, or at least read it frequently. Important announcements — and more! — will appear there.

2 Probability and computing

Why is probability important for computing? Why is randomness important for computing? Here are some example applications:

Cryptography. Randomness is essential for all of crypto. Most cryptographical algorithms involve the parties picking secret keys. This *must* be done randomly. If an algorithm deterministically said, "Let the secret key be 8785672057848516," well, of course that would be broken.

Simulation. When writing code to simulate, e.g., physical systems, often you model real-world events as happening randomly.

Statistics via sampling. Today we often work with huge data sets. If one wants to approximate basic statistics, e.g., the mean or mode, it is more efficient to sample a small portion of the data and compute the statistic, rather than read all the data. This idea connects to certain current research topics: "property testing algorithms" and "streaming algorithms".

Learning theory. Much of the most successful work in AI is done via learning theory and other statistical methods, wherein one assumes the data is generated according to certain kinds of probability distributions.

Systems & queueing theory. When studying the most effective policies for scheduling and processor-sharing, one usually models job sizes and job interarrival times as coming from a stochastic process.

Data compression. Data compression algorithms often work by analyzing or modeling the underlying probability distribution of the data, or its information-theoretic content.

Error-correcting codes. A large amount of the work in coding theory is based on the problem of redundantly encoding data so that it can be recovered if there is random noise.

Data structures. When building, e.g., a static dictionary data structure, one can optimize response time if one knows the probability distribution on key lookups. Even moreso, the time for operations in hash tables can be greatly improved by careful probabilistic analysis.

Symmetry-breaking. In distributed algorithms, one often needs a way to let one of several identical processors "go first". In combinatorial optimization algorithms — e.g., for solving TSP or SAT instances — it is sometimes effective to use randomness to decide which city or variable to process next, especially on highly symmetric instances.

Theory of large networks. Much work on the study of large networks — e.g., social networks like Facebook, or physical networks, like the Internet — models the graphs as arising from special kinds of random processes. Google's PageRank algorithm is famously derived from modeling the hyperlinks on the internet as a "Markov chain".

Quantum computing. The laws of physics are quantum mechanical and there has been tremendous recent progress on designing "quantum algorithms" that take advantage of this (even if quantum computers have yet to be built). Quantum computing is inherently randomized — indeed, it's a bit like computing with probabilities that can be both positive and negative.

Statistics. Several areas of computing — e.g., Human-Computer Interaction — involve running experimental studies, often with human subjects. Interpreting the results of such studies, and deciding whether their findings are "statistically significant", requires a strong knowledge of probability and statistics.

Games and gambling. Where would internet poker be without randomness?

Making algorithms run faster. Perhaps surprisingly, there are several examples of algorithmic problems which seem to have nothing to do with randomness, yet which we know how to solve much more efficiently using randomness than without. This is my personal favorite example of probability in computing.

We will see an example of using randomness to make algorithms more efficient today, in the problem of *verifying matrix multiplication*.

3 About this course

This course will explore several of the above uses of probability in computing. To understand them properly, though, you will need a thorough understanding of probability theory. Probability is traditionally a "math" topic, and indeed, this course will be very much like a math class. The emphasis will be on rigorous definitions, proofs, and theorems. Your homework solutions will be graded according to these "mathematical standards".

One consequence of this is that the first part of the course may be a little bit dry, because we will spend a fair bit of time going rigorously through the basics of probability theory. But of course it is essential that you go through these basics carefully so that you are prepared for the more advanced applications. I will be interspersing some "computer science" applications throughout

this introductory material, to try to keep a balance between theory and applications.

In particular, today we're going to *start* with an application, verifying matrix multiplication. Don't worry if you don't understand all the details today; it would be a bit unfair of me to insist you do, given that we haven't even done the basic theory yet! I wanted to give you a flavor of things to come, before we get down to the nitty-gritty of probability theory.

4 Verifying matrix multiplication

Let's see an example of an algorithmic task which a probabilistic algorithm can perform much more efficiently than any deterministic algorithm we know. The task is that of *verifying* matrix multiplication.

4.1 Multiplying matrices

There exist extremely sophisticated algorithms for multiplying two matrices together. Suppose we have two $n \times n$ matrices, A and B. Their product, $C = AB$, is also an $n \times n$ matrix, with entries given by the formula

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}. \tag{1}$$

How long does it take to compute C? Since C has n^2 entries, even just writing it down in memory will take at least n^2 steps. On the other hand, the "obvious" method for computing C given by (1) takes about n^3 steps; to compute each of the n^2 entries it does roughly n many arithmetic operations.

Actually, there may be some extra time involved if the numbers involved get very large; e.g., if they're huge, it could take a lot of time just to, say, multiply two of them together. Let us eliminate this complication by just thinking about:

Matrix multiplication mod 2. Here we assume that the entries of A and B are just bits, 0 or 1, and we compute the product mod 2:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \pmod 2. \tag{2}$$

This way, every number involved is just a bit, so we needn't worry about the time of doing arithmetic on numbers. The "obvious" matrix multiplication algorithm therefore definitely takes at most $O(n^3)$ steps.[1]

As you may know, there are very surprising, nontrivial algorithms that multiply matrices in time faster than $O(n^3)$. The first and perhaps most famous is *Strassen's algorithm*:

Theorem 1. *(Strassen, 1969.) It is possible to multiply two matrices in time roughly* $n^{\log_2 7} \approx n^{2.81}$.

[1] By the way, if you are not so familiar with big-Oh notation and analysis of running time, don't worry too much. This is not an algorithms course, and we won't be studying algorithmic running time in too much detail. Rather, we're just using it to motivate the importance of probability in algorithms and computing.

Incidentally, I've heard from Manuel Blum that Strassen told him that he thought up his algorithm by first studying the simpler problem of matrix multiplication mod 2. (Strassen's algorithm works for the general case, non-mod-2 case too.)

There were several improvements on Strassen's algorithm, and the current "world record" is due to Coppersmith and Winograd:

Theorem 2. *(Coppersmith-Winograd, 1987.) It is possible to multiply two matrices in time roughly* $n^{2.376}$.

Many people believe that matrix multiplication can be done in time $n^{2+\epsilon}$ for any $\epsilon > 0$, but nobody knows how to do it.

4.2 Verification

We are not actually going to discuss algorithms for matrix multiplication. Instead we will discuss *verification* of such algorithms.

Suppose your friend writes some code to implement the Coppersmith-Winograd algorithm (mod 2). It takes as input two $n \times n$ matrices of bits A and B and outputs some $n \times n$ matrix of bits, D. The Coppersmith-Winograd algorithm is very complicated, and you might feel justifiably concerned about whether your friend implemented it correctly. How could you test this?

Well, to test $AB \stackrel{?}{=} D \pmod 2$, you could write the "obvious" algorithm for multiplying A and B, and check that the answer is D. But this takes time n^3. You could also write your *own* implementation of Coppersmith-Winograd really carefully, and use it to "double-check" your friend's results. But this still takes time $n^{2.376}$, and is equally complicated. It only really makes sense to have separate code for "verifying" $AB \stackrel{?}{=} D \pmod 2$ if that code is simpler and more efficient than just multiplying A and B.

Fact 3. *The fastest known* deterministic *algorithm for checking* $AB \stackrel{?}{=} D \pmod 2$ *takes time about* $n^{2.376}$.

Theorem 4. *(Rusins Freivalds, 1979.) There is a probabilistic algorithm which checks whether* $AB \stackrel{?}{=} D \pmod 2$ *in only* $O(n^2)$ *steps. On every input, the probability the algorithm gives the wrong answer is at most* 2^{-200}.

Admittedly, this algorithm has a small probability of failure, unlike correct deterministic algorithms. But please note that this probability is really, ***really*** small: far far smaller than the probability of a computer's hardware error, and indeed far smaller than the probability of a meteor destroying the earth in the next second. If you ran this algorithm once every "Planck time" for the entire age of the universe, it might fail around once :)

4.3 Freivalds's algorithm

Freivalds's algorithm is about as simple as possible:

Algorithm "Basic-Freivalds" On input A, B, and D:

1. Choose a random n-bit vector x, by making each bit x_i either 0 or 1 "independently", "with probability 1/2" each.
2. Compute $y = ABx$ and $z = Dx$. Output "yes" if $y = z$.

We will discuss later what is meant precisely by our choosing x "randomly" in this way. But for now I will assume that you've seen a little bit of probability before, and you know intuitively what this means.

Before getting to probability, let's first confirm the running time:

Proposition 5. *Basic-Freivalds can be executed in $O(n^2)$ steps.*

Proof. Step 1 involves writing down the n bits of x: $O(n)$ steps. What about Step 2? Let's start with computing $z = Dx$. This is a matrix-vector product; if we use the "obvious" formula,

$$z_i = \sum_{j=1}^{n} D_{ij}x_j \pmod 2,$$

the computation will take $O(n^2)$ steps — $O(n)$ steps per entry of z.

To compute $y = ABx$, we think of it as $y = A(Bx)$. We can compute $w = Bx$ in $O(n^2)$ steps, and then compute $y = Aw$ in another $O(n^2)$ steps. □

Note that this algorithm is faster than the $n^{2.37}$ time to multiply two matrices, and also that it is very very simple to implement correctly! That said, does Basic-Freivalds actually properly check $AB \stackrel{?}{=} D \pmod 2$? There are two cases to check:

First: When $AB = D \pmod 2$, does the algorithm output "yes"?

Second: When $AB \neq D \pmod 2$, does the algorithm output "no"?

Proposition 6. *When $AB = D \pmod 2$, Basic-Freivalds* always *outputs "yes".*

Proof. This is obvious. If $AB = D \pmod 2$, then $ABx = Dx \pmod 2$ for *any* vector x, so the algorithm will always have $y = z$. □

Here is the amazing aspect of Basic-Freivalds:

Theorem 7. *For* any *input matrices A, B, and D such that $AB \neq D \pmod 2$, the Basic-Freivalds algorithm will output "yes" with probability* at most 1/2.

I.e., when $AB \neq D \pmod 2$, the probability Basic-Freivalds gives the wrong answer is at most 1/2. Now this doesn't look like what was promised, namely a failure probability of at most 2^{-200}. But there is a simple way to deliver on the promise: just repeat the Basic-Freivalds algorithm a bunch of times.

Freivalds's Algorithm: On input A, B, and D, run the Basic-Freivalds algorithm 200 times. If it ever outputs "no", then output "no". If it always outputs "yes", then output "yes".

The total running time for Freivalds's Algorithm is $200 \times O(n^2) \leq O(n^2)$ steps. Again, we should check two cases when analyzing correctness:

Theorem 8. *When $AB = D \pmod 2$, Freivalds's Algorithm always outputs "yes". On any input with $AB \neq D \pmod 2$, Freivalds's Algorithm outputs "yes" (i.e., acts incorrectly) with probability at most 2^{-200}.*

Proof. The first case is clear: when $AB = D \pmod 2$, Basic-Freivalds always outputs "yes", so the overall Freivalds Algorithm will output "yes". As for the second case, suppose $AB \neq D \pmod 2$. By Theorem 7, the probability one run of Basic-Freivalds outputs "yes" is $1/2$. Therefore, the probability that all 200 runs output "yes" is at most

$$\frac{1}{2} \times \frac{1}{2} \times \cdots (200 \text{ times}) \cdots \times \frac{1}{2} = 2^{-200}.$$

□

Actually, justifying this proof requires understanding the basics of probability, and we will develop these basics properly only over the next few lectures. I hope, though, that you've seen a little probability before and that you intuitively agree with the above proof.

By the way, this repetition of the Basic-Freivalds test illustrates a common idea in probabilistic computing: trading off running time and correctness probability.

5 Analyzing Basic-Freivalds, and other randomized code

Let's now prove Theorem 7 — i.e., analyze the Basic-Freivalds algorithm when $AB \neq D \pmod 2$. In fact, this naturally leads to a basic theme for the first part of this course:

PROBABILITY THEORY = ANALYZING RANDOMIZED CODE.

This might look surprising, as the left-hand side seems to be a math topic and the right-hand side seems to be a computer science topic. We'll return to this.

Let's write out the Basic-Freivalds code a little differently to highlight the *randomized* component:

```
Input A, B, D
x[1] ← Bernoulli(.5)
x[2] ← Bernoulli(.5)
...
x[n] ← Bernoulli(.5)
Output ''yes'' if ABx = Dx (mod 2), output ''no'' if ABx ≠ Dx (mod 2)
```

You might ask, what the heck is "`Bernoulli`(.5)"? This will be our notation for a built-in function which returns 0 with probability .5 and 1 with probability .5. Basically, it's a "fair coin

flip". For some reason, the mathematicians named this basic built-in function after the 17th century Swiss mathematician Jacob Bernoulli. We throw them a bone by using their terminology.

The analysis of Basic-Freivalds uses a fairly clever "trick". The first step will be to rewrite the last step a little:

$$\begin{aligned} ABx \neq Dx \quad &\Leftrightarrow \quad ABx - Dx \neq \vec{0} \quad \Leftrightarrow \quad (AB-D)x \neq \vec{0} \\ &\Leftrightarrow \quad (AB-D)x \pmod 2 \text{ has a 1 in at least one coordinate.} \end{aligned}$$

So we can rewrite the last line of the algorithm as

`Output ''no'' if and only if` $(AB-D)x \pmod 2$ `has a 1 in at least one coordinate.`

Given A, B, D, let

$$E = AB - D \pmod 2.$$

Remember, our goal is to prove Theorem 7; in other words:

Goal: *show that when* $AB \neq D \pmod 2$*, output is "no" with probability* at least $1/2$.

Of course,

$$\begin{aligned} AB \neq D \pmod 2 \quad &\Leftrightarrow \quad AB - D \neq 0 \pmod 2 \quad \Leftrightarrow \quad E \neq 0 \pmod 2 \\ &\Leftrightarrow \quad E_{s,t} = 1 \pmod 2 \text{ for at least one pair } 1 \leq s,t \leq n. \end{aligned}$$

So to achieve the goal, it suffices to show the following:

Theorem 9. *Let E be any nonzero $n \times n$ matrix of bits; say $E_{s,t} = 1$. Then when the n-bit vector x is chosen randomly as in the code,*

$$\mathbf{Pr}[\textit{the vector } Ex \textit{ mod } 2 \textit{ has a } 1 \textit{ in at least one coordinate}] \geq 1/2.$$

Proof. Just so we don't get too much notation, let's suppose that $s = 5$, $t = 3$, say. I leave it to you to write out the proof for general s and t. So we know

$$E_{5,3} = 1.$$

We will actually show something stronger than what we need, namely,

$$\mathbf{Pr}[\text{the vector } Ex \text{ mod } 2 \text{ has a 1 in the 5th coordinate}] = 1/2.$$

I.e., already the 5th coordinate has a $1/2$ chance of being 1, and so the chance that at least one coordinate has a 1 is at least $1/2$.

Think about what the 5th coordinate of $Ex \pmod 2$ is: it's the 5th row of E dot-producted with the vector $x \pmod 2$. In other words:

$$(Ex)_5 = E_{5,1} \cdot x[1] + E_{5,2} \cdot x[2] + E_{5,3} \cdot x[3] + E_{5,4} \cdot x[4] + \cdots + E_{5,n} \cdot x[n] \pmod 2.$$

We are assuming $E_{5,3} = 1$, so let's put that in, and rearrange a little:

$$(Ex)_5 = \Big(E_{5,1} \cdot x[1] + E_{5,2} \cdot x[2] + E_{5,4} \cdot x[4] + \cdots + E_{5,n} \cdot x[n]\Big) + x[3] \mod 2.$$

Now comes the main trick. Observe that the following two blocks of code clearly have the same behavior:

$x[1] \leftarrow$ `Bernoulli(.5)`
$x[2] \leftarrow$ `Bernoulli(.5)`
$\cdots$
$x[n] \leftarrow$ `Bernoulli(.5)`
`answer` $\leftarrow E_{5,1} \cdot x[1] + E_{5,2} \cdot x[2] + E_{5,3} \cdot x[3] + E_{5,4} \cdot x[4] + \cdots + E_{5,n} \cdot x[n] \pmod 2$

versus

$x[1] \leftarrow$ `Bernoulli(.5)`
$x[2] \leftarrow$ `Bernoulli(.5)`
$x[4] \leftarrow$ `Bernoulli(.5)`
$x[5] \leftarrow$ `Bernoulli(.5)`
$\cdots$
$x[n] \leftarrow$ `Bernoulli(.5)`
`temp` $\leftarrow E_{5,1} \cdot x[1] + E_{5,2} \cdot x[2] + E_{5,4} \cdot x[4] + \cdots + E_{5,n} \cdot x[n] \pmod 2$
$x[3] \leftarrow$ `Bernoulli(.5)`
`answer` $\leftarrow$ `temp` $+ x[3] \pmod 2$

In other words, it doesn't actually matter what order we pick the $x[i]$'s in, so we can imagine $x[3]$ being picked last. The point now is that *it doesn't matter what* `temp` turns out to be; just based on the random choice of $x[3]$ we'll have that `answer` is 1 with probability exactly $1/2$. This is because both $0 + x[3] \pmod 2$ and $1 + x[3] \pmod 2$ are equally likely to be 0 or 1 when we choose $x[3] \leftarrow$ `Bernoulli(.5)`. □

Neat trick, huh?

15-359: Probability and Computing
Fall 2009

Lecture 2: Discrete probability modeling; probability of events; conditioning

In this lecture we will learn the basics of probability — sample spaces, events, conditioning, etc. Our basic framework will have a "computer science-y" slant. In my opinion, this is the right way to think about probability — even if you're a mathematician with no interest in computer science.

1 Probability Modeling

1.1 Modeling with computer code

Probability is all about modeling reality with math. (Actually, this is what all of math is about.) But the way math people teach and model probability is often not so great. They do probability modeling like this:

$$\text{Description of reality} \quad \rightarrow \quad \text{Math.}$$

This often leads to vagueness (e.g., what exactly is a "random experiment") or ambiguities (think of the "Monty Hall Problem").

The right way to do probability is the computer science way: always be *analyzing randomized code*! This is how we will do all our probability modeling:

$$\text{Description of reality} \quad \rightarrow \quad \text{Computer code} \quad \rightarrow \quad \text{Math}$$

And often, the reality will *be* the computer code. The beauty of this is that any ambiguity in the English-language description can be debated when doing the first step, modeling the real-world situation with randomized code. But once the randomized code is decided on, there will be an unambiguous and clear-cut way to analyze it mathematically.

Let's explain. You know how all probability exercises start with something like, "A person flips 2 fair coins..."? We will immediately model this with code:

```
coin1 ← Bernoulli(1/2)
coin2 ← Bernoulli(1/2)
```

(Well, I suppose `Bernoulli`(.5) returns 0 or 1, but you can add in statements like

```
if coin1 = 0 then coin1 ← ‘‘tails’’ else coin1 ← ‘‘heads’’
```

if you like. But we won't care much about such niceties.)

1.2 "Built-in" random functions

Recall from last lecture we had the following:

ASSUMPTION: We have a built-in random function, `Bernoulli`$(1/2)$ which returns 0 with probability $1/2$ and 1 with probability $1/2$.

More implicitly, we also have the following:

ASSUMPTION: Multiple calls to built-in random functions like `Bernoulli`$(1/2)$ behave *"independently"*; the return value of one call in no way affects the return value of any other call.

Are these assumptions "realistic"? For the purposes of doing math (and for the purposes of this course, really), it doesn't matter; these will be our basic, idealized, axioms. In the reality of computer code, they are *NOT* really true. When you call the equivalent of `Bernoulli`$(1/2)$ in your favorite programming language, it will actually return a number from a stream of "pseudorandom numbers", whatever that means. You won't be getting 0 or 1 with probability *exactly* $1/2$ each, and furthermore, the "independence assumption" is *definitely wrong*; if you know the past history of what the pseudorandom number generators have returned, this gives you partial information about what the next return value will be. We will discuss pseudorandom number generators further in a later lecture.

In fact, one may forget about computer code all together and go straight to the physical world. Is there anything in nature which (according to the laws of physics) acts like `Bernoulli`$(1/2)$? And does it satisfy the "independence" assumption? These are deep questions of physics, engineering, and philosophy. We can discuss them later — let's get back to the basics of probability for now!

What about other basic probability exercise classics? E.g., "A person rolls a fair die..."?[1] To model this, I will grant you:

ASSUMPTION: We have a built-in random function, `RandInt`(m), which given $m \in \mathbb{N}$, returns one of the integers $\{1, 2, \ldots, m\}$, each with equal probability $\frac{1}{m}$.

You should immediately model the real-world event as though you were "simulating" it, with code:

```
die ← RandInt(6).
```

As one more example, an old-school probability exercise might begin, "A patient has a 1 in 10 chance of having a certain disease..." Again, write it like a computer simulation:

```
u ← RandInt(10)
if u = 1, patient.hasDisease ← 1
else patient.hasDisease ← 0
```

Actually, the above kind of thing comes up a lot; let's assume we have a built-in function for it:

[1] Dice are some of probabilists' favorite items. In case you didn't know: one of these six-sided cubes is called a "die"; multiple such cubes are "dice".

ASSUMPTION: We have a built-in random function, `Bernoulli`(p), which given a real $0 \leq p \leq 1$, returns

$$1 \quad \text{with probability } p,$$

$$0 \quad \text{with probability } 1-p.$$

Now the above code could simply be written as:

```
patient.hasDisease ← Bernoulli(.1)
```

1.3 "Discrete" randomness only for now

So far we've assumed access to two kinds of "built-in" random number generators: `Bernoulli`(p) for any real $0 \leq p \leq 1$, and `RandInt`(m) for any $m \in \mathbb{N}$. On the homework, you will explore how, using these tools, you can build up other random number-generating functions such as

$$\texttt{Rand}\begin{pmatrix} 1 & \text{with probability } .2 \\ 5 & \text{with probability } .3 \\ -3 & \text{with probability } .5 \end{pmatrix}.$$

Really, anything where there are *finitely* many different outcomes with nonnegative probabilities adding up to 1.

What we are absolutely **disallowing** for the first half of the course is the use of a random number generator like "`Unif`$([0,1])$", which returns a uniformly random real number between 0 and 1 — whatever that means!

Why do we disallow this? As we are about to see, to analyze randomized code we will be drawing trees, with a node for each call to a random number generator and a branch for each possible return value it may have. It will be mighty hard to draw such a tree if it is to have infinitely (uncountably!) many branches, as it would be with "`Unif`$([0,1])$". Seriously, random processes with uncountably infinitely many different outcomes require significantly more complicated analysis, which it's best to postpone for later.

You might be disappointed, since every standard programming language (claims to) have a random number generator like "`Unif`$([0,1])$". Of course, you also know that you can't truly store a real number in a computer, so *actually*, these random number generators do something like returning

`RandInt`$(2^{64})/2^{64}$.

Indeed, as we will see in the second half of the class, it's a good idea to think about analyzing "`Unif`$([0,1])$" in precisely this way — except we use "dx" rather than 2^{-64}...

2 Analyzing randomized code & Basic definitions

So we've decided we will model probabilistic stuff with randomized code; for example, something like the following:

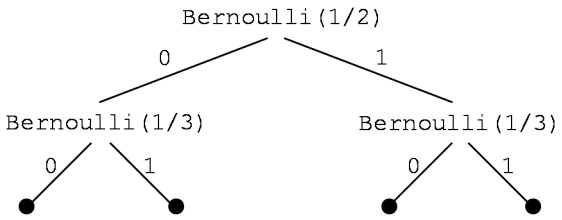
```
flip1 ← Bernoulli(1/2)
```

```
flip2 ← Bernoulli(1/3)
```

This little block of code models a standard probability exercise beginning with, "You flip 2 coins, the first of which is fair, the second having bias 1/3 for heads..." In mathematics, they make the following definition:

Definition: An *experiment* is a block of randomized code.

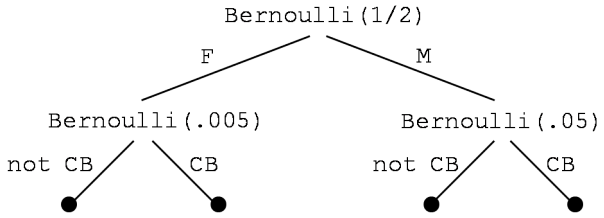
Here is how you analyze experiments/code: You build a "probability tree", with branching *for each call to a random number generator*. For the above example:



Or, a probability exercise might begin, "An ophthalmology patient has a 5% chance of being color blind if male, and a .5% chance otherwise..."' We first write the code:

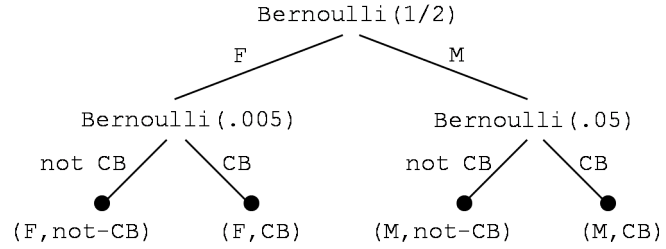
```
isMale ← Bernoulli(1/2)
if isMale, isColorBlind ← Bernoulli(.05)
else isColorBlind ← Bernoulli(.005)
```

We then draw the probability tree (in which it is okay to use "meaningful" names rather than 0 and 1):



Definition: An *outcome* is a *leaf* in the probability tree of an experiment. Equivalently, it is a possible sequence of all values returned by the random number generator calls.

For example, here is the above tree with its *outcomes* labeled:



Definition: The *sample space* of an experiment is the *set* of all outcomes; i.e., the set of leaves. It is often written Ω.

So for example, in the ophthalmology example,

$$\Omega = \{(\text{F, not-CB}), (\text{F, CB}), (\text{M, not-CB}), (\text{M, CB})\}.$$

In the other experiment with the two coins, the sample space is $\Omega = \{(0,0), (0,1), (1,0), (1,1)\}$, which we'd usually write with meaningful names as

$$\Omega = \{\text{TT}, \text{TH}, \text{HT}, \text{HH}\}.$$

Definition: The *probability of an outcome* is defined as follows: Given an outcome (leaf) ℓ, its probability, denoted $\mathbf{P}[\ell]$ or $\mathbf{Pr}[\ell]$ is the *product of the probabilities along the path to* ℓ.

For example, in the ophthalmology example we have

$$\mathbf{Pr}[(\text{F, CB})] = (1/2) \cdot (.005) = .0025,$$

$$\mathbf{Pr}[(\text{M, not-CB})] = (1/2) \cdot (.95) = .475.$$

Under this definition it is easy to see that we have the following two most basic facts about probability:

Fact: For all outcomes ℓ we have $\mathbf{Pr}[\ell] \geq 0$.

Fact: The total probability is always 1:

$$\sum_{\ell \in \Omega} \mathbf{Pr}[\ell] = 1.$$

(The second fact is easy to see, right? It's like you start with 100% of a universe at the top of a probability tree, and then each branch breaks it into fractions summing up to 1. So each branching maintains the fact that the sum of the probabilities at the leaves is 100%.)

3 Events

The next definition is a very important one, and perhaps one that takes a little getting used to:

Definition: An *event* is any collection of outcomes. I.e., it is a *subset* $A \subseteq \Omega$.

We also *overload* the $\mathbf{Pr}[\cdot]$ notation, so that the thing inside the $[\cdot]$ can be either an outcome or an event (set of outcomes):

Definition: If $A \subseteq \Omega$ is an event, we define the *probability* of A by

$$\mathbf{Pr}[A] = \sum_{\ell \in A} \mathbf{Pr}[\ell];$$

it's the sum of all the probabilities of the outcomes in A.

3.1 Example

Let's do an example. Here is the beginning of a typical probability exercise:

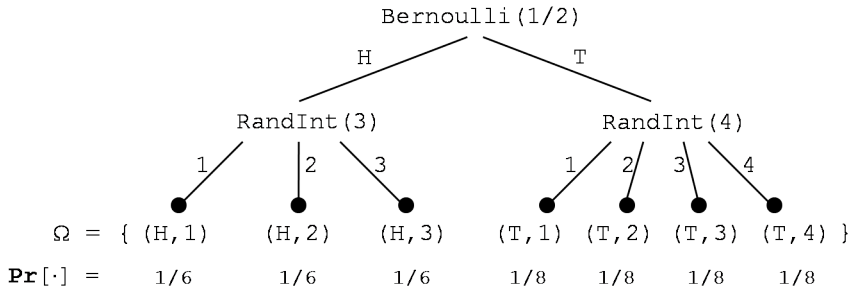
Flip a fair coin. If it comes up heads, roll a 3*-sided die; if it comes up tails, roll a* 4*-sided die.*

(Yes, we will talk about d-sided dice. These exist in real life for certain values of d, most notably $d = 6$; in probability-land, they exist for all d. Actually, this just goes to show it's simpler to talk about `RandInt`(d).)

The very first thing you should do is write, "We model this with the following code:

```
flip ← Bernoulli(1/2)
if flip = 1, die ← RandInt(3)
else         die ← RandInt(4)."
```

Great. Now you have an unambiguous random experiment to analyze. The next thing you should do is write, "This gives the following probability tree:" And you should label the outcomes and their probabilities.



Note that the sample space here, Ω, has cardinality 7. We computed the probabilities of the outcomes by multiplying the probabilities along each path; e.g.:

$$\mathbf{Pr}[(\mathrm{H}, 1)] = (1/2) \cdot (1/3) = 1/6,$$

$$\mathbf{Pr}[(\mathrm{T}, 3)] = (1/2) \cdot (1/4) = 1/8$$

and so forth.

Now we go back and continue reading the exercise. Suppose it continues:

What is the probability that the die roll is at least 3*?*

Here is how you answer this problem. First, you write, "Let A be the event of rolling at least 3. We have

$$A = \{(\mathrm{H}, 3), (\mathrm{T}, 3), (\mathrm{T}, 4)\}."$$

It's helpful to also circle this event on your probability tree. You then continue, "Therefore,

$$\mathbf{Pr}[A] = \mathbf{Pr}[(\mathrm{H}, 3)] + \mathbf{Pr}[(\mathrm{T}, 3)] + \mathbf{Pr}[(\mathrm{T}, 4)] = 1/6 + 1/8 + 1/8 = 5/12. \qquad \square"$$

Please memorize this fundamental process! It will be on your next quiz...

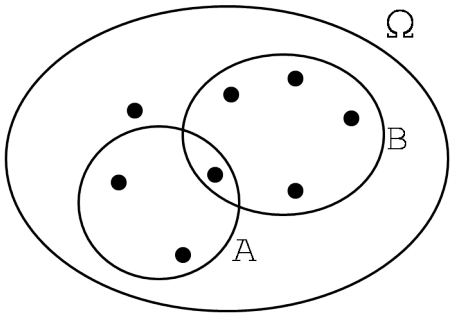
4 Basic facts

Here are some simple but important basic facts about the probabilities of events:

Facts:

- $A \subseteq B \Rightarrow \mathbf{Pr}[A] \leq \mathbf{Pr}[B]$
- $\mathbf{Pr}[A^c] = 1 - \mathbf{Pr}[A]$.
- $\mathbf{Pr}[A \cup B] = \mathbf{Pr}[A] + \mathbf{Pr}[B] - \mathbf{Pr}[A \cap B]$.

Here we have used some set theory notation: A^c denotes the complement of A, meaning $\Omega \setminus A$, or "not A"; $A \cup B$ is the union of A and B, meaning "A or B"; $A \cap B$ is the intersection of A and B, meaning "A and B". All of these facts are easy to prove, just by the definition of the probability of an event as the sum of the probabilities of the outcomes in it. The following diagram might be helpful; it shows a sample space Ω, with the outcomes as dots. Remember, each of these has a probability, and all the probabilities add up to 1. An event A is just a subset of outcomes, and its probability is the sum of their probabilities.



The third fact above is important to remember, in light of the common

$$\textbf{FALLACY: } \mathbf{Pr}[A \cup B] = \mathbf{Pr}[A] + \mathbf{Pr}[B].$$

Since $\mathbf{Pr}[A \cap B] \geq 0$, though, we *do* have the following:

Fact: $\mathbf{Pr}[A \cup B] \leq \mathbf{Pr}[A] + \mathbf{Pr}[B]$.

More generally, we have the very trivial but *extremely* useful fact called the "Union Bound":

$$\textbf{Union Bound:} \qquad \mathbf{Pr}[A_1 \cup A_2 \cup \cdots \cup A_n] \leq \sum_{i=1}^{n} \mathbf{Pr}[A_i].$$

It may seem indulgent to give a **bold-faced** name to such a simple fact, but please remember its name; it really comes up over and over and over again.

5 Conditioning

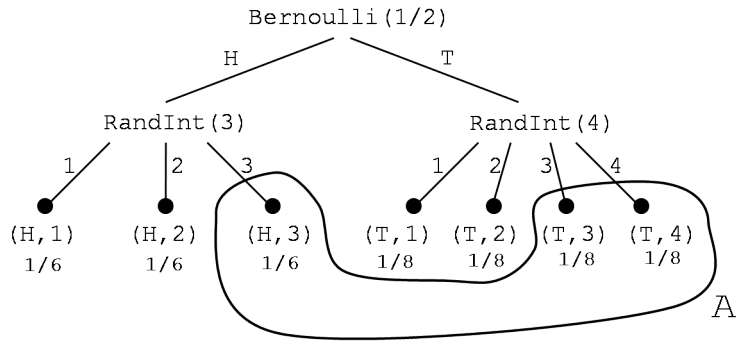
Conditioning is actually one of the trickier concepts in basic probability, but it is utterly essential, so it's important to learn it carefully. Roughly speaking:

Conditioning = Revising probabilities based on "partial information".

What is "partial information"?

Partial information = An event.

To illustrate, let's go back to the example from Section 3.1. In this example, let's "condition on the event A, rolling at least 3". Saying "condition on A" is like promising that you are in one of the "roll ≥ 3" outcomes. Let's circle those outcomes:



The idea now is that we want to *revise* our probabilities for *all* 7 outcomes, based on the "partial information" that event A occurred. The definition is pretty natural. Let's start with:

$$\mathbf{Pr}[(\mathrm{H}, 1) \mid A].$$

The "$\mid A$" part is pronounced "given A" or "conditioned on A" (these are synonyms). The whole expression is pronounced "the probability of outcome $(\mathrm{H}, 1)$ given A". And what should its value be? Clearly:

$$\mathbf{Pr}[(\mathrm{H}, 1) \mid A] = 0,$$

since if I promise you A occurred, i.e., the roll was at least 3, then the outcome $(\mathrm{H}, 1)$ definitely did not occur. Similarly,

$$\mathbf{Pr}[(\mathrm{T}, 2) \mid A] = 0.$$

On the other hand, what about, say, $\mathbf{Pr}[(\mathrm{H}, 3) \mid A]$? The way to think about it is as follows: Think of each random branch in the tree as taking us into one of many worlds. We know that in 5/12 of the worlds, the event A occurred. Now *among* those worlds, $(1/6)/(5/12)$ of them correspond to the outcome $(\mathrm{H}, 3)$. So we should have

$$\mathbf{Pr}[(\mathrm{H}, 3) \mid A] = \frac{1/6}{5/12} = \frac{2}{5}.$$

Similarly reasoning gives:

$$\mathbf{Pr}[(\mathrm{T}, 3) \mid A] = \frac{1/8}{5/12} = \frac{3}{10},$$

$$\mathbf{Pr}[(\mathrm{T}, 4) \mid A] = \frac{1/8}{5/12} = \frac{3}{10}.$$

What's happened is that "conditioning on A" has effectively given us a new "probability distribution" on the outcomes:

$$\begin{array}{rccccccc} \Omega = & \{ & (\mathrm{H},1), & (\mathrm{H},2), & (\mathrm{H},3), & (\mathrm{T},1), & (\mathrm{T},2), & (\mathrm{T},3), & (\mathrm{T},4) & \} \\ \mathbf{Pr}[\cdot] = & \{ & 0, & 0, & \frac{2}{5}, & 0, & 0, & \frac{3}{10}, & \frac{3}{10} & \} \end{array}$$

Now that we have new conditional probabilities for *outcomes*, we can define new conditional probabilities for *events*. For example, let B be the event of flipping a head:

$$B = \{(\mathrm{H},1), (\mathrm{H},2), (\mathrm{H},3)\} \subseteq \Omega.$$

Now just as before, we calculate the "probability of B given A" by adding up the (conditional) probabilities of the outcomes making up B:

$$\begin{aligned} \mathbf{Pr}[B \mid A] &= \mathbf{Pr}[(\mathrm{H},1) \mid A] + \mathbf{Pr}[(\mathrm{H},2) \mid A] + \mathbf{Pr}[(\mathrm{H},3) \mid A] \\ &= 0 + 0 + \frac{2}{5} \quad = \quad \frac{2}{5}. \end{aligned}$$

You may find it slightly weird to think about things like "$B \mid A$", namely, "the probability you flipped heads given that the roll was at least 3", but you'll get used to it :) If we're interested in the probability that you flipped *tails* given that the roll was at least 3, there's no need to add up conditional outcome probabilities: we can just use the "complement fact":

$$\mathbf{Pr}[\text{tails} \mid A] = \mathbf{Pr}[B^c \mid A] = 1 - \mathbf{Pr}[B \mid A] = 1 - \frac{2}{5} = \frac{3}{5}.$$

5.1 The official rules of conditioning

Having gone through an example, let's now give the official rules and definitions for conditioning:

Definition: Given an event $A \subseteq \Omega$ with $\mathbf{Pr}[A] \neq 0$, define the *conditional probability of outcome ℓ* (AKA the *probability of ℓ given A*) to be

$$\mathbf{Pr}[\ell \mid A] = \begin{cases} 0 & \text{if } \ell \notin A, \\ \frac{\mathbf{Pr}[\ell]}{\mathbf{Pr}[A]} & \text{if } \ell \in A. \end{cases}$$

What about the conditional probability of an event? Well, $\mathbf{Pr}[B \mid A]$ is the sum of conditional probabilities of all outcomes in B. Hence we claim:

Corollary 1.

$$\mathbf{Pr}[B \mid A] = \frac{\mathbf{Pr}[B \cap A]}{\mathbf{Pr}[A]} = \frac{\mathbf{Pr}[A \cap B]}{\mathbf{Pr}[A]}$$

Proof. The second equality is of course just because $B \cap A = A \cap B$. As for the first,

$$\begin{aligned} \mathbf{Pr}[B \mid A] &= \sum_{\ell \in B} \mathbf{Pr}[\ell \mid A] \\ &= \sum_{\ell \in B \cap A} \mathbf{Pr}[\ell \mid A] + \sum_{\ell \in B \setminus A} \mathbf{Pr}[\ell \mid A] \\ &= \sum_{\ell \in B \cap A} \frac{\mathbf{Pr}[\ell]}{\mathbf{Pr}[A]} + \sum_{\ell \in B \setminus A} 0 \\ &= \frac{1}{\mathbf{Pr}[A]} \sum_{\ell \in B \cap A} \mathbf{Pr}[\ell] \\ &= \frac{\mathbf{Pr}[B \cap A]}{\mathbf{Pr}[A]}. \end{aligned}$$

□

Of course, just by interchanging A and B, we have that (assuming $\mathbf{Pr}[B] \neq 0$)

$$\mathbf{Pr}[A \mid B] = \frac{\mathbf{Pr}[A \cap B]}{\mathbf{Pr}[B]}.$$

If you're going to memorize this formula, the important thing is that the event you're conditioning on goes into the denominator on the right.

Let's see these definitions in action one more time. Remember that we showed $\mathbf{Pr}[\text{heads} \mid \text{roll} \geq 3] = 2/5$. What about the reverse, $\mathbf{Pr}[\text{roll} \geq 3 \mid \text{heads}]$? Well, if you think about it, "obviously" the answer should be $1/3$. Let's check:

$$\begin{aligned}
\mathbf{Pr}[\text{roll} \geq 3 \mid \text{heads}] &= \frac{\mathbf{Pr}[\text{heads} \cap \text{roll} \geq 3]}{\mathbf{Pr}[\text{heads}]} \\
&= \frac{\mathbf{Pr}[(\text{H},3)]}{\mathbf{Pr}[(\text{H},1)] + \mathbf{Pr}[(\text{H},2)] + \mathbf{Pr}[(\text{H},3)]} \\
&= \frac{1/6}{1/6 + 1/6 + 1/6} \quad = \quad 1/3.
\end{aligned}$$

One other corollary of the conditioning formula is the following:

Formula: $\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B \mid A]$ (assuming $\mathbf{Pr}[A] \neq 0$).

We often use this formula to *calculate* the probability of an intersection — i.e., the "and" of two events. We use the following phraseology: "For A *and* B to happen, first A has to happen (which has probability $\mathbf{Pr}[A]$), and then also B has to happen *given* that A happened (which has probability $\mathbf{Pr}[B \mid A]$).]]

The above formula generalizes to what is occasionally called the "Chain Rule" or the "Multiplication Rule":

Formulas:

$$\mathbf{Pr}[A \cap B \cap C] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B \mid A] \cdot \mathbf{Pr}[C \mid A \cap B], \quad \text{and more generally,}$$

$$\mathbf{Pr}[A_1 \cap A_2 \cap \cdots \cap A_n] = \mathbf{Pr}[A_1] \cdot \mathbf{Pr}[A_2 \mid A_1] \cdot \mathbf{Pr}[A_3 \mid A_1 \cap A_2] \cdots \mathbf{Pr}[A_n \mid A_1 \cap A_2 \cap \cdots \cap A_{n-1}]$$

(assuming all the conditioned-on events have nonzero probability). We leave the proof of this as an exercise for you.

15-359: Probability and Computing
Fall 2009

Lecture 3: Law of Total Probability, independence. Picking random primes.

In today's lecture we will finish the "basic theory" of events, the last theory topic before we move on to random variables in the next lecture. After this, we will discuss some applications of probability in number theory and cryptography.

1 Law of Total Probability

Last lecture we introduced conditioning. When we discussed it then, it might have seemed that first you figure out all the probabilities involved in a random experiment, and *then* you can start calculating conditional probabilities. This is actually not at all how it works in practice; rather, in practice, you use conditioning to *help* you calculate the basic probabilities of events.

The most important formula involved is sometimes called the "Law of Total Probability". It's used *all the time*. To state it, we first need some setup:

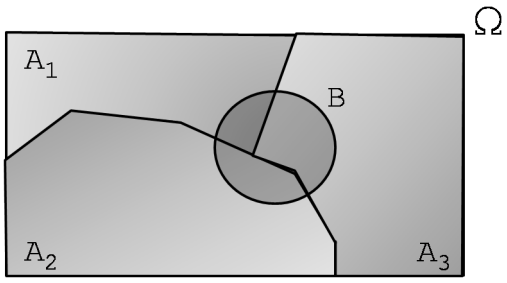
Definition: We say that events $A_1, \ldots, A_n$ form a "partition" of the sample space Ω if:

- They are "mutually exclusive", meaning $A_i \cap A_j = \emptyset$ for all $i \neq j$. In other words, *at most* one event A_i ever happens.
- They are "collectively exhaustive", meaning $A_1 \cup A_2 \cup \cdots \cup A_n = \Omega$. In other words, *at least* one A_i always happens.

Law of Total Probability: Suppose $A_1, \ldots, A_n$ form a partition of Ω. Then for any event $B \subseteq \Omega$,

$$\mathbf{Pr}[B] = \mathbf{Pr}[A_1] \cdot \mathbf{Pr}[B \mid A_1] + \mathbf{Pr}[A_2] \cdot \mathbf{Pr}[B \mid A_2] + \cdots + \mathbf{Pr}[A_n] \cdot \mathbf{Pr}[B \mid A_n].$$

(Note: Technically, we should also add "assume also that $\mathbf{Pr}[A_i] \neq 0$ for all i", because you are not allowed to condition on an event with probability 0. Adding this caveat all the time is a bit annoying, so we'll stop doing it. Alternately, you can make the convention that $0 \cdot (\text{undefined}) = 0$, and then the above formula holds with no caveat!) Here is a picture to help illustrate things, with $n = 3$:



When I use this law, I always say something like the following:

What is the probability of B? Either A_1 happens, in which case we have the probability of B given A_1; or, A_2 happens, in which case we have the probability of B given A_2; or, ...; or else A_n must happen, in which case we have the probability of B given A_n.

The proof of this law is simply that the right-hand side equals $\mathbf{Pr}[A_1 \cap B] + \mathbf{Pr}[A_2 \cap B] + \cdots + \mathbf{Pr}[A_n \cap B]$, which is easily seen to be $\mathbf{Pr}[B]$ (draw a Venn diagram if you don't see it immediately). An important special case is when we just have the partition A, A^c:

Special Case: $\mathbf{Pr}[B] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B \mid A] + \mathbf{Pr}[A^c] \cdot \mathbf{Pr}[B \mid A^c]$.

Again, when I use this special case, I always say a phrase like the following:

Regarding the probability of B, either A happens (with probability $\mathbf{Pr}[A]$) in which case B occurs with probability $\mathbf{Pr}[B \mid A]$; or, A does not happen (with probability $\mathbf{Pr}[A^c] = 1 - \mathbf{Pr}[A]$) in which case B occurs with probability $\mathbf{Pr}[B \mid A^c]$

It might not be obvious that the Law of Total Probability is actually helpful for calculating probabilities — you might think that being able to compute quantities like $\mathbf{Pr}[B \mid A]$ and $\mathbf{Pr}[B \mid A^c]$ might be strictly harder than just computing $\mathbf{Pr}[B]$. But often this is not the case! Let's see an example...

1.1 Law of Total Probability — an example

This example is very similar to the key fact used in our analysis of Freivalds's Algorithm which we saw in Lecture 1: namely, the fact that if e is a nonzero vector of bits and x is a random vector of bits (each x_i being $\texttt{Bernoulli}(1/2)$) then $\mathbf{Pr}[e \cdot x \neq 0 \pmod 2] = 1/2$.

Problem: I roll 5 fair dice.[1] What is the probability that their sum is divisible by 6?

You *could* draw the whole probability tree, with 6^5 outcomes, but that looks challenging. Here is the "clever" solution:

Solution: The trick is to condition on the sum of the first 4 dice.[2] For each possible sum, $s = 4, 5, 6, \ldots, 24$, let A_s be the event that the first 4 dice sum to s. It's easy to see that $\{A_4, \ldots, A_{24}\}$ form a partition of Ω, and none has probability 0. Let B be the event that all 5 dice sum to a multiple of 6.

By the Law of Total Probability,

$$\mathbf{Pr}[B] = \mathbf{Pr}[A_4]\mathbf{Pr}[B \mid A_4] + \mathbf{Pr}[A_5]\mathbf{Pr}[B \mid A_5] + \cdots + \mathbf{Pr}[A_{24}]\mathbf{Pr}[B \mid A_{24}].$$

Now the trick is to resist the temptation to calculate each $\mathbf{Pr}[A_s]$! Instead, let's ask ourselves, what is $\mathbf{Pr}[B \mid A_4]$? Conditioned on the first four dice summing to 4, the final total will be either 5, 6, 7, 8, 9, or 10, each with probability 1/6. In exactly one case does B occur, when the total is 6. Hence $\mathbf{Pr}[B \mid A_4] = 1/6$.

[1] If unspecified, a die has 6 sides.

[2] By the way, you might ask what "the first 4 dice" means — in the problem statement, the dice are indistinguishable. But implicitly, we are imagining that we have modeled the experiment by "$\texttt{die1} \leftarrow \texttt{RandInt(6)}; \texttt{die2} \leftarrow \texttt{RandInt(6)}$; etc.".

What about $\mathbf{Pr}[B \mid A_5]$? Conditioned on A_5, the final total will be either 6, 7, 8, 9, 10, or 11, each with probability 1/6. Again, in exactly one case does B occur, when the total is 6. So again, $\mathbf{Pr}[B \mid A_5] = 1/6$.

Let's do one more, $\mathbf{Pr}[B \mid A_6]$. Conditioned on A_6, the final total will be either 7, 8, 9, 10, 11, or 12, each with probability 1/6. Once more, in exactly one case does B occur, when the total is 12. So $\mathbf{Pr}[B \mid A_6] = 1/6$.

(Note: here we have moved slightly away from rigorous definitions-based reasoning, towards more abstract reasoning. But you should always double-check arguments like this against the actual definitions. In this case, imagine the probability tree just for the first 4 rolls. There may be many several paths to outcomes in the event A_6. For each such outcome, we are reasoning that when the tree is extended to include the final roll, this final `RandInt`(6) splits into 6 outcomes, exactly one of which is in B. Therefore, one can reason that $\mathbf{Pr}[B \mid A_6]$ is indeed 1/6.)

Indeed, you can now probably see that for *every* value of s we have $\mathbf{Pr}[B \mid A_s] = 1/6$, because exactly one of the numbers $s+1$, $s+2$, ..., $s+6$ is divisible by 6. Therefore,

$$\begin{aligned}\mathbf{Pr}[B] &= \mathbf{Pr}[A_4] \cdot (1/6) + \mathbf{Pr}[A_5] \cdot (1/6) + \cdots + \mathbf{Pr}[A_{24}] \cdot (1/6) \\ &= (1/6) \cdot (\mathbf{Pr}[A_4] + \mathbf{Pr}[A_5] + \cdots + \mathbf{Pr}[A_{24}]) \\ &= 1/6. \qquad \square\end{aligned}$$

(We chose this problem because it is a good illustration of the Law of Total Probability. However, we must admit that it relies very heavily on the "trick" that $\mathbf{Pr}[B \mid A_s]$ is always 1/6 because we ask about divisibility by 6. What if we had asked for the probability that the final sum is divisible by 5? In that case, the only way to solve the problem is through the very long slog of calculating everything out!)

2 Independence of Two Events

Let's now discuss one more key property of events: *independence*. Here is the "formal definition":

Definition 1. *Two events $A, B \subseteq \Omega$ are* independent *if and only if*

$$\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A]\mathbf{Pr}[B]. \tag{1}$$

This is actually a slightly obscure way to define independence. Assuming that $\mathbf{Pr}[B] \neq 0$, we can write

$$\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A]\mathbf{Pr}[B] \qquad \Leftrightarrow \qquad \frac{\mathbf{Pr}[A \cap B]}{\mathbf{Pr}[B]} = \mathbf{Pr}[A] \qquad \Leftrightarrow \qquad \mathbf{Pr}[A \mid B] = \mathbf{Pr}[A].$$

This is how we think about it a bit more frequently: A and B are independent if and only if $\mathbf{Pr}[A \mid B] = \mathbf{Pr}[A]$; in other words, the probability of A doesn't depend at all on B's happening. Sometimes you'll hear this stated as "A is independent of B", but really it's a symmetric situation; this phrase is equivalent to "A and B are independent".

Similarly, assuming $\mathbf{Pr}[A] \neq 0$ it holds that A and B are independent events if and only if $\mathbf{Pr}[B \mid A] = \mathbf{Pr}[B]$. The reason the textbooks like to make the definition with equation (1) is so that the edge case of 0-probability events is included. Technically, if A is an event with $\mathbf{Pr}[A] = 0$, then A is independent of every event. But really, who spends a lot of time discussing events with probability 0?

2.1 A rant, and a new explanation of independence

Actually, we are of the opinion that there is something a bit bogus about the way textbooks make Definition 1, the definition of independent events. Normally textbooks make this definition and then ask you some exercises about it to "explain" the concept. For example, they often ask, "You flip 2 coins. Let A be the event that the first coin is heads. Let B be the event that both coins land the same way. Are A and B independent?" Well, it doesn't *sound* like they're independent: both events seem to depend on how the first flip turns out. But if you go and do the calculation you see that

$$\mathbf{Pr}[A \cap B] = \mathbf{Pr}[(\mathrm{H}, \mathrm{H})] = \frac{1}{4} = \frac{1}{2} \cdot \frac{1}{2} = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B],$$

and hence according to the definition they *are* independent.

The fact that textbooks make this one of the first few exercises about independence is very sneaky and a bit unfair because it's not true to life. In real life, no probabilist spends their time calculating $\mathbf{Pr}[A \cap B]$, $\mathbf{Pr}[A]$, and $\mathbf{Pr}[B]$, checking equation (1) and then exclaiming, "Aha! A and B are independent!" Because why would they? They know everything they want to know; who cares if some numbers in their calculations happen to be equal?

No, in real life, what happens is that you have two events A and B and you want to know $\mathbf{Pr}[A \cap B]$. You tell some kind of story in words, and then your story ends with the phrase, "... which means that A and B are independent." And then you say, "*Therefore*, we calculate that $\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A]\mathbf{Pr}[B]$".

Textbooks do this! And I'm sure you can see the circular logic. Or, rather, the invalid logic. How can you prove that A and B are "independent" without first verifying that $\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A]\mathbf{Pr}[B]$?

What's going on is that these textbooks are cheating. They're secretly using the following principle:

Principle: Suppose you identify two parts of an experiment,

$$\text{block 1:} \quad \left\{ \begin{array}{l} \cdots\cdots\cdots \\ \cdots\cdots\cdots \\ \cdots\cdots\cdots \end{array} \right.$$

$$\text{block 2:} \quad \left\{ \begin{array}{l} \cdots\cdots\cdots \\ \cdots\cdots\cdots \\ \cdots\cdots\cdots \end{array} \right.$$

and you mathematically *prove* that they cannot affect one another. (E.g., you could run them in the opposite order and it would be equivalent.) Let A be an event which you can prove depends only on block 1's execution, and let B be an event which you can prove depends only on block 2's execution. Then A and B are independent.

With some thought you should be able to convince yourself that the principle is correct; i.e., that if A and B satisfy the conditions then indeed $\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A]\mathbf{Pr}[B]$. (Think about rearranging the probability tree...)

2.2 Independence of Multiple Events

You might think we're done with defining independence by now, but we're not. We've only defined what it means for *two* events to be independent. Here's the full definition:

Definition 2. *Events* $A_1, \ldots, A_n \subseteq \Omega$ *are* independent *if and only if*

$$\mathbf{Pr}\left[\bigcap_{i \in S} A_i\right] = \prod_{i \in S} \mathbf{Pr}[A_i] \tag{2}$$

for every subset $S \subseteq \{1, 2, \ldots, n\}$.

Notice that this is a bit of a tricky definition. Here are some important remarks:

- Notice that it is *not* sufficient to check that $\mathbf{Pr}[A_1 \cap A_2 \cap \cdots A_n] = \mathbf{Pr}[A_1]\mathbf{Pr}[A_2] \cdots \mathbf{Pr}[A_n]$. I.e., you really need to check equation 2 for *all* subsets $S \subseteq \{1, 2, \ldots, n\}$, not just $S = \{1, 2, \ldots, n\}$.
- Further, it is *not* sufficient to check that A_i and A_j are independent for all pairs i, j with $i \neq j$. (This corresponds to testing all S with $|S| = 2$.) When all pairs of events are independent, the collection of all n events is called *pairwise independent* (naturally). But this does not mean the whole collection of events is "independent".
- *If* $A_1, \ldots, A_n$ are indeed independent, then we have

  $$\mathbf{Pr}[A_1 \mid \text{any event based on } A_2, A_3, \ldots, A_n] = \mathbf{Pr}[A_1].$$

  By "any event based on A_2, A_3, ..., A_n", we mean something like $(A_2 \cap A_3) \setminus A_4 \cup A_5$.

You will see all of these points discussed on the homework.

Just as before, the *real* way people employ independence of multiple events is via the **Principle** described above. The generalization is the obvious one: Suppose you have n blocks of randomized code and you can prove that none of them can affect any other — i.e., you could completely reorder them with no changes. Then if A_i is an event depending only on the ith block, the events A_1, ..., A_n are independent — and so you can *conclude* stuff like

$$\mathbf{Pr}[A_1 \mid (A_2 \cap A_3) \setminus A_4 \cup A_5] = \mathbf{Pr}[A_1].$$

One final comment on independence: In a way, we've seen it already. One of our axioms/assumptions about "randomized code", with calls to `RandInt` and `Bernoulli`, is that the executions of all lines in a "code path" are independent. E.g., any event depending on a call to `RandInt` is independent of any event depending on a different call to `RandInt`.

3 Applications in cryptography and number theory

Before moving on to the next part of probability theory (random variables), let's discuss some basic applications in cryptography and number theory. Although almost every area of computer science uses probability by choice, cryptography is one area where there is absolutely no option: probability is utterly essential for cryptography, and you literally can't do anything cryptographic without it. All cryptographic protocols use randomness, so any time you buy something online, any time you go to an `htpps://` web page, any time you use RSA or SSL, you are relying on probabilistic methods.

3.1 Factoring vs. primality testing

A typical cryptographic protocol looks something like this:

1. Pick two random k-bit prime numbers, p and q. (A typical value for k, the "security parameter", is 1024.)
2. Let $H = pq$. The number H is your "secret key".
3. (More steps...)

You have probably seen examples of protocols that start like this in 251; e.g., RSA. The security of such protocols depends on the empirical fact that we *do not know efficient algorithms for factoring integers.*

Fact 3. *The fastest known algorithm for factoring integers is the "Number Field Sieve" algorithm of John Pollard (with contributions from Buhler, H. Lenstra, Pomerance, and others). It factors k-bit numbers in time*[3] *approximately*

$$2^{1.9k^{1/3}(\log_2 k)^{2/3}}.$$

For $k = 1024$ this is about 2^{90}, which is pretty huge. I believe it currently takes something like several hundred CPU-years to factor a 1024-bit number on a typical desktop computer. The working assumption in cryptography is that malicious adversaries do not have this kind of time (and do not secretly know faster factoring algorithms). As you might know, there would be efficient factoring algorithms if we had large-scale "quantum computers"... But we digress.

The extreme difficulty of factoring might make you ask, "Wait a minute, how do we implement the protocol described above? It requires getting your hands on prime numbers. How do we know a number is prime is we can't efficiently factor it?"

Fact 4. *Strange but true: there are efficient algorithms for* testing *whether a number is prime.*

3.2 A *random* prime (and the Most Useful Approximation Ever)

Suppose, for a moment, that we have such an algorithm, call it `IsPrime`(m). Suppose further that it is a *deterministic* algorithm. We are still not done, because cryptographic protocols (such as the one described above) require you to choose a *random* k-bit prime (two of them, actually), not just find any old prime. (Having your secret key's primes be *random* is, of course, an essential defense against secrecy being broken.) There's only one solution: Pick a random number of at most k bits and hope/check it's prime!

Let's slightly simplify and assume we are looking for a random prime of *up to k digits*; i.e., between 0 and $2^k - 1$. (Really, this is just to make notation simpler.) In other words, if we define

$$P = \{m : 0 \le m < 2^k - 1 \text{ and } m \text{ is prime}\},$$

we want our algorithm to choose a random number p from P, each with equal probability $1/|P|$. How do we do *this*?

[3] Heuristically, actually; the algorithm's running time is not proven.

```
FindPrime(k):
    m ← RandInt(2^k) − 1
    if IsPrime(m) return m else return ‘‘fail’’.
```

For each $p \in P$, let A_p be the event that this code outputs p. Let S be the event that this code outputs succeeds; i.e., does *not* output "fail". We immediately have

$$\mathbf{Pr}[\text{outputs } p \mid \text{does not fail}] = \mathbf{Pr}[A_p \mid S] = \frac{\mathbf{Pr}[A_p \cap S]}{\mathbf{Pr}[S]}.$$

Each $p \in P$ is prime, so we have $A_p \cap S = A_p$. We also have $\mathbf{Pr}[A_p] = 1/2^k$ and $\mathbf{Pr}[S] = |P|/2^k$. Hence

$$\mathbf{Pr}[\text{outputs } p \mid \text{does not fail}] = \frac{1/2^k}{|P|/2^k} = 1/|P|.$$

Hence *conditioned on the algorithm succeeding*, the algorithm outputs each prime in P with equal probability, $1/|P|$.

Yes, but what is the probability of success, really? For this, we need to know $|P|/2^k$; equivalently, the number of primes less than 2^k. The famous Prime Number Theorem says that:

$$\frac{\#\text{ primes} \leq N}{N} \sim \frac{1}{\ln N} \qquad \text{as } N \to \infty.$$

Note that when $N = 2^k$ we have $\ln N = \Theta(1/k)$. Indeed, the following is pretty easy to prove (see the homework!):

Weak Prime Number Theorem:

$$\mathbf{Pr}[S] = \frac{|P|}{2^k} \geq \frac{1}{2k} \qquad \text{for all } k > 1.$$

At first, this doesn't look so great, at first: if $k = 1024$ then we may have as little as a $1/2048$ chance that the random number we pick is prime.[4] But as usual, we can boost our chances by repetition. Suppose we do:

```
for i ← 1...t
    if FindPrime(k) succeeds, break, returning the prime
```

If all t runs return "fail", we declare an overall failure. First, it's not hard to show (similar to Problem 2e on Homework 1) that still, each $p \in P$ is output with equal probability $1/|P|$ conditioned on overall success. On the other hand, if we draw the probability tree for the above code, with a branch for each call to `FindPrime`, we see that the event of overall failure consists of a single outcome, which has probability

$$\left(1 - \frac{|P|}{2^k}\right) \cdot \left(1 - \frac{|P|}{2^k}\right) \cdots \ (t \text{ times}) \ \cdots \cdot \left(1 - \frac{|P|}{2^k}\right).$$

Hence by the Weak Prime Number Theorem we have

$$\mathbf{Pr}[\text{overall failure}] \leq (1 - 1/2k)^t.$$

Is that small? To analyze it, we need to use the Most Useful Approximation.

[4] It would of course be a little smarter to pick a random *odd* number, which of course increases the success probability by a factor of 2.

Most Useful Approximation Ever: When x is a small real number near 0, possibly negative, we have

$$e^x \approx 1 + x.$$

(Recall that $e^x = 1 + x + x^2/2! + x^3/3! + \cdots$.) In particular, $e^x \geq 1 + x$ for all $x \in \mathbb{R}$.

To use this, we take x to be the small real number $-1/2k$, we have

$$\mathbf{Pr}[\text{overall failure}] \leq (1 - 1/2k)^t \leq (e^{-1/2k})^t = e^{-t/2k}.$$

Hence if we take t to be a large multiple of k we are in good shape; e.g., if $t = 400k$, we get

$$\mathbf{Pr}[\text{overall failure}] \leq e^{-(400k)/2k} = e^{-200}.$$

3.3 Testing primality

Next lecture we will return to the question of how to implement `IsPrime`(m).

15-359: Probability and Computing
Fall 2009

Lecture 4: Primality, Birthday Problem, Intro to Random Variables

1 Primality testing

Let's conclude our discussion from last lecture by returning to the problem of primality *testing*. We are given a number m, at most k bits long. How can we decide whether or not m is prime? Here is one well-known idea:

For $i = 2 \dots \sqrt{m}$,
    check if m is divisible by i.

You probably also know that this is a **bad** idea. The number of steps this little primality testing algorithm takes is at least something like

$$\sqrt{m} \approx \sqrt{2^k} = 2^{k/2}.$$

If, e.g., $k = 1024$ (a reasonable setting in the cryptographic protocols we discussed last lecture), then $2^{k/2} = 2^{512}$. Of course, this number of steps is utterly infeasible. So we need a better idea.

CMU's Professor Gary Miller had a better idea in 1975. He started by recalling the following simple theorem from number theory:

Fermat's Little Theorem: If m is prime and $1 \leq a < m$, then $a^{m-1} = 1 \bmod m$.

This suggests a way to prove that a number is *not prime* ("composite") without actually factoring it:

$$\text{if } 1 \leq a < m \text{ is such that } a^{m-1} \neq 1 \bmod m \text{ then } m \text{ is not prime.}$$

Now you might worry about whether we can check this efficiently when m is a k-bit number. But actually this is easy: first, compute

$$a \pmod m, \quad a^2 \pmod m, \quad a^4 \pmod m, \quad a^8 \pmod m, \dots, a^{2^k} \pmod m.$$

You can do this by starting with $a \pmod m$ and then repeatedly doing: square-and-reduce-mod-m. Since you are always working with k-bit numbers, doing one square-and-reduce-mod-m can be done in time $O(k^2)$ with the standard multiplication algorithm, or time $\tilde{O}(k)$ with a sophisticated multiplication algorithm.[1] You do k such square-and-reduces in total. Finally, having computed a raised to every power of 2, you can get a^{m-1} by multiplying together appropriate powers, based on the binary expansion of $m - 1$ (think about it). In short, you can indeed compute $a^{m-1} \pmod m$

[1] In 2007, Martin Fürer over at Penn State beat the 35-year-old Schönhage-Strassen algorithm for multiplying k-bit integers. Fürer's algorithm runs in time $k \log k 2^{O(\log^* k)}$, believe it or not!

efficiently, in time $\tilde{O}(k^2)$.

An obvious place to start is $a = 2$: if $2^{m-1} \neq 1 \pmod{m}$ then you can output "composite". But if $2^{m-1} = 1 \pmod{m}$, you can't be sure. So you could check if $3^{m-1} \neq 1 \pmod{m}$ — if so, output "composite", if not, perhaps prime. It doesn't make sense to go on forever, though:

Fact 1. *The converse of Fermat's Little Theorem is false. Robert Carmichael showed there exist non-primes m such that $a^{m-1} = 1$ mod m for all $a \neq 0 \pmod{m}$. These are called Carmichael numbers.*

1.1 The Miller-Rabin Test

Prof. Miller was not discouraged, however. He added a twist to the Fermat's Little Theorem test[2]:

Miller-Test(m, a)**:**
1. Compute $a^{m-1} \pmod{m}$; if $\neq 1$ then output ``composite''.
2. Check if any of the intermediate powers constructed in the previous computation are nontrivial square roots of 1 (i.e., satisfy $b^2 = 1 \pmod{m}$, yet $b \neq \pm 1 \pmod{m}$). If so, output ``composite''.

Definition 2. *If a number $1 \leq a < m$ causes Miller-Test(m, a) to output "composite", we say a is* a witness *for M.*

As this is a probability class, not a number theory class, let us content ourselves with stating the following facts:

1. If a is a "witness" then m is definitely composite.
2. If m is composite, then the number of a's in $\{1, 2, \ldots, m-1\}$ which are witnesses is at least $(3/4)(m-1)$.

In other words, if m is composite, there are plenty of witnesses to this fact. So how should we test if a given m is prime?

Since this is a probability class, you know the answer! Namely, pick a *random* a and test if it is a witness. Probability and computing were not so famously together back in 1975, so it actually took someone else, Michael Rabin, to point this out (he also proved Fact 2 above). The following is thus known as the Miller-Rabin test:

Algorithm Miller-Rabin(m)**:**
$a \leftarrow$ RandInt$(m-1)$.
if Miller-Test(m, a) determines a is a witness, output ``composite''
else output ``prime''

By Fact 2 above, we conclude the following:

Theorem 3. *Miller-Rabin is an efficient ($\tilde{O}(k^2)$ time) randomized algorithm for primality testing such that:*

[2]This was all in his Ph.D. thesis, by the way.

- *If m is prime, the algorithm* always *gives the correct answer.*
- *If m is composite, the probability the algorithm gives the wrong answer is $\leq \frac{1}{4}$.*

Of course, as we've seen several times now, if you don't like a failure probability of 1/4, you can repeat the algorithm 100 times and make the failure probability at most 4^{-100}.

1.2 Remarks on primality and factoring

There are a lot of fascinating algorithms and theorems surrounding primality and factoring.

1. For a long time, primality-testing and the Miller-Rabin algorithm gave the most famous example of a problem which could be solved efficiently with a probabilistic algorithm, but for which there was *no known* efficient deterministic algorithm. After decades of research (centuries, in some sense), this changed dramatically in 2002, when Manindra Agrawal, Neeraj Kayal, and Nitin Saxena gave a deterministic algorithm testing primality in polynomial time (now called the "AKS Algorithm"). That said, the fastest known variant of their algorithm, due to Pomerance and H. Lenstra, runs in time $\tilde{O}(k^6)$ for k-bit numbers, far slower than the Miller-Rabin algorithm. Miller-Rabin is still by the algorithm of choice for testing primality in practice.

2. Actually, although the converse of Fermat's Little Theorem is false, just checking whether $2^{m-1} = 1 \pmod{m}$ is an excellent primality test if m is a *randomly* chosen k-bit number. We know that Carmichael numbers (and others) will fool this test, but it turns out there are extremely few k-bit Carmichael numbers. Indeed, suppose you pick a random k-bit number and just check whether $2^{m-1} = 1 \pmod{m}$, outputting "prime" is this holds and "composite" otherwise. Of course, if m is prime your output will be correct. But Pomerance showed that if m is composite, the probability the algorithm wrongly outputs prime is roughly $2^{-k/\log k}$ — really small! So if you are only trying to find a random prime, just doing the "Fermat Test" with $a = 2$ will almost certainly give the right answer.

3. For certain non-cryptographic applications, you don't actually need a random prime. You just need *any* k-bit prime. Some people have wondered if there is an efficient deterministic algorithm for finding any old k-bit prime — I mean, we know how to test if a number is prime deterministically (AKS) and we know from the Prime Number Theorem that the fraction of k-bit numbers which are prime is pretty large: at least $1/2k$. But in fact, no one knows how to do this! The fastest known deterministic algorithm for finding a k-bit prime takes time roughly $2^{\sqrt{k}}$. But things may change soon — perhaps even this week! There is a large-scale, online, public mathematical collaboration working on this problem as we speak, with several famous mathematicians and computer scientists participating! Check it out at `http://polymathprojects.org`.

4. For some cryptographic problems, what you really want is a *random* number between 1 and 2^k, along with its factorization. Now, this looks totally infeasible — on one hand, how can you get a random number except by calling `RandInt`(2^k). And then once you have it, how can you get its factorization without running an extremely slow, $2^{k^{1/3}}$ or so, time algorithm? Shockingly, *there is an efficient algorithm for this problem*! You will prove it on Homework 2!

2 The Birthday Problem (or "Paradox", or "Attack")

You have probably seen the Birthday Problem before. Along with the Monty Hall Problem, it is the ultimate probability chestnut. Whereas Monty Hall depends on the precise wording of the problem (which is sometimes confusing) and is ultimately not especially interesting, the Birthday Problem actually comes up a lot and has important consequences. It is sometimes called the Birthday "Paradox" because it seems very surprising at first. And in cryptography, it's called the Birthday Attack, for reasons we'll see.

Question: There are m students in a room. What is the probability they all have different birthdays?

Modeling: We ignore Feb. 29 and possible seasonal variation in birthdays. More precisely, we model the birthdays like this:

```
for i ← 1...m,
    student[i].birthday ← RandInt(365)
```

For example, suppose $m = 40$. A common mistake is:

FALLACY: $\mathbf{Pr}[\text{all different birthdays}] = 1 - \mathbf{Pr}[\text{2 with same birthday}] = 1 - \frac{40}{365}$.

You can tell this is wrong almost by "type-checking": If $m = 366$ the answer should be 0 (by the Pigeonhole Principle), but the above fallacious argument would give a negative probability! Here is the correct solution:

Answer: Imagine going through the loop. For each $i = 1 \ldots m$, let A_i be the event that `student[`i`]`'s birthday differs from all the previously chosen birthdays. Let D be the event that *all* birthdays are different. Thinking carefully, we see that

$$D = A_1 \cap A_2 \cap A_3 \cap \cdots \cap A_m.$$

(Also note that $A_1 = \Omega$!) By the "Chain Rule",

$$\mathbf{Pr}[D] = \mathbf{Pr}[A_1] \cdot \mathbf{Pr}[A_2 \mid A_1] \cdot \mathbf{Pr}[A_3 \mid A_1 \cap A_2] \cdots \mathbf{Pr}[A_m \mid A_1 \cap A_2 \cap \cdots \cap A_{m-1}].$$

So what is, say, $\mathbf{Pr}[A_i \mid A_1 \cap \cdots \cap A_{i-1}]$? The event being conditioned on, $A_1 \cap \cdots \cap A_{i-1}$ means that the first $i-1$ students all had different birthdays. So out of the 365 possible birthdays, some $i-1$ *distinct* dates are occupied. *Given* that, the probability that the ith birthday chosen differs from all the previous ones, is

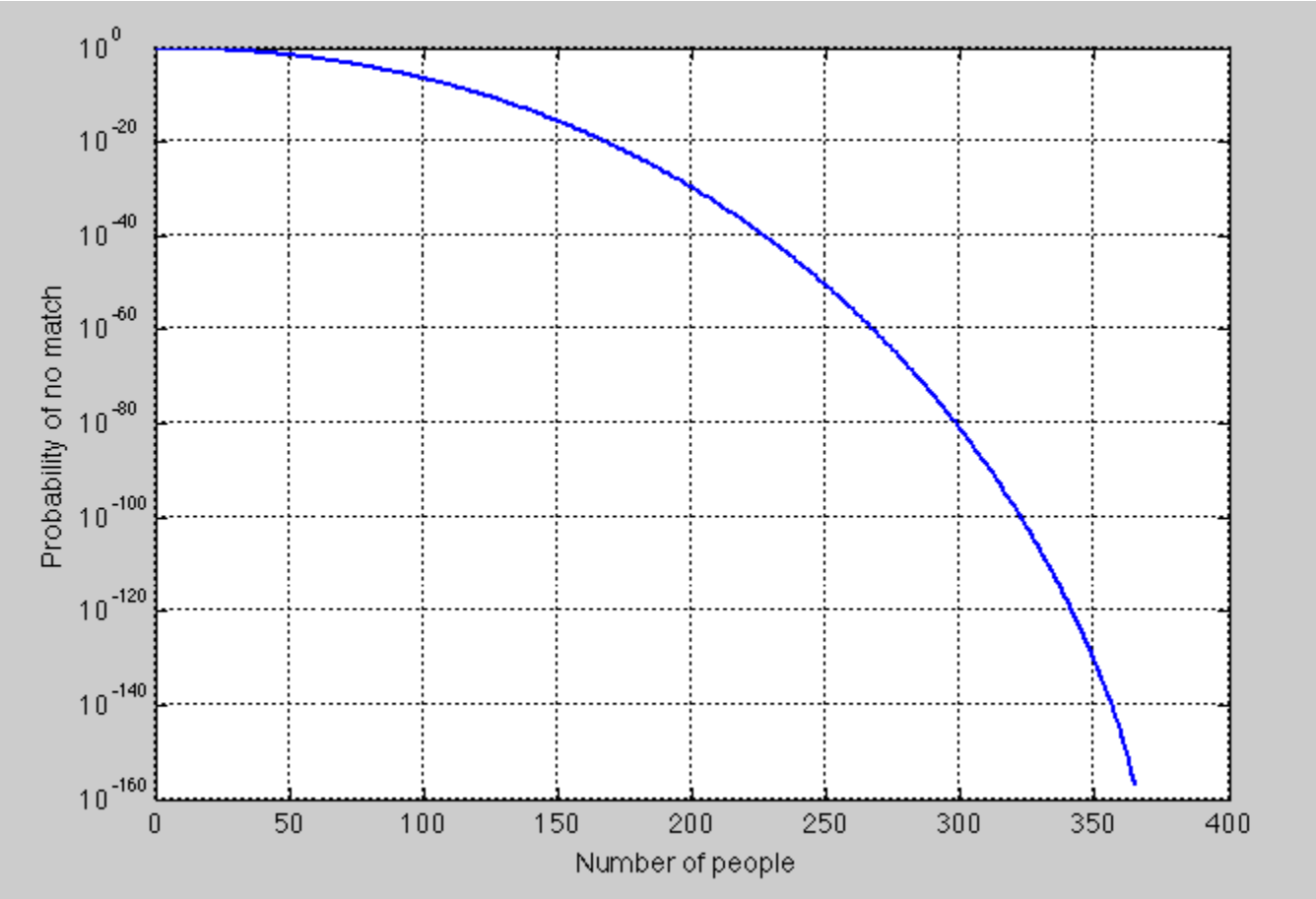
$$\mathbf{Pr}[A_i \mid A_1 \cap \cdots \cap A_{i-1}] = \frac{365 - (i-1)}{365},$$

assuming $m \leq 365$. Thus

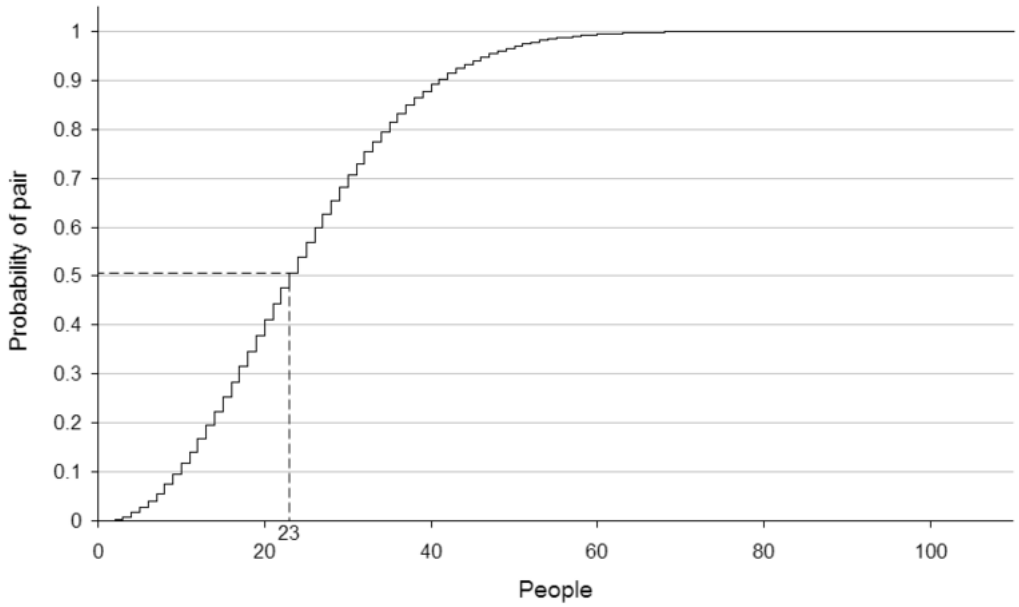
$$\mathbf{Pr}[D] = \frac{365-0}{365} \cdot \frac{365-1}{365} \cdot \frac{365-2}{365} \cdots \frac{365-(m-1)}{365}.$$

This is the final answer. □

Here is a plot of this probability as a function of m:



Here is a plot of "one minus this probability" — i.e., the probability of having at least one pair of matching birthdays:



Perhaps surprisingly, as you can see from the second plot, once $m \geq 23$, the probability of having a birthday match is greater than 50-50. So in a class our size, there should be a reasonable chance of a match. It is traditional in probability classes to try this out. It is also traditional for it to backfire and not work :)

2.1 Generalized Birthday Problem, and approximations

Instead of fixing the magic number 365, let's introduce the *Generalized Birthday Problem*. In the Generalized Birthday Problem, each of m students has a random number ("birthday") drawn from $\texttt{RandInt}(N)$. The Birthday Problem is the case $N = 365$. By the same analysis, the probability that all m numbers are distinct is

$$\begin{aligned} P_m &= \frac{N-0}{N} \cdot \frac{N-1}{N} \cdot \frac{N-2}{N} \cdots \frac{N-(m-1)}{N} \\ &= (1-1/N) \cdot (1-2/N) \cdots (1-(m-1)/N). \end{aligned}$$

This is hard to analyze, so we use the Most Useful Approximation Ever. Assuming $m \ll N$ we have that each i/N is "small", and hence $1 - i/N \approx e^{-i/N}$. I.e.,

$$P_m \approx e^{-1/N} e^{-2/N} \cdots e^{-(m-1)/N} = e^{-1/N-2/N-\cdots-(m-1)/N} = e^{-\frac{(m-1)m}{2N}},$$

where we used that the sum of 1 through $m-1$ is $\frac{(m-1)m}{2}$. The above gives a very nice closed-form approximation (which is accurate assuming $m \ll N$).

Given it, we can ask, what value of m is the 50-50 point? I.e., for what value of m do we have $P_m \approx 1/2$? Well, that's

$$1/2 = e^{-\frac{(m-1)m}{2N}} \quad \Rightarrow \quad \ln 2 = \frac{(m-1)m}{2N} \quad \Rightarrow \quad (2\ln 2)N = (m-1)m \approx m^2$$
$$\Rightarrow \quad m \approx \sqrt{2\ln 2}\sqrt{N} = 1.18\sqrt{N}.$$

I.e., the break-even point is asymptotic to $\sqrt{N}$. The approximations we made, by the way, are extremely accurate. Even for the relatively small $N = 365$ the approximation gives a break-even of $m \approx 22.5$, whereas the true 50-50 point is $m \approx 23$.

2.2 The Birthday Attack

In Cryptography, a "Cryptographic Hash Function" is a map which "scrambles" long strings into k-bit "hashes". A good Cryptographic Hash Function f has two properties:

- Given the hash $f(M)$ of a message string M, it's computationally infeasible to recover M.
- It's computationally infeasible to find a "collision", meaning a pair of distinct messages $M_1 \neq M_2$ such that $f(M_1) = f(M_2)$.

Cryptographic Hash Functions are used all the time, in authentication schemes, data integrity schemes, digital signatures, e-cash, etc. Ron Rivest (of RSA) proposed the first popular Cryptographic Hash Function, called MD5. Later, the NSA proposed one called SHA (also known as SHA-0). Later, the NSA mysteriously decided that SHA was unsafe, and promoted a successor called SHA-1. SHA-1 is used now quite widely, in SSL, PGP, and other protocols; it has a k value

of 160.

Suppose we try to "break" a hash function by finding a collision. (If we can do this, it's possible to, e.g., trick people into digitally signing stuff.) One way to do this is to just take a huge number of messages M, hash them all, and hope to find two with the same hash value. If a hash function f is really safe, it basically means that $f(M)$ acts like a random number between 1 and 2^k. Assuming this, how many messages would you have to try before there was at least a 50% chance of finding two with the same hash? This is precisely the Generalized Birthday Problem, with $N = 2^k$. Thus our above analysis says that we'd need to try about $\sqrt{2^k} = 2^{k/2}$ many messages. For SHA-1, this is 2^{80} messages.

In general, any Cryptographic Hash Function designer should be aware of the "Birthday Attack" — i.e., trying to find collisions by testing many random messages. A Cryptographic Hash Function is typically thought of as "broken" in the crypto community if there is a way of finding collisions much faster than by the Birthday Attack.

And indeed, SHA-1 is now considered broken! In February 2005, Xiaoyun Wang (and two students) gave a method for finding SHA-1 collisions with about 2^{69} tests. Note that this is about 2000 times faster than the Birthday Attack. Later, Wang (and two more coauthors) got it down to 2^{63}; i.e., about $130,000$ times faster than the Birthday Attack! Consequently, "SHA-2", and indeed "SHA-3" is on the way...

Xiaoyun Wang

3 Random Variables

That's enough cryptography applications for now. We'll now go back to our regularly scheduled programming: basic probability theory. We move on from events to *random variables*. Random variables have a slightly confusing nature, but are a very important concept. In fact, in the end we'll talk about them a lot more than we'll ever talk about events.

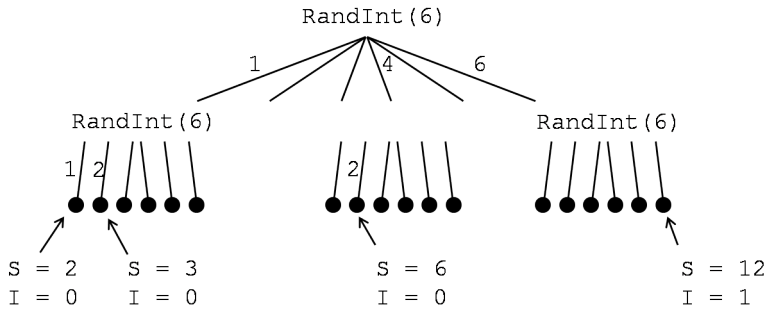
Sort of the definition: Suppose you have an experiment (block of randomized code). Suppose X is one of the variables used and suppose that its data type is "real". If we consider X's value at the end of the execution, we call that a *random variable*.

(A tiny point: X should always be *defined*, so please make sure you initialize all your variables :)

Here is a "rolling two dice" kind of example:

$S \leftarrow$ `RandInt`(6) + `RandInt`(6)
`if` $S = 12$,
    $I \leftarrow 1$
`else`
    $I \leftarrow 0$

As usual we draw the probability tree. Let's also write down the variables' values at the outcomes. (Only partial details are shown in the below diagram.)



Here there are two random variables, S and I.

Math textbooks don't normally define random variables this way. Actually, they don't define sample spaces and probabilities like we do either. For those, they often just tell you Ω and tell you the probabilities (summing to 1) of each outcome — they don't actually give a random experiment generating these outcomes and probabilities. There's a similar way to introduce random variables: just associate a real number to each outcome.

Official math definition: A *random variable* X is a *function* $\Omega \to \mathbb{R}$ from outcomes to real numbers.

In our example, we have

$$S(\ (1,1)\ ) = 2, \quad S(\ (1,2)\ ) = 3, \quad \cdots \quad S(\ (6,6)\ ) = 12,$$

$$I(\ (1,1)\ ) = 0, \quad I(\ (1,2)\ ) = 0, \quad \cdots \quad I(\ (6,6)\ ) = 1.$$

By the way, it is extremely standard practice for random variables to be denoted by capital letters. So far this doesn't look so great because we also denote *events* by capital letters, but there you have it. Most frequently, capital letters will mean random variables — especially letters like X, Y, Z.

15-359: Probability and Computing
Fall 2009

Lecture 5: Random variables, indicators, expectation and linearity thereof, PMFs

1 Random Variables

Today's lecture will be all about random variables.

1.1 Recall from last lecture

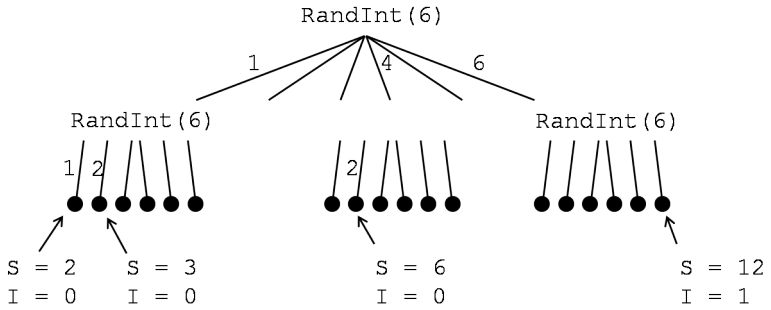
Remember from last time, we essentially had two definitions of what a random variable is:

Sort of the definition: Suppose you have an experiment (block of randomized code). Suppose X is one of the variables used and suppose that its data type is "real". If we consider X's value at the end of the execution, we call that a *random variable*.

Official math definition: A *random variable* X is a *function* $\Omega \to \mathbb{R}$ from outcomes to real numbers.

Let's also recall the "rolling two dice" scenario we were discussing last time:
$S \leftarrow$ `RandInt`(6) + `RandInt`(6)
`if` $S = 12$,
    $I \leftarrow 1$
`else`
    $I \leftarrow 0$

There are two random variables (so far), S and I. Here is the associated probability tree, with the variables' values at the outcomes. (Only partial details are shown in the below diagram.)



In the official way of looking at things, with random variables as functions from the sample space to the reals, we have

$$S(\ (1,1)\ ) = 2, \quad S(\ (1,2)\ ) = 3, \quad \cdots \quad S(\ (6,6)\ ) = 12,$$
$$I(\ (1,1)\ ) = 0, \quad I(\ (1,2)\ ) = 0, \quad \cdots \quad I(\ (6,6)\ ) = 1.$$

1.2 Introducing Random Variables (r.v.'s)

We are already getting a little familiar with random variables, as you can tell from the fact that we have started casually abbreviating them as "r.v.'s". One thing about random variables that takes some getting used to is that there are several ways to "introduce" them when you're solving a probability problem. E.g.,

1. **Retroactively:** For example, having shown the above "rolling two dice" experiment, one might say, "Let D be the random variable given by subtracting the first roll from the second roll." In terms of r.v.'s as functions from the sample space into the reals, this of course means

   $$D(\ (1,1)\ ) = 0, \quad \ldots, \quad D(\ (5,3)\ ) = -2,$$

   etc.

2. **In terms of other r.v.'s:** Again, after the example we may say something like, "Let $Y = S^2 + D$." Then Y is a random variable, and, e.g., $Y(\ (2,3)\ ) = 26$ (i.e., on outcome $(2,3)$, the r.v. Y takes the value 26). Sometimes, we don't even bother to give a random variable like this its own name! For example, we may just say, "The random variable $S^3 + 2D - 4\ldots$".

   Let's do a further example: Suppose you win \$30 if the roll is double-sixes, and you lose \$1 otherwise. Let W be the r.v. representing your winnings, in dollars. Then, to relate W to r.v.'s already defined, we have

   $$W = 31 \cdot I - 1.$$

3. **Without bothering to introduce an experiment:** Finally, sometimes we take things so far as to introduce a random variable without even explicitly giving an experiment (code) which generates it. For example, it is very common to say something like, "Let T be a random variable which is uniformly distributed on[1] $\{1, 2, \ldots, 100\}$." Such an English-language sentence should be translated to:

   $$T \leftarrow \texttt{RandInt}(100).$$

   Similarly, the English-language sentence "Let X be a Bernoulli random variable with parameter p." should be translated to

   $$X \leftarrow \texttt{Bernoulli}(p).$$

   Note: People often call the parameter p the "success probability". It's a weird term, we agree, but you'll need to get used to it.

4. **By PMF:** We'll see what this means later in the lecture.

1.3 From Random Variables to Events

You will often use r.v.'s to define events. For example, in our rolling-two-dice experiment, we might say something like, "Let A be the event that $S \geq 10$." This means

$$A = \{(4,6), (5,5), (5,6), (6,4), (6,5), (6,6)\},$$

which in turn implies

$$\mathbf{Pr}[S \geq 10] = 6/36 = 1/6.$$

[1] "Uniform on" or "uniformly distributed on" means that each outcome is to have equal probability.

Note that in this second statement, the expression "$S \geq 10$" denotes an *event*. It is shorthand for the event

$$\{\ell : S(\ell) \geq 10\}.$$

Similarly (recalling the r.v. D), we have

$$\mathbf{Pr}[D = 1] = 5/36,$$

because $D = 1$ is shorthand for the *event* equal to $\{(1,2),(2,3),(3,4),(4,5),(5,6)\}$.

1.4 From Events to Random Variables: Indicator Random Variables

There is a very frequently used way of defining random variables from events, too:

Definition 1. *Let A be an event. The* indicator *of A is the random variable X which is 1 if A occurs and is 0 if A does not occur. More formally, if Ω is the sample space in which A lives, we have the definition*

$$X : \Omega \to \mathbb{R}, \qquad X(\ell) = \begin{cases} 1 & \text{if } \ell \in A, \\ 0 & \text{if } \ell \notin A. \end{cases}$$

Please don't forget about indicator random variables; it seems like a somewhat useless little definition, but it will come up a lot.

2 Expectation / Expected Value

The most common thing you will ever do with a random variable is calculate its "expected value"; i.e., its "average value". This is, by the way, the notion with the most aliases in all of probability!

Definition 2. *The* expected value *AKA* expectation *AKA* mean *(occasionally AKA* average*) of the random variable $X : \Omega \to \mathbb{R}$ is defined by*

$$\mathbf{E}[X] = \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] \cdot X(\ell).$$

(Sometimes our sample spaces Ω will be countably infinite. In that case, there are potential caveats about infinite sums. We will not worry about them for now.)

Intuitively, the expectation (AKA mean, AKA expected value...) is what you feel the average value of X would be if you ran your experiment millions and millions of times. Let's do some examples:

- Let $R \leftarrow \texttt{RandInt}(6)$. Then

$$\mathbf{E}[R] = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 4 + \frac{1}{6} \cdot 5 + \frac{1}{6} \cdot 6 = \frac{21}{6} = 3.5.$$

  Note here that the expected value, 3.5, is not actually a value that the random variable R ever takes!

- Our "gambling winnings" example when we roll two dice:

$$W(\ell) = \begin{cases} +30 & \text{if } \ell = (6,6), \\ -1 & \text{else.} \end{cases}$$

Then

$$\begin{aligned} \mathbf{E}[W] &= \frac{1}{36} \cdot (-1) + \frac{1}{36} \cdot (-1) + \cdots + \frac{1}{36} \cdot (-1) + \frac{1}{36} \cdot (30) \\ &= -\frac{35}{36} + \frac{30}{36} \\ &= -\frac{5}{36} \\ &= -13.9 \text{ cents.} \end{aligned}$$

- Let $S \leftarrow \texttt{RandInt}(6) + \texttt{RandInt}(6)$. Then

$$\mathbf{E}[S] = \frac{1}{36} \cdot 2 + \frac{1}{36} \cdot 3 + \frac{1}{36} \cdot 4 + \cdots + \frac{1}{36} \cdot 12 = \cdots = 7.$$

Here the terms corresponding to outcomes $(1,1)$, $(1,2)$, $(1,3)$, and $(6,6)$ are shown above, and we've omitted some laborious arithmetic which leads to the answer 7.

2.1 Linearity of Expectation

The calculations involved in that last example, the expected value of the sum of two dice, are a bit laborious. How can we simplify them?

The following is the **#1 trick** you will ever use in probability. It seems so obvious at first, but weirdly enough, it'll seem less obvious the more you use it! In any case, it's the best probability trick ever. It is called...

Linearity of Expectation: Let X and Y be any random variables at all (defined over the same sample space Ω). Then

$$\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y].$$

Proof. The proof is a most straightforward calculation. Let $Z = X + Y$ (a random variable, remember). Then

$$\begin{aligned} \mathbf{E}[Z] &= \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] \cdot Z(\ell) \\ &= \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] \cdot (X(\ell) + Y(\ell)) \\ &= \sum_{\ell \in \Omega} (\mathbf{Pr}[\ell] \cdot X(\ell) + \mathbf{Pr}[\ell] \cdot Y(\ell)) \\ &= \sum_{\ell \in \Omega} (\mathbf{Pr}[\ell] \cdot X(\ell)) + \sum_{\ell \in \Omega} (\mathbf{Pr}[\ell] \cdot Y(\ell)) \\ &= \mathbf{E}[X] + \mathbf{E}[Y]. \end{aligned}$$

The first, second, and last lines are "by definition". The other two lines are just arithmetic. □

Also: Also falling under the "Linearity of Expectation" is umbrella is the fact that

$$\mathbf{E}[cX] = c\mathbf{E}[X]$$

for any constant $c \in \mathbb{R}$ and any random variable X. The proof is similar and easier.

Let's do some examples to try out this wonderful Linearity of Expectation trick. We'll start with the rolling two dice example. Let R_1 be the first die's value (a random variable) and let R_2 be the second die's value. As we've seen before, we have $\mathbf{E}[R_1] = 3.5$ and $\mathbf{E}[R_2] = 3.5$. Let $S = R_1 + R_2$, the random variable giving the sum of the two dice. Then by Linearity of Expectation,

$$\mathbf{E}[S] = \mathbf{E}[R_1 + R_2] = \mathbf{E}[R_1] + \mathbf{E}[R_2] = 3.5 + 3.5 = 7,$$

just as calculated before.

Linearity of Expectation also implies that for any random variables $X_1, \ldots, X_n$,

$$\mathbf{E}[X_1 + X_2 + \cdots + X_n] = \mathbf{E}[X_1] + \mathbf{E}[X_2] + \cdots + \mathbf{E}[X_n].$$

You can easily prove that yourself by induction, from the case of adding two random variables. As a corollary, we get, e.g., that

$$\mathbf{E}[\text{sum of 100 dice}] = 350.$$

You'd *never* make it to this result if you did, "With probability $1/6^{100}$, the outcome is $(1, 1, \ldots, 1)$, and the sum is 100. With probability $1/6^{100}$ the outcome is…"!

2.2 Another trick: the expectation of an indicator random variable

Let's quickly look at another simple trick, which plays extremely nicely with our favorite trick, linearity of expectation.

Trick 2: Let A be an event, and let I be the indicator random variable for A. Then

$$\mathbf{E}[I] = \mathbf{Pr}[A].$$

Proof. The proof is very simple:

$$\mathbf{E}[I] = \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] \cdot I(\ell) = \sum_{\ell \in A} \mathbf{Pr}[\ell] \cdot 1 + \sum_{\ell \notin A} \mathbf{Pr}[\ell] \cdot 0 = \sum_{\ell \in A} \mathbf{Pr}[\ell] = \mathbf{Pr}[A].$$

□

Simple, but useful; please remember it.

2.3 A formula for expectation

Here is a formula for the expected value of a random variable that you might have seen before. In fact, you might have seen it as the *definition* of the expectation. However, we prefer to make the definition we made, and treat this as a formula:

Expectation formula:

$$\mathbf{E}[X] = \sum_{u \in \text{range}(X)} \mathbf{Pr}[X = u] \cdot u.$$

Note here that the u's are real numbers, the expression range(X) means the set of real numbers that X might take on, and the expression $X = u$ is an *event*.

Proof. The proof can be thought of as following the "counting two ways" technique. The idea is to *group together all the outcomes in which X takes the same value.*

$$\begin{aligned}
\mathbf{E}[X] &= \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] \cdot X(\ell) \\
&= \sum_{u \in \text{range}(X)} \sum_{\substack{\ell \text{ such that} \\ X(\ell)=u}} \mathbf{Pr}[\ell] \cdot X(\ell) \qquad \text{(here is the grouping step)} \\
&= \sum_{u \in \text{range}(X)} \sum_{\substack{\ell \text{ such that} \\ X(\ell)=u}} \mathbf{Pr}[\ell] \cdot u \\
&= \sum_{u \in \text{range}(X)} u \cdot \sum_{\substack{\ell \text{ such that} \\ X(\ell)=u}} \mathbf{Pr}[\ell] \\
&= \sum_{u \in \text{range}(X)} u \cdot \mathbf{Pr}[X = u].
\end{aligned}$$

In the second-to-last line here we pulled the u out of the inner summation, since this summation does not depend on u. In the last line, we simply used the definition of $\mathbf{Pr}[X = u] = \mathbf{Pr}[\{\ell : X(\ell) = u\}]$. □

Let's see how this formula can speed up computations by doing an example. Remember the random variable W, our winnings in dollars in the game where we get \$30 for double-sixes and lose \$1 otherwise? Here range$(W) = \{30, -1\}$, so we can now compute as follows:

$$\mathbf{E}[W] = \mathbf{Pr}[W = +30] \cdot 30 + \mathbf{Pr}[W = -1] \cdot (-1) = \frac{1}{36} \cdot 30 + \frac{35}{36} \cdot (-1) = -\frac{5}{36},$$

just as before.

Here's yet another way to compute $\mathbf{E}[W]$, using an indicator random variable. Remember we noticed that $W = 31 \cdot I - 1$, where I is the indicator random variable for the event of double-sixes, i.e., $\{(6, 6)\}$. You can think of this as

$$W = 31 \cdot I + L,$$

where L is the random variable which is *always* -1. This might seem like a weird random variable, but it's perfectly valid. Again, it's the function which associates to each of the outcomes the

value -1. Now

$$\begin{aligned}
\mathbf{E}[W] &= \mathbf{E}[31 \cdot I + L] \\
&= 31 \cdot \mathbf{E}[I] + \mathbf{E}[L] \qquad \text{(by linearity of expectation)} \\
&= 31 \cdot \mathbf{Pr}[\text{double-sixes}] + (-1) \qquad \text{(since the expectation of an indicator is the probability of the associated event)} \\
&= 31 \cdot \frac{1}{36} - 1 \\
&= -\frac{5}{36}.
\end{aligned}$$

As you become more comfortable with random variables, you will stop introducing a special name like L for the random variable which is always -1; you'll just refer to this random variable as -1. And you'll just write,

$$\begin{aligned}
\mathbf{E}[W] &= \mathbf{E}[31 \cdot I - 1] \\
&= 31 \cdot \mathbf{E}[I] - 1 \\
&\text{etc.}
\end{aligned}$$

3 Probability Mass Functions (PMFs)

To introduce the concept of probability mass functions, let's do another practice problem on computing expectations:

Question: Suppose X is a uniformly random integer between 1 and 10 (i.e., $X \leftarrow \texttt{RandInt}(10)$). What is $\mathbf{E}[X \bmod 3]$?

Answer: The sample space here is $\Omega = \{1, 2, \ldots, 10\}$. Let Y be the random variable X mod 3. Then Y's range is $\{0, 1, 2\}$, and

$$\begin{aligned}
\mathbf{E}[Y] &= 0 \cdot \mathbf{Pr}[Y = 0] + 1 \cdot \mathbf{Pr}[Y = 1] + 2 \cdot \mathbf{Pr}[Y = 2] \\
&= \mathbf{Pr}[Y = 1] + 2 \cdot \mathbf{Pr}[Y = 2] \\
&= \mathbf{Pr}[\{1, 4, 7, 10\}] + 2 \cdot \mathbf{Pr}[\{2, 5, 8\}] \\
&= 4/10 + 2 \cdot (3/10) \quad = \quad 1.
\end{aligned}$$

As you can see from this example, in computing $\mathbf{E}[Y]$ we didn't care too much about *how* Y was generated; really, we just needed to know $\mathbf{Pr}[Y = u]$ for each number u. Similarly, looking at the expected value formula,

$$\mathbf{E}[X] = \sum_{u \in \text{range}(X)} \mathbf{Pr}[X = u] \cdot u,$$

you see that given an r.v. X it's quite handy to know the value of $\mathbf{Pr}[X = u]$ for each real number u that X might take on. Let's give a definition for that:

Definition 3. *Given a random variable X, its* probability mass function *(PMF) is the function $p_X : \mathbb{R} \to \mathbb{R}$ defined by*

$$p_X(u) = \mathbf{Pr}[X = u].$$

Note that a PMF is 0 on most inputs u; it's only nonzero on the numbers which are in X's range.[2]

Here are some easy-to-check facts about PMFs:

- $\sum_{u \in \text{range}(X)} p_X(u) = 1$ always, since the events $\{X = u\}$ partition Ω.
- $\mathbf{Pr}[X \in S] = \sum_{u \in S} p_X(u)$ for any set $S \subseteq \mathbb{R}$.
- Our formula for expected value can be rewritten as

$$\mathbf{E}[X] = \sum_{u \in \text{range}(X)} u \cdot p_X(u).$$

In fact, practically everything you want to know about a random variable X can be determined from its PMF. You rarely need to know how X was generated, or what Ω is. Only if you're literally interested in the question, "how did we get X?" do you need to know anything beyond X's PMF. Because of this, *very* often a random variable is just specified by its PMF.

For example, we might say something like, "Let X be the random variable with $p_X(1) = 1/6$, $p_X(2) = 1/3$, and $p_X(3) = 1/2$." You should try translating this into a simple piece of randomized code which sets X as desired. To be a legal PMF, we must have the following three conditions:

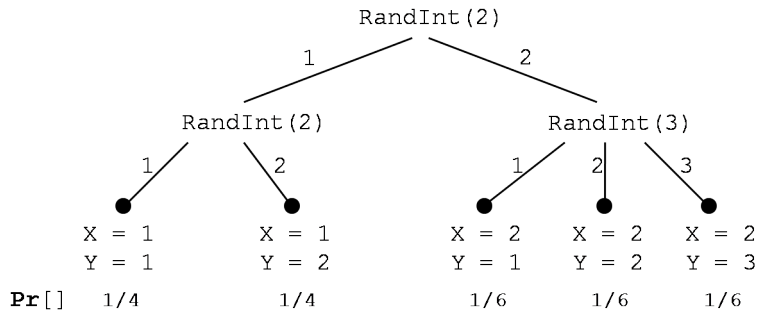
- $p_X(u) \geq 0$ for all $u \in \mathbb{R}$,
- $p_X(u) \neq 0$ for at most countably many $u \in \mathbb{R}$,
- $\sum_u p_X(u) = 1$.

3.1 Joint PMFs

Let's now talk about issues that come up when dealing with more than one random variable. We'll begin with an example experiment:

$X \leftarrow \texttt{RandInt}(2)$
$Y \leftarrow \texttt{RandInt}(1 + X)$

Here is the associated probability tree, with the values of the two random variables X and Y given at each outcome:



[2] and there are at most countably infinite many such inputs

Let's compute the PMFs of X and Y:

$$p_X(1) = 1/2, \qquad p_X(2) = 1/2$$

$$p_Y(1) = 1/4 + 1/6 = 5/12, \qquad p_Y(2) = 1/4 + 1/6 + 5/12, \qquad p_Y(3) = 1/6$$

Great! Now that we know the PMF of X, computing its expectation is no problem:

$$\mathbf{E}[X] = (1/2) \cdot 1 + (1/2) \cdot 2 = 3/2.$$

Similarly, now that we know the PMF of Y, computing its expectation is no problem:

$$\mathbf{E}[Y] = (5/12) \cdot 1 + (5/12) \cdot 2 + (1/6) \cdot 3 = 7/4.$$

What about $\mathbf{E}[XY]$? Easy: it's $\mathbf{E}[X]\mathbf{E}[Y] = 21/8$, right? No!

FALLACY: $\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y]$.

What is the actual value of $\mathbf{E}[XY]$? Well, let's go right back to the definition:

$$\mathbf{E}[XY] = \sum_{\ell \in \Omega} \mathbf{Pr}[\ell] X(\ell) Y(\ell) = (1/4) \cdot 1 \cdot 1 + (1/4) \cdot 1 \cdot 2 + (1/6) \cdot 2 \cdot 1 + (1/6) \cdot 2 \cdot 2 + (1/6) \cdot 2 \cdot 3 = 11/4.$$

As you notice, $11/4 \neq 21/8$, demonstrating the fallaciousness of the fallacy. What we see here is that to understand the expectations of expressions involving two random variables X and Y, it's *not* enough in general to know the PMF of X and the PMF of Y. You need to know the probabilities of each *pair* of values they may take. This leads us to the following definition, a generalization of PMFs:

Definition 4. *The* joint PMF *of two random variables X and Y is the function*

$$p_{XY} : \mathbb{R}^2 \to \mathbb{R}$$

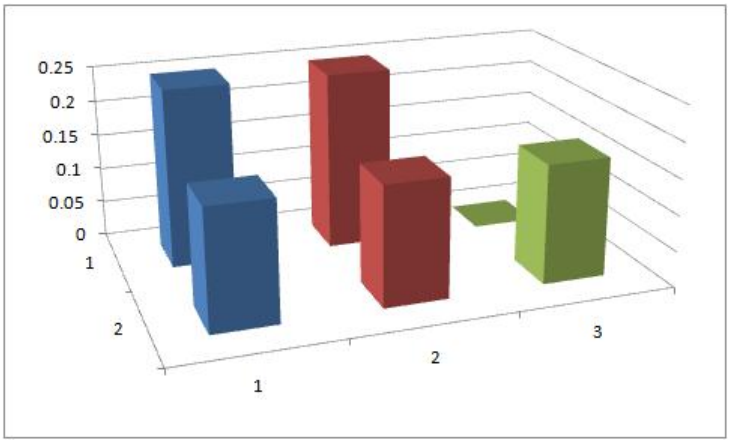
defined by

$$p_{XY}(u, v) = \mathbf{Pr}[X = u \text{ and } Y = v].$$

I.e., for each *pair* of real numbers (u, v) that (X, Y) can take on, the joint PMF tells us the probability of this occurring. It's good to think of joint PMFs in a table,

X \ Y	1	2	3
1	1/4	1/4	0
2	1/6	1/6	1/6

with the possible X values as rows, the possible Y values as columns, and the probabilities of (X, Y) pairs in the cells. For example, the above table for p_{XY} tells us that $p_{XY}(2, 3) = 1/$, $p_{XY}(1, 3) = 0$, etc. Or, one can think of showing the PMF as a bar chart,



Given a joint PMF of X and Y, it's easy to get the (single) PMF of each of them: to get the PMF of X sum the rows; to get the PMF of Y sum the columns. From the above table we see that this indeed gives

$$p_X(1) = 1/4 + 1/4 = 1/2, \qquad p_X(2) = 1/6 + 1/6 + 1/6 = 1/2,$$

and

$$p_Y(1) = 1/4 + 1/6 = 5/12, \qquad p_Y(2) = 1/4 + 1/6 = 5/12, \qquad p_Y(3) = 0 + 1/6 = 1/6,$$

just like we calculated before.

More importantly, with the joint PMF of X and Y, you can calculate more easily the expectation of any "function of X and Y"; here's the formula, just like the formula for the expectation of one random variable:

Formula: Given any function of X and Y, say $f(X,Y)$, we have

$$\mathbf{E}[f(X,Y)] = \sum_{u,v} p_{XY}(u,v) \cdot f(u,v).$$

In the case $f(X,Y) = XY$, you can see how this formula gives us $\mathbf{E}[XY] = 11/4$ as in our previous calculation. It's important that you get a lot of practice in working with joint PMFs, so let's do another example.

Question: What is $\mathbf{E}[\frac{Y}{X}]$?

Answer: In light of the fallacy, you can be sure it's not $\frac{\mathbf{E}[Y]}{\mathbf{E}[X]}$! Instead, using the formula and then the table for the joint PMF p_{XY} of X and Y, we have:

$$\begin{aligned}
\mathbf{E}\left[\frac{Y}{X}\right] &= p_{XY}(1,1) \cdot \frac{1}{1} + p_{XY}(1,2) \cdot \frac{2}{1} + p_{XY}(1,3) \cdot \frac{3}{1} + p_{XY}(2,1) \cdot \frac{1}{2} + p_{XY}(2,2) \cdot \frac{2}{2} + p_{XY}(2,3) \cdot \frac{3}{2} \\
&= (1/4) \cdot 1 + (1/4) \cdot 2 + 0 \cdot 3 + (1/6) \cdot \frac{1}{2} + (1/6) \cdot 1 + (1/6) \cdot \frac{3}{2} \\
&= \frac{5}{4}.
\end{aligned}$$

Finally, of course, all of this joint PMF theory generalizes to the case of 3 or more random variables. For example, given three random variables X, Y, and Z, they have a joint PMF

$$p_{XYZ}(u, v, w) = \mathbf{Pr}[X = u \text{ and } Y = v \text{ and } Z = w];$$

and, if you are interested in $\mathbf{E}[g(X, Y, Z)]$ for some function g of the three random variables, the formula is

$$\mathbf{E}[g(X, Y, Z)] = \sum_{u,v,w} p_{XYZ}(u, v, w) \cdot g(u, v, w).$$

15-359: Probability and Computing
Fall 2009

Lecture 6: Independent r.v.'s, conditional expectation, Binomial and Geometric r.v.'s, Linearity of Expectation + Indicators method, Max-Cut

In this lecture we will begin by reviewing the PMFs (or "distributions") of random variables and do some more theory of multiple random variables. Then we'll see your two favorite kinds of random variables, plus one of your favorite techniques for solving expectation problems...

1 Multiple random variables

1.1 Joint PMFs

We ended the last lecture discussing *joint PMFs* of random variables, and how to compute expectations of quantities depending on several random variables. Let's recall the definition:

Definition 1. *The* joint PMF *of random variables $X_1, \ldots, X_n$ is the function*

$$p_{X_1 X_2 \cdots X_n} : \mathbb{R}^n \to \mathbb{R}$$

defined by

$$p_{X_1 X_2 \cdots X_n}(u_1, \ldots, u_n) = \mathbf{Pr}[X_1 = u_1 \text{ and } X_2 = u_2 \text{ and } \cdots \text{ and } X_n = u_n].$$

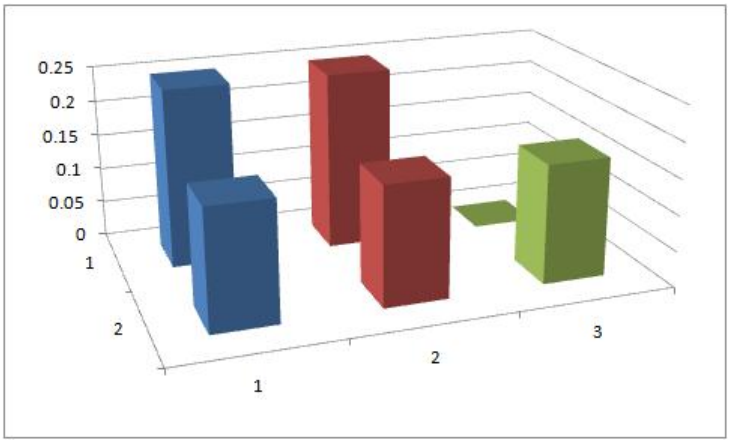
E.g., last time we discussed the case of two random variables, X and Y, generated as follows:

$X \leftarrow \texttt{RandInt}(2)$
$Y \leftarrow \texttt{RandInt}(1 + X)$

When you have just two random variables, it's always a good idea to think of their joint PMF in a table — in this case,

X \ Y	1	2	3
1	1/4	1/4	0
2	1/6	1/6	1/6

with the possible X values as rows, the possible Y values as columns, and the probabilities of (X, Y) pairs in the cells. For example, the above table for p_{XY} tells us that $p_{XY}(2, 3) = 1/$, $p_{XY}(1, 3) = 0$, etc. Or, one can think of showing the PMF as a bar chart,



Given a joint PMF of X and Y, it's easy to get the (single) PMF of each of them: to get the PMF of X sum the rows; to get the PMF of Y sum the columns. From the above table we see that this indeed gives

$$p_X(1) = 1/4 + 1/4 = 1/2, \qquad p_X(2) = 1/6 + 1/6 + 1/6 = 1/2,$$

and

$$p_Y(1) = 1/4 + 1/6 = 5/12, \qquad p_Y(2) = 1/4 + 1/6 = 5/12, \qquad p_Y(3) = 0 + 1/6 = 1/6,$$

as we calculated in the last lecture.

With the joint PMF of random variables, you can calculate the expectation of any function of them. For example, suppose you have two random variables X and Y. Then:

Formula: Given any function of X and Y say $f(X, Y)$, we have

$$\mathbf{E}[f(X,Y)] = \sum_{u,v} p_{XY}(u,v) \cdot f(u,v).$$

The analogous formula of course holds when you have 3 random variables, or n random variables. Let's do an example with our specific X and Y from above:

Question: What is $\mathbf{E}[\frac{Y}{X}]$?

Answer: Using the formula and then the table for the joint PMF p_{XY} of X and Y, we have:

$$\begin{aligned}
\mathbf{E}\left[\frac{Y}{X}\right] &= p_{XY}(1,1) \cdot \frac{1}{1} + p_{XY}(1,2) \cdot \frac{2}{1} + p_{XY}(1,3) \cdot \frac{3}{1} + p_{XY}(2,1) \cdot \frac{1}{2} + p_{XY}(2,2) \cdot \frac{2}{2} + p_{XY}(2,3) \cdot \frac{3}{2} \\
&= (1/4) \cdot 1 + (1/4) \cdot 2 + 0 \cdot 3 + (1/6) \cdot \frac{1}{2} + (1/6) \cdot 1 + (1/6) \cdot \frac{3}{2} \\
&= \frac{5}{4}.
\end{aligned}$$

1.2 Independence of random variables

We talked about events being independent — there is also a definition of random variables being independent. As with events, we'll start with the case of two random variables.

Remember the **Principle** which let us infer that two events were independent? Let's imagine a similar scenario. Suppose you have two provably non-interacting blocks of randomized code. Suppose X is a random variable whose value provably depends only on block 1, and Y is a random variable whose value provably depends only on block 2. Then for each real $u \in \mathbb{R}$, the *event* "$X = u$" only depends on block 1; and similarly, for every $v \in \mathbb{R}$, the *event* "$Y = v$" only depends on block 2. Therefore by the Principle, these two events are independent. It follows (by definition) that

$$\mathbf{Pr}[(X = u) \cap (Y = v)] = \mathbf{Pr}[X = u]\mathbf{Pr}[Y = v].$$

Recalling the definition of PMFs and joint PMFs, this can be equivalently stated as

$$p_{XY}(u, v) = p_X(u)p_Y(v). \tag{1}$$

We take equation (1) as our *definition* of independence for random variables.

Definition 2. *X and Y are* independent random variables *if and only if*

$$p_{XY}(u, v) = p_X(u)p_Y(v) \qquad \forall u, v \in \mathbb{R}.$$

Just as with events, the way things work in the real world is that you argue that X and Y are independent using the Principle, and then you *use* equation (1).

Let us mention that one obvious-seeming fact is indeed a fact: Suppose X and Y are independent random variables. Then $f(X)$ and $g(Y)$ are also independent random variables, for any real functions $f, g : \mathbb{R} \to \mathbb{R}$. For example, if X and Y are independent random variables, then $V = X^2 - 1$ and $W = \sin Y$ are also independent random variables. Of course, if the Principle was the *reason* X and Y were independent, then this is obvious — once the non-interacting blocks of code compute X and Y, then can non-interactingly compute $X^2 - 1$ and $\sin Y$. But if X and Y are independent for some other reason — i.e., if (1) holds merely "by a fluke" — then it's not immediately obvious. Still, it's not too hard to prove, and you will do this on the homework.

We've seen the definition for two random variables. What about multiple random variables? As you remember, the definition for a collection of events being independent is slightly complicated. The one for multiple random variables is actually easier:

Definition 3. *Random variables $X_1, \ldots, X_n$ are* independent *if and only if*

$$p_{X_1 X_2 \cdots X_n}(u_1, \ldots, u_n) = p_{X_1}(u_1)p_{X_2}(u_2) \cdots p_{X_n}(u_n) \qquad \forall u_1, \ldots, u_n \in \mathbb{R}. \tag{2}$$

As you can see, it's not exactly analogous — there's none of that subset stuff. Why not? Actually, the above definition *implies* the analogous product rule for subsets — we'll let you think about why, giving partial illustration in a homework problem.

By the way, one ridiculously common way to introduce random variables is to state their distributions (i.e., PMFs) and to state that they are "independent". In that case, you are being told that the joint PMF of the random variables is given by the formula (2). In our world of randomized code, this is also equivalent to just saying the random variables are generated by noninteracting blocks of code.

1.3 $\mathbf{E}[XY]$ when X and Y are independent

Perhaps the most important fact of all about independent random variables is the following:

Theorem 4. *If X and Y are independent random variables then*

$$\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y]$$

is true.

Proof. Using the formula,

$$\begin{aligned}
\mathbf{E}[XY] &= \sum_{\substack{u\in\text{range}(X)\\ v\in\text{range}(Y)}} p_{XY}(u,v)\cdot uv \\
&= \sum_{u,v} p_X(u)p_Y(v)\cdot uv \qquad \text{(by independence)} \\
&= \sum_{u,v} up_X(u)\cdot vp_Y(v) \\
&= \sum_{u} up_X(u)\sum_{v} vp_Y(v) \\
&= \sum_{u} up_X(u)\mathbf{E}[Y] \\
&= \mathbf{E}[Y]\cdot\sum_{u} up_X(u) \\
&= \mathbf{E}[Y]\mathbf{E}[X]
\end{aligned}$$

which of course also equals $\mathbf{E}[X]\mathbf{E}[Y]$. □

The analogous statement for more than 2 random variables is also true:

Theorem 5. *If $X_1, X_2, \ldots, X_n$ are independent random variables then*

$$\mathbf{E}[X_1X_2\cdots X_n] = \mathbf{E}[X_1]\mathbf{E}[X_2]\cdots\mathbf{E}[X_n].$$

Finally, bear in mind that if, e.g., X, Y, and Z are independent, then so are $f(X)$, $g(Y)$, and $h(Z)$; hence, you can deduce things like

$$\mathbf{E}[X^2(Y-1)\sin Z] = \mathbf{E}[X^2]\mathbf{E}[Y-1]\mathbf{E}[\sin Z].$$

1.4 Conditional Expectation

We can condition expected values on an event. The definition is the "obvious" one:

Definition 6. *Let A be an event (with $\mathbf{Pr}[A]\neq 0$) and X a random variable. Then the* conditional expectation of X given A *is*

$$\mathbf{E}[X \mid A] = \sum_{\ell\in\Omega} \mathbf{Pr}[\ell \mid A]X(\ell).$$

We can also use the formula

$$\mathbf{E}[X \mid A] = \sum_{u \in \text{range}(X)} u \cdot \mathbf{Pr}[X = u \mid A].$$

Here is a ridiculously important fact for calculation method, analogous to the Law of Total Probability:

Theorem 7. *Let $A_1, \ldots, A_n$ be a partition of Ω. Then*

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{Pr}[A_i]\mathbf{E}[X \mid A_i].$$

Recall that given another random variable Y and a value $v \in \mathbb{R}$, the expression "$Y = v$" is an event. As a corollary, we have:

Corollary 8. *Let X and Y be random variables. Then*

$$\mathbf{E}[X] = \sum_{v \in range(Y)} \mathbf{Pr}[Y = v] \cdot \mathbf{E}[X \mid Y = v].$$

This is because the collection of all events $Y = v$ partition Ω.

2 Our favorite kinds of random variables

Let me tell you about your two favorite kinds of (discrete) random variables. Perhaps you did not yet know what your favorites are, but you will soon :) Right now we are pretty familiar with Bernoulli random variables and random variables which have the uniform distribution on $\{1, 2, \ldots, m\}$. But these random variables are a bit boring. Let's see some more fun ones.

2.1 Binomial random variables

Definition 9. *Let $X_1, \ldots, X_n$ be* independent *Bernoulli random variables with parameter p. Let $X = X_1 + X_2 + \cdots + X_n$. We say that X is a* binomial *random variable with parameters n and p. We write this as $X \sim Binomial(n, p)$.*

The parameter n is a natural number; the parameter p is a real in the range $[0, 1]$.

The first thing you should always ask yourself on encountering a random variable X is, "what is the PMF of X?" Let's answer that. In the natural experiment generating X there are 2^n possible outcomes; each is a string of 0's and 1's (representing the X_i values). For example, if $n = 9$, one possible outcome is 001011100. The r.v. X has value u if and only if the outcome has u many 1's, and therefore $n - u$ many 0's. The probability of such an outcome is

$$p^u \cdot (1 - p)^{n-u},$$

and the number of such outcomes is $\binom{n}{u}$. Therefore:

Formula: If X is a Binomial(n, p) random variable, its PMF is given by

$$p_X(u) = \binom{n}{u} p^u (1-p)^{n-u}, \qquad u = 0, 1, 2, \ldots, n.$$

As a quick check, we know that the sum of a PMF's values should be 1. And indeed,

$$\begin{aligned} \sum_{u=0}^{n} \binom{n}{u} p^u (1-p)^{n-u} &= (p + (1-p))^n \qquad \text{(by the "Binomial Theorem")} \\ &= 1^n \quad = \quad 1. \end{aligned}$$

The second thing one tends to ask oneself after seeing a random variable is "what is $\mathbf{E}[X]$?" We could try to start computing

$$\mathbf{E}[X] = \sum_{u=0}^{n} u \cdot \binom{n}{u} p^u (1-p)^{n-u} = \cdots$$

but of course, the smart way to compute $\mathbf{E}[X]$ is to use linearity of expectation:

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \cdots + X_n] = \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_n] = p + \cdots + p = np.$$

Formula: If $X \sim$ Binomial(n, p), then $\mathbf{E}[X] = np$.

As a small point, we often think of a Binomial(n, p) random variable X *per se*, with no reference to any independent Bernoulli X_i's summing to it; it's just a random variable with the PMF given above. Of course, if you want you can introduce the X_i's and act as though they existed all along.

2.2 Geometric random variables

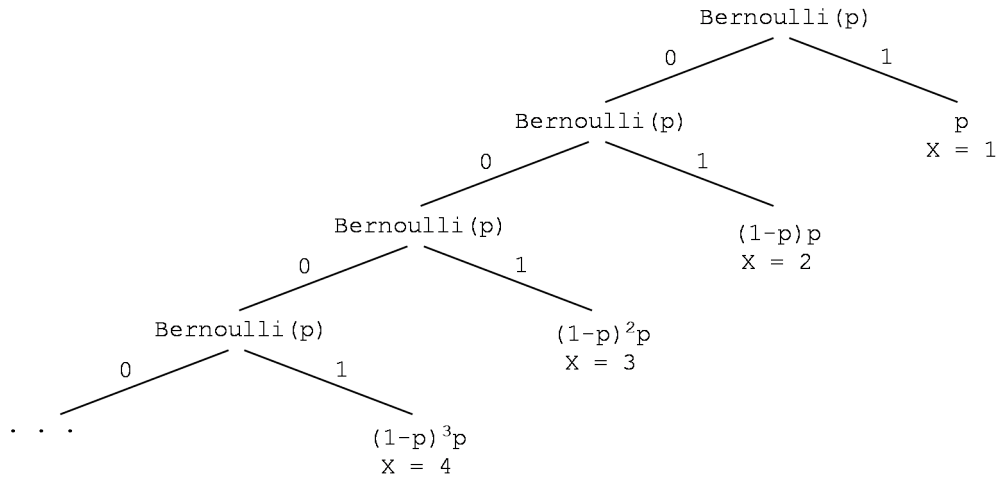
Definition 10. *Suppose we repeatedly draw independent Bernoulli random variables with parameter $p > 0$ until we get a 1. Let X denote the number of trials up to and including the first 1. We then say that X is a* geometric *random variable with parameter p. We write this as $X \sim$ Geometric(p).*

In other words, we are considering the following experiment:

$X \leftarrow 1$
`while` Bernoulli$(p) = 0$,
    $X \leftarrow X + 1$

Please note that the range of possible values for X is $\{1, 2, 3, \ldots\}$.[1] This is our first example of a random value whose range is not finite (it is countably infinite). Do not be alarmed! It's cool. Here is the associated probability tree for the above experiment; at each outcome we've written the probability of the outcome and also X's value:

[1] Do not think that "∞" is a possible value for X. "∞" is not a number.



It's easy to see the PMF from this tree:

Formula: If X is a Geometric(p) random variable, its PMF is given by

$$p_X(u) = (1-p)^{u-1}p, \qquad u = 1, 2, 3, \ldots$$

Again, let's check that those values sum to 1:

$$\sum_{u=1}^{\infty} p_X(u) = \sum_{u=1}^{\infty}(1-p)^{u-1}p = p \cdot (1 + (1-p) + (1-p)^2 + (1-p)^3 + \cdots) = p \cdot \frac{1}{1-(1-p)} = \frac{p}{p} = 1.$$

In the second-to-last step there we used the formula for the sum of a geometric series. This is where the Geometric random variable gets its name.

Just as we sometimes "forget" how a Binomial-distributed r.v. X was generated (summing independent Bernoullis), sometimes we "forget" about the experiment generating a Geometric random variable X, and just remember its PMF, as given above. Once again, you can imagine introducing the experiment if you want to.

What about the expected value of X, when $X \sim$ Geometric(p)? If I told you I was flipping a coin with a 5% chance of heads and asked what the average number of flips required was to get a head, well, you'd probably guess 20. And you'd be right!

Formula: If $X \sim$ Geometric(p), then $\mathbf{E}[X] = 1/p$.

There are many fun and instructive ways to prove this; we encourage you to try to come up with your own proof. Here is a fairly straightforward one. By definition, we have

$$\mathbf{E}[X] = \sum_{u=1}^{\infty} u \cdot p_X(u) = 1 \cdot p + 2 \cdot (1-p)p + 3 \cdot (1-p)^2 p + 4 \cdot (1-p)^3 p + \cdots \tag{3}$$

How do we evaluate this infinite series?[2] Here's a trick that works:

[2]For the worry-mathers among you: You might be concerned about infinite sums here. First, when a random variable X has countably infinitely many possible outcomes, we make the following convention: $\mathbf{E}[X]$ is *only defined* if $\mathbf{E}[|X|] < \infty$: i.e., $\sum_{u \in \text{range}(X)} p_X(u)|u|$ converges. In the particular case of a geometric random variable, this means that before we declare $\mathbf{E}[X] = 1/p$, we first have to verify that the series in (3) converges (note that $|X| = X$ already here). This justification is a straightforward application of the Ratio Test, assuming $p \neq 0$!

Take equation (3),

$$\mathbf{E}[X] \quad = \quad 1 \cdot p + 2 \cdot (1-p)p + 3 \cdot (1-p)^2 p + 4 \cdot (1-p)^3 p + \cdots$$

and multiply it by $1-p$, giving

$$(1-p)\mathbf{E}[X] = \quad 1 \cdot (1-p)p + 2 \cdot (1-p)^2 p + 3 \cdot (1-p)^3 p + \cdots$$

Now subtract the second equation from the first, yielding

$$p\mathbf{E}[X] = \quad p \quad + (1-p)p \quad + \quad (1-p)^2 p \quad + \quad (1-p)^3 p + \cdots$$

and we saw that this last sum is 1, when checking the PMF of X. Hence $p\mathbf{E}[X] = 1$; i.e., $\mathbf{E}[X] = 1/p$, as claimed.

3 Linearity of Expectation + Indicators method

Now that we have a good amount of theory of random variables/expectation down, let's solve some problems.

One outstanding problem-solving technique is what we call the "Linearity of Expectation + Indicators method':

Linearity of Expectation plus Indicators method: Suppose that there are m different events $A_1, \ldots, A_m$ that might occur, and you want to know the expected *number/count* that occur. Let X be a random variable counting the number of events occurring, and write

$$X = X_1 + X_2 + \cdots + X_m,$$

where X_i is the indicator random variable for event A_i. Use Linearity of Expectation to deduce

$$\mathbf{E}[X] = \mathbf{E}[X_1] + \mathbf{E}[X_2] + \cdots + \mathbf{E}[X_m],$$

and then use the fact that the $\mathbf{E}[X_i] = \mathbf{Pr}[A_i]$ to deduce

$$\mathbf{E}[X] = \mathbf{Pr}[A_1] + \mathbf{Pr}[A_2] + \cdots + \mathbf{Pr}[A_m].$$

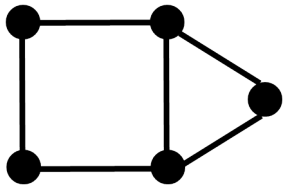
Let's do an example. The method is so great, we'll do a couple more examples in the next lecture.

3.1 The Enemybook Schism Problem

Do you know about Enemybook? Enemybook is an anti-social utility that disconnects you to the so-called friends around you.[3] It's just like Facebook, except with "enemyships" connecting people, rather than friendships.

Suppose that there are n students using Enemybook, and there are m enemyships among them. (Enemyships are between pairs of students.) It's natural to illustrate the state of Enemybook with a graph: the nodes are students, the edges are enemyships. Here's an example with $n = 5$, $m = 6$:

[3] Actually, Enemybook is a Facebook app created by Prof. O'Donnell's colleague Kevin Matulef. Now with SuperFlipOff app!

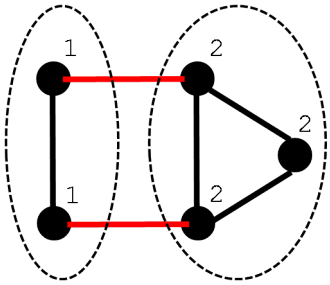


We're interested in devising a schism in Enemybook: we want to split the students into two "teams" so that many enemyships are "broken". We're trying to get really cohesive teams here, so we are happy when two enemies get split up.

Here's a really simple thing we can do: just divide the students into two teams at random. By this we mean precisely:

```
for i ← 1...n
    team[i] ← RandInt(2)
```

Question: We say an enemyship is broken when the two enemies are put into different teams. Let X be the number of broken enemyships when we pick two random teams as described above. What is $E[X]$? I.e., how many enemyships do we break up on average?



Example division into two teams. Broken enemyships shown in red.

Let us tell you, it is pretty impossible to solve this by calculating the PMF of X, when you aren't given any specific information about the Enemybook graph. But, whenever you get a problem that asks about the expected number of events (of a certain type) that occur, you should think about using the Linearity of Expectation plus Indicators method.

Answer: As suggested by the method, the trick is to *write X as a sum of indicator random variables*. Specifically, for each enemyship e in the graph, introduce an indicator random variable X_e for the event B_e that enemyship B_e is broken. Recall this means that $X_e = 1$ if B_e happens — i.e., edge e is broken — and $X_e = 0$ if e is unbroken. The key observation is that

$$X = \sum_{e \text{ in graph}} X_e.$$

Remember, this is a fact about random variables: whatever the actual outcome is, if you add up all the indicator variables (for an outcome), this gives the number of enemyships broken (for that outcome). Hence by Linearity of Expectation,

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_e X_e\right] = \sum_e \mathbf{E}[X_e] = \sum_e \mathbf{Pr}[B_e],$$

where the last step used the fact we saw in the last lecture, that the expectation of an indicator random variable equals the probability of the event it indicates. But, for each edge e, we claim that $\mathbf{Pr}[B_e] = 1/2$. This is just because this enemyship e gets broken up if and only if the two enemies go to different teams, which clearly happens with probability 1/2. Hence we get

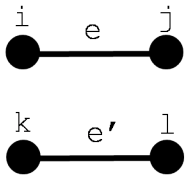
$$\mathbf{E}[X] = \sum_e (1/2) = \frac{m}{2}.$$

So the expected number of enemyships broken up is exactly half of the number of enemyships. Not too bad! □

3.2 Non-independence

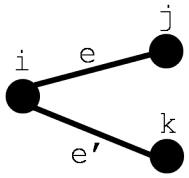
Please note that in general, the collection of B_e events is *not* independent. (Of course, the above argument never suggested they were.) Let's investigate this statement, since it will give us some practice in analyzing independence.

Suppose we have two "disjoint" enemyships as shown below:



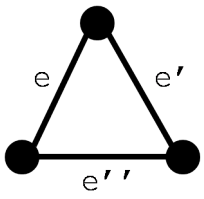
e joins students i and j and e' joins students k and ℓ. Are B_e and $B_{e'}$ independent events? Actually, yes. To argue this, it suffices to say, "B_e is completely determined by the team coin flips for i and j. $B_{e'}$ is completely determined by the team coin flips for k and ℓ. And these pairs of coin flips do not at all affect one another."

What about the following case:



Are B_e and $B_{e'}$ independent? Actually yes — and this is a bit of a "trick". The point is that you can easily convince yourself that $\mathbf{Pr}[B_e \cap B_{e'}]$, the probability that both enemyships get broken, is 1/4. (Basically, regardless of i's team, j and k have to be on different teams — 1/2 chance each, independently.) And 1/4 is indeed $\mathbf{Pr}[B_e] \cdot \mathbf{Pr}[B_{e'}]$. So indeed B_e and $B_{e'}$ are independent here.

Finally, what about this case:



Are B_e, $B_{e'}$, and $B_{e''}$ independent here? Finally, the answer is "no", justifying our early statement that the B_e events are not in general independent. One way to see this is that $\mathbf{Pr}[B_e \cap B_{e'} \cap B_{e''}] = 0$! There's just no way for all three enemyships to be simultaneously broken. But $\mathbf{Pr}[B_e] \cdot \mathbf{Pr}[B_{e'}] \cdot \mathbf{Pr}[B_{e''}] = 1/8 \neq 0$, so the three events are not independent.

3.3 Max-Cut

You might think that the Enemybook Schism problem is a bit silly, but it's actually a dressed-up version of an extremely well-known and basic algorithmic task called "Max-Cut". Here you are given a graph G and the task is to partition the vertices into 2 parts so that as many edges as possible are "cut" (broken). This problem comes up in practical applications, in circuit design, vision, and statistical mechanics. There is a great deal of contemporary algorithmic work on coming up with good Max-Cut algorithms.

Now it is known that the Max-Cut problem is actually "NP-complete". (Indeed, it was one of the very first problems to be shown NP-complete, by Karp in 1972.) This means it is highly unlikely there is an efficient algorithm guaranteed to always find the cut-maximizing partition in an input graph. But what if we don't care about finding the absolute maximizing partition? What if we are interested in just finding a pretty good partition that cuts a decent fraction of edges? Believe it or not, the completely brain-dead-seeming randomized method just described (proposed first in the mid 1970's) was the *best known efficient algorithm* for about 20 years! It was not until the mid 1990's that Michel Goemans and David Williamson came up with a better efficient algorithm (also a randomized algorithm) whose expected-number-of-cut-edges was significantly better than $m/2$! Specifically, the Goemans-Williamson algorithm guarantees finding a partition cutting at least .878 times as many edges as whatever the maximizing partition cuts.

15-359: Probability and Computing
Fall 2009

Lecture 7: Linearity + Indicators applications:
Max-Cut, Changes to the Minimum, Quicksort

This lecture is devoted to some applications illustrating the power of the "Linearity of Expectation + Indicators" Method.

1 Max-Cut

Recall the "Enemybook Schism" problem from last time. You might think it a bit silly, but it's actually a dressed-up version of an extremely well-known and basic algorithmic task called "Max-Cut". Here you are given a graph $G = (V, E)$ and the task is to partition the vertices into 2 parts so that as many edges as possible are "cut"; i.e., go between part 1 and part 2. This problem comes up in several practical applications, in circuit design, vision, and statistical mechanics. There is a great deal of contemporary algorithmic work on coming up with good Max-Cut algorithms.

As we saw last time, the almost brain-dead algorithm of putting each vertex randomly into either part 1 or part 2 has the property that if there are m edges, then the expected number of cut edges is $m/2$; that is, half of them.

This is actually pretty good! It is known that the Max-Cut problem is actually "NP-complete". (Indeed, it was one of the very first problems to be shown NP-complete, by Karp in 1972.) This means it is highly unlikely there is an efficient algorithm guaranteed to always find the cut-maximizing partition in an input graph. So giving an efficient algorithm for cutting half of the edges is pretty good![1]

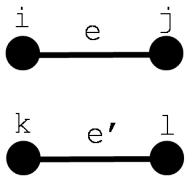
In fact, this completely trivial algorithm, first proposed in the mid 1970's, was the *best known efficient Max-Cut algorithm* for about 20 years! It was not until the mid 1990's that Michel Goemans and David Williamson came up with a better efficient algorithm (also a randomized algorithm) whose expected-number-of-cut-edges is significantly better than $m/2$! Specifically, the Goemans-Williamson algorithm guarantees finding a partition cutting at least .87856 times as many edges as whatever the maximizing partition cuts.

1.1 Non-independence

Before going on, it's nice to point out that in our analysis of the random algorithm for Max-Cut, the events B_e we introduced are *not* in general independent. Recall that B_e was the event that the particular edge e was cut. Let's look carefully:

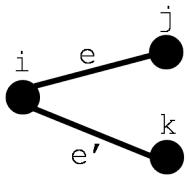
Suppose we have two "disjoint" enemyships/edges as shown below:

[1] You might like to prove that the natural greedy algorithm always cuts at least half the edges too, and does so deterministically.



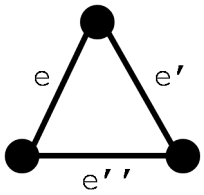
e joins students i and j and e' joins students k and ℓ. Are B_e and $B_{e'}$ independent events? Actually, yes. To argue this, it suffices to say, "B_e is completely determined by the team coin flips for i and j. $B_{e'}$ is completely determined by the team coin flips for k and ℓ. And these pairs of coin flips do not at all affect one another."

What about the following case —



— are B_e and $B_{e'}$ independent? Actually yes — and this is a bit of a "trick". The point is that you can easily convince yourself that $\mathbf{Pr}[B_e \cap B_{e'}]$, the probability that both enemyships/edges get cut is $1/4$. (Basically, regardless of i's team, j and k have to be on different teams — $1/2$ chance each, independently.) And $1/4$ is also equal to $\mathbf{Pr}[B_e] \cdot \mathbf{Pr}[B_{e'}]$. So indeed B_e and $B_{e'}$ are independent here.

Finally, what about this case?



Are B_e, $B_{e'}$, and $B_{e''}$ independent here? Finally, the answer is "no", justifying our early statement that the B_e events are not in general independent. One way to see this is that $\mathbf{Pr}[B_e \cap B_{e'} \cap B_{e''}] = 0$! There's just no way for all three enemyships/edges to be simultaneously cut. But $\mathbf{Pr}[B_e] \cdot \mathbf{Pr}[B_{e'}] \cdot \mathbf{Pr}[B_{e''}] = 1/8 \neq 0$, so the three events are not independent.

2 Changes to the Minimum

Here is another elegant problem that can be solved with the Linearity of Expectation + Indicators method. Let's say we have an array of N numbers, "`data[ ]`". For simplicity, assume all the numbers in the array are distinct. We would like to compute the minimum of the numbers. Of course, there's nothing to do but the following:

```
Compute-Min:
    min ← ∞
    for t ← 1...N
        if data[t] < min
```

```
min ← data[t]
print "The new minimum is ", min          (*)
```

Let's assume that line (*) is very expensive. Indeed, output operations usually are far slower than simple logical/arithmetic/assignment operations. Or, you might imagine that every time the minimum changes, you have to go update a web page; or, every time the minimum changes, you have to go rebuild a data structure. We will see an example of this last possibility on an upcoming homework.

The question then becomes:

Question: How many times does line (*) get executed?

Answer: Well, it depends `data[ ]`, naturally! Actually, let's make one additional observation: It doesn't actually depend on the *magnitudes* of the values in `data[ ]`; it just depends on their *comparative* magnitudes; i.e., which t corresponds to the smallest value, which t corresponds to the second-smallest value, etc.

So what are the possibilities? Let's think about two natural cases:

Worst case: It's not too hard to see that the "worst case" is when the `data` array is sorted in descending order. Then the minimum will change (i.e., we'll execute (*)) for every single value of t. I.e., in the worst case, there are N changes to the minimum.

Average case: What does this mean? Let's assert that it means that each of the $N!$ different possible orderings is equally likely. At the formal level of *modeling*, you can imagine you insert an "experiment" (i.e., randomized code) at the beginning of the `Compute-Min` code which randomly permutes `data[ ]`. Now sometimes (as we will see), it can be time-saving to *literally do this*, depending on how expensive (*) really is. But even if we don't, we can imagine scenarios in which the data actually *can* be assumed randomly ordered. For example, suppose `data[ ]` was generated by choosing each entry from `RandInt`(2^{64}). If you think about it, then each possible ordering of `data[ ]` becomes equally likely (conditioning on no equal entries). Or at a less formal level, you might just say, "Hey, in my application I don't really expect anything in particular about the initial ordering of `data[ ]`, so I choose to model things probabilistically as if each possible ordering had probability $1/N!$."

In any case, under this assumption, let X be the random variable giving the number of times that line (*) is executed. What is $\mathbf{E}[X]$?

As promised, we use the Linearity of Expectation + Indicators method. The obvious way to do this is to first let A_t be the event that (*) is executed on step t. This introduces N events. Next, let X_t be the indicator random variable for A_t. We now make the important observation that

$$X = \sum_{t=1}^{N} X_t.$$

Thus by linearity of expectation,

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{t=1}^{N} X_t\right] = \sum_{t=1}^{N} \mathbf{E}[X_t] = \sum_{t=1}^{N} \mathbf{Pr}[A_t], \tag{1}$$

where in the last step we used the fact that the expectation of an indicator random variable equals the probability of the event it indicates.

Okay, now what is $\mathbf{Pr}[A_t]$. If we think carefully about what A_t means, we see that

$$A_t = \text{event that ``}\texttt{data}[t] \text{ is the minimum among } \{\texttt{data}[1], \ldots, \texttt{data}[t]\}\text{''}. \tag{2}$$

We claim that the probability of this is precisely $1/t$. We will give an argument for this which is a bit informal compared to the arguments we've made in the class so far. (Indeed, as we all "get used" to probability, we'll start making more and more informal arguments. However, we should all take the time once in a while to really check such statements 100% rigorously. We strongly encourage you to do this here, with probability trees and combinatorics...) One can think of the experiment of randomly permuting the N data items in the following nonstandard-but-equivalent way: First, t out of the N data items are chosen, with equal probability $1/\binom{n}{t}$ of choosing each possible subset. These items are then randomly permuted, forming the first t elements in the array. Finally, the remaining $N-t$ are randomly permuted, forming the last $N-t$ elements of the array.

Take some time to convince yourself that this is equivalent to simply choosing a random permutation of the N items.

Given this alternate way of looking at the experiment, one sees that the event A_t only depends on the choice of the first t elements and how they're permuted, not on how the remaining $N-t$ elements are permuted. Further, suppose we *condition* on which t elements get chosen to be the first t. Then one of these elements is smallest, and when we do the permutation this smallest item is equally likely to go into each of $\texttt{data}[1], \ldots, \texttt{data}[t]$. Hence the probability the smallest item goes into $\texttt{data}[t]$ — which is $\mathbf{Pr}[A_t]$ — is precisely $1/t$.

Again, please take some time to rigorously justify the overall argument to yourself.

Done? Great. Now we can continue from (1), knowing that $\mathbf{Pr}[A_t] = 1/t$. We get

$$\mathbf{E}[X] = \sum_{t=1}^{N} (1/t) = H_N.$$

You may be asking, "What is H_N?" Well, it's defined to be $\sum_{t=1}^{N}(1/t)$. It's called the "Nth harmonic number". It's known to be quite close to $\ln N$, the natural logarithm of N. More precisely,

$$H_N = \ln N + \gamma \pm O(1/N),$$

where $\gamma = .577\cdots$ is the "Euler-Mascheroni constant". Sometimes computer scientists get a little slacky and just remember

$$H_N \leq O(\log N).$$

You should definitely remember at least this. By the way, the reason that $H_N \approx \ln N$ is

$$\sum_{t=1}^{N}(1/t) \approx \int_1^N (1/t)\,dt = \ln t\,|_{t=1}^{t=N} = \ln N - \ln 1 = \ln N.$$

So anyway, we have that the expected number of changes to the minimum, $\mathbf{E}[X]$, is about $\ln N$.

Conclusion: The "average case" ($O(\log N)$ changes on average) is *exponentially* cheaper than the "worst case" (N changes) for the number of changes to the minimum.

3 Quicksort

Let's do an even more interesting — and also more complicated — example: Quicksort. "In practice" (whatever that means, exactly), it seems that Quicksort is one of the fastest and most popular general sorting algorithms. One reason for this is that it is an "in-place" algorithm requiring no additional storage. Another reason is that the code is pretty simple and operates well with respect to the cache. The funny thing is, though, Quicksort is actually a very poor sorting algorithm from the point of view of "worst-case" theoretical guarantees. But as we'll see, Randomized Quicksort has very good "average case" or "typical case" performance.

Here is the pseudocode for Quicksort, omitting implementation details:

Quicksort: Input is a list $S = (x_1, \ldots, x_n)$ of numbers (assumed distinct, for simplicity).
    if $n \leq 1$, return S
    pick, somehow, a "pivot", x_m
    compare x_i to all other x's (**)
    let S_1 = the list of x's which are $< x_m$
    let S_2 = the list of x's which are $> x_m$
    recursively Quicksort each of S_1, S_2
    return $[S_1, x_i, S_2]$

Without formally doing the details (even though they are simple to do), please believe this:

Fact: The running time of Quicksort is proportional to the total number of *comparisons* made across all executions of line (**).

How many comparisons Quicksort actually makes depends on two things: (i) the rule (not yet specified) for how the pivot is chosen; (ii) the initial ordering of the data in the list. Intuitively, good things happen when the pivot divides the list S into roughly equal-sized halves: then the divide-and-conquer paradigm kicks in. But bad things can happen if this fails. Before doing some analysis, let's make the following assumption:

Assumption: The initial list consists of the values ①, ②, ③, …, ⓝ in some unknown order.

Really, we're assuming here that the list consists of the value 1, 2, 3, …, n, but we'll draw circles around them to remind ourselves that these are data elements in the list. For analysis purposes,

this assumption is *without loss of generality*. The reason is that, just like with the Changes-to-the-Minimum problem, the algorithm/analysis does not depend on the actual magnitudes of the data, just the *comparative* magnitudes. So it's like we're just using ① to mean "the smallest number in S", using ② to mean the "second-smallest number in S", etc.

Okay, on to the analysis of Quicksort!

Worst case analysis: Quicksort may make $\Omega(n^2)$ comparisons, in the worst case. For example, suppose the pivot-choosing rule is "always pivot on x_1" and that the initial input list is in descending order, $[\text{(n)}, \ldots, ③, ②, ①]$. Then it's easy to see that the algorithm's "recursion tree" is actually a path, and it repeatedly splits the list $[\text{(i)}, \ldots, ①]$ into $[\text{(i-1)}, \ldots, ①]$ and $[\ ]$. Thus the total number of comparisons is

$$(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2} = \Omega(n^2).$$

Pretty bad, since of course we all know several different sorting algorithms which use $O(n \log n)$ comparisons/time in the worst case.

Randomized Quicksort: Like with the Changes-to-the-Minimum problem, we could imagine that the initial list S is randomly ordered and see what happens. But the beautiful observation here is that we don't have to — we can effectively convert to this case by doing the following:

always choose the pivot uniformly at random.

The intuition behind this is that if you pick a pivot element at random, there should be a good chance that you pick an element which is somewhat in the middle of the sorted order, and thus your division will be into two roughly equal-sized lists. In fact, if you work at it you can analyze this intuition, but we can give a far slicker analysis using the Linearity of Expectation + Indicators method.

3.1 Analyzing Randomized Quicksort

Let the initial list S consist of $①, \ldots, \text{(n)}$ in some order, and suppose we do Randomized Quicksort, meaning we always choose the pivot uniformly at random from among all elements in the current list. Let C be the random variable equal to the total number of comparison made over the course of the algorithm. Our goal is to compute $\mathbf{E}[C]$. Interestingly, it will turn out that $\mathbf{E}[C]$ does not depend on the initial ordering of the list.

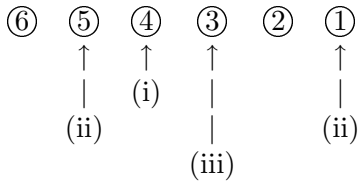
To use the Linearity of Expectation + Indicators method, we want to express C as a sum of indicator random variables. You might consider different ways of doing this, but here is one that works. Define A_{ij} be the event that (i) and (j) ever get compared. Here we have an event for each $1 \leq i < j \leq n$. Let C_{ij} be the indicator r.v. for A_{ij}. Notice that any two elements in the array either get compared zero times or one time — never two or more times. This is because if we ever compare (i) and (j), it's because one of them is the pivot, and the pivot never participates in any future comparisons. Thus we have

$$C = \sum_{1 \leq i < j \leq n} C_{ij},$$

and we conclude (by linearity of expectation and the fact that the expectation of an indicator is the probability of the event)

$$\mathbf{E}[C] = \sum_{1 \le i < j \le n} \mathbf{Pr}[A_{ij}]. \tag{3}$$

Let's do an example, as that will significantly clarify the picture. Suppose that the initial list S is [⑥, ⑤, ④, ③, ②, ①]. (This is a nice case to think about, since it was the bad case for our deterministic Quicksort example.) Recall now that in each (recursive) call to Quicksort, we're imagining that the entry to pivot on is chosen uniformly at random. In our example, the three rounds of pivots might look like this:



Let's explain. Here, at level (i) in the recursion we randomly chose to pivot on the third entry, which happens to contain ④. So S_1 becomes [③, ②, ①] and S_2 becomes [⑥, ⑤]. At level (ii) in the recursion, we randomly chose to pivot on the third entry in [③, ②, ①] (i.e., on the entry containing ①), and to pivot on the second entry in [⑥, ⑤] (i.e., on the entry containing ⑤). So [③, ②, ①] splits into [ ] and [③, ②]; and, [⑥, ⑤] splits into [ ] and [⑥]. In level (iii) of the recursion, everything is done except for the sublist [③, ②]; here we randomly choose to pivot on the first entry, ③. We recurse on [②] and [ ], and no further comparisons are made. So in this example we have:

(i) $C_{14} = C_{24} = C_{34} = C_{45} = C_{46} = 1$, because we compared ④ against everything else;
(ii) $C_{12} = C_{13} = 1$ and $C_{56} = 1$;
(iii) $C_{23} = 1$;

and all other C_{ij}'s are 0.

So much for the example; back to the analysis. Given equation (3), all we have to do is compute $\mathbf{Pr}[A_{ij}]$ for each $i < j$. To do this, consider the *set* of data elements

$$Y^{ij} = \{\text{ⓘ}, \text{ⓘ+1}, \ldots, \text{ⓙ}\}.$$

Initially, all the elements in Y^{ij} are hanging out together in the list S. (They're not necessarily in order, or even contiguous, of course.) Think about what happens to the set when the pivot is chosen.

- If the pivot's value is not in the set Y^{ij} then it's either bigger than everything in Y^{ij} or else it's smaller than everything in Y^{ij}. Either way, all the elements in Y^{ij} will be together in one of the two sublists recursed on.
- If the pivot's value is *strictly* between ⓘ and ⓙ then:
  - ⓘ and ⓙ are *not compared* in this level of the recursion;
  - and in fact they will *never* be compared, because ⓘ will go into S_1 and ⓙ will go into S_2, which are sorted separately.
- Finally, if the pivot is actually ⓘ or ⓙ then these two elements *are* compared, because the pivot gets compared to everything.

Conclusion: Event A_{ij} occurs if and only if the first pivot chosen from Y^{ij} is ⓘ or ⓙ.

But: Think about the probability of this. The first time a pivot is chosen from Y^{ij}, all of these elements are hanging out together in some sublist. *Conditioned* on one of them being chosen as the pivot, it's clear from symmetry that each of them is equally likely to be the chosen pivot. Since there are $j - i + 1$ elements in Y^{ij}, we conclude

$$\mathbf{Pr}[A_{ij}] = \frac{2}{j-i+1}. \tag{4}$$

Let's quickly sanity-check this claim. It should make sense that the probability that ⓘ and ⓙ are compared is bigger the closer i and j are together as numbers. If i and j are far apart numbers, probably they will never get directly compared; probably they'll get split apart at some stage when an intermediate pivot is chosen. On the other hand suppose i and j are really close together as numbers; e.g., $i = 7$, $j = 8$. Equation (4) is claiming that ⑦ and ⑧ are compared with probability $\frac{2}{8-7+1} = 1$ — i.e., always! But indeed, this is true. Any sorting algorithm *has* to compare ⑦ and ⑧ at some point, because otherwise it has no way of knowing which of the two should come first in sorted order. Think about it.

In any case, we're now home free.

3.2 Finishing the calculation

Combining equations (3) and (4), we get that the expected number of comparisons made is

$$\mathbf{E}[C] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = 2 \sum_{1 \le i < j \le n} \frac{1}{j-i+1}. \tag{5}$$

Let's compute the sum on the right. If we imagine first adding up all the terms with $i = 1$, then with $i = 2$, etc., we get

$$\begin{array}{ccccccccccc} & \frac{1}{2} & + & \frac{1}{3} & + & \cdots & + & \frac{1}{n-1} & + & \frac{1}{n} \\ + & \frac{1}{2} & + & \frac{1}{3} & + & \cdots & + & \frac{1}{n-1} & & \\ + & \cdots & & & & & & & & \\ + & \frac{1}{2} & + & \frac{1}{3} & & & & & & \\ + & \frac{1}{2} & & & & & & & & \end{array}$$

If we're being lazy, we can just say that the first row adds up to $H_n - 1 \le \ln n$, and each of the other rows adds up to something even smaller. Hence the total sum is at most

$$(n-1)\ln n \le n \ln n,$$

and (remembering the factor of 2 in equation (5)) we have

$$\mathbf{E}[C] \le 2n \ln n.$$

In other words, Randomized Quicksort makes at most $2n \ln n$ comparisons on average. Not only is this $O(n \log n)$, the constant out front is extremely small!

By the way, if you're a little more fastidious, you can sum up the table of fractions above in a column-wise fashion, do some tiny arithmetic rearrangement, and conclude that

$$\mathbf{E}[C] = 2(n+1)H_n - 4n.$$

So indeed, as mentioned earlier, the *expected* number of comparisons Randomized Quicksort does is always the same, $2(n+1)H_n - 4n$, regardless of the initial ordering in S.

15-359: Probability and Computing
Fall 2009

Lecture 8: Markov's Inequality, Variance, Standard Deviation, Chebyshev's Inequality

1 The expected value of an r.v. doesn't always tell you too much about it

Let's play a gambling game. Er, we probably shouldn't encourage you to gamble, so let's *imagine* a gambling game. Here is the gamble: With probability 99%, you owe me \$10. With probability 1%, I owe you \$1000.

Do you want to play?

Seriously. I'm not sure what you'd say. Perhaps it depends how risk-averse you are. Perhaps it depends if I assent to play the game multiple times. Perhaps your beloved puppy has been dognapped and the ransom, due tomorrow, is exactly \$1000.

Some people automatically think, "Calculate the expected winnings — if it is positive you should play; if it's negative you shouldn't." But the expectation of a random value does *not* tell you everything there is to know about a random variable.

Still, it's a start. Let X represent *your* winnings in dollars, in one play of this gamble. We can easily calculate

$$\mathbf{E}[X] = .99 \cdot (-10) + .01 \cdot (+1000) = -9.9 + 10 = .1 = 10 \text{ cents.}$$

The gamble is thus sort of in your favor — you win 10 cents "on average" when you play. But this number is deceptively small compared to the actual values that X can take on. Indeed, in deciding whether or not to play this gamble, it's probably not so germane to you that your expected winnings is 10 cents. You're probably more interested in the *probability* with which you win various amounts. For example, clearly

$$\mathbf{Pr}[X \geq \mathbf{E}[X]] = \mathbf{Pr}[X \geq .1] = .01$$

(since the only case where you win more than 10 cents is when you win the \$1000). Restating this with emphasis, the probability that you win at least your expected winnings is only 1%. This is a far cry from the naive (false) intuition that you ought to win at least your average winnings with probability 1/2. Actually, things are even rougher for you: the probability you win any money at all, $\mathbf{Pr}[X \geq 0]$, is only 1%.

By contrast, let's look at the random variable for *my* winnings, in dollars. Obviously, $Y = -X$, so clearly $\mathbf{E}[Y] = -\mathbf{E}[X] = -.1$. I lose 10 cents on average. But,

$$\mathbf{Pr}[Y \geq \mathbf{E}[Y]] = \mathbf{Pr}[Y \geq -.1] = .99.$$

So the probability I win at least my expected winnings is 99%, just the opposite. In conclusion, we've seen that a random variable (e.g., X) might be almost *never* at least its expectation, and also that a random variable (e.g., Y) might be almost *always* at least its expectation.

2 Markov's inequality

2.1 A worry about our Changes-to-the-Minimum analysis

Remember last time we talked about the Changes-to-the-Minimum problem. We were feeling quite pleased about ourselves because X, the number of changes to the minimum when the data was permuted randomly, satisfied $\mathbf{E}[X] = H_n \approx \ln n$ — and this is much much smaller than the worst-case possibility of n.

But now you should be feeling a little queasy about things. True, $\mathbf{E}[X] \approx \ln n$, but if it were the case that 99% of the time the number of changes to the minimum was n, that would be no good at all.

Question: Could it really be that $X \geq n$ has probability .99?

Answer: Well, no, because it's pretty easy to see that $X \geq n$ iff $X = n$ iff the data is in descending order. Since the data is randomly ordered, this only happens with probability $1/n! \ll .99$.

Question: Okay, but could it be that, say, $\mathbf{Pr}[X \geq n/2] \geq .99$?

Your intuition is probably that the answer is "no" — and you'd be right. But why? How is it different from the situation with my gambling winnings, above, where we had a 99% chance that Y is far far bigger than its expectation?

The difference is that X, the number of changes to the minimum in the algorithm, only takes on *nonnegative* values (which is not true of my gambling winnings Y).

This is the first illustration of a theme for this lecture — kind of a "The More You Know" theme. The more you know about a random variable — e.g., that it is always nonnegative — the more you can "tame" it.

Coming back to this Changes-to-the-Minimum problem, suppose by way of contradiction that 99% of the time we have $X \geq n/2$. Intuitively, it seems clear that the average value of X, i.e. $\mathbf{E}[X]$, should be at least $.99 \cdot (n/2)$, since there are no possible negative values of X to bring it lower. But

$$.99 \cdot (n/2) = .495n \gg \ln n \approx H_n = \mathbf{E}[X],$$

a contradiction. To make this more formal,[1] we can note that (since the two events $X \geq n/2$, $X < n/2$ partition the sample space)

$$\mathbf{E}[X] = \mathbf{Pr}[X \geq n/2] \cdot \mathbf{E}[X \mid X \geq n/2] + \mathbf{Pr}[X < n/2] \cdot \mathbf{E}[X \mid X < n/2]. \tag{1}$$

Let's look at the expressions on the right; three of them are easy to reason about, and the fourth is the one we care about. It's easy to see (and intuitively obvious) that $\mathbf{E}[X \mid X \geq n/2] \geq n/2$.

[1] But not 100% formal, because we will ignore for now the issue of conditioning on 0-probability events.

Further, $\mathbf{Pr}[X < n/2] \geq 0$ being a probability, and $\mathbf{E}[X \mid X < n/2]$ is also nonnegative since X is *always* nonnegative. Therefore from equation (1) we can deduce

$$\mathbf{E}[X] \geq \mathbf{Pr}[X \geq n/2] \cdot (n/2) + 0 \qquad \Rightarrow \qquad \mathbf{Pr}[X \geq n/2] \leq (2/n)\mathbf{E}[X] \approx \frac{2 \ln n}{n}.$$

Of course, $(2 \ln n)/n$ is a very small number if n is large, so we conclude that it is very unlikely that $X \geq n/2$. Phew!

2.2 Markov's Inequality

Andrey Andreyevich Markov lived from 1856 to 1922; he was a contemporary of Tolstoy. Markov was a pretty cool guy and a powerful mathematician. You might recognize his name from, e.g., "Markov Chains" (which we will talk a lot about later in this course). Today we will talk about "Markov's Inequality".[2] By the way, the early history of probability was dominated by French mathematicians (although Bernoulli, whose name sounds French, was actually Swiss). Starting in the late 19th century and early 20th century, it came to be dominated by Russian mathematicians; for example, Kolmogorov (1903–1987) pretty much invented rigorous probability theory. We'll hear about two more Russians today, Markov and Chebyshev.

Anyway, here is Markov's Inequality, which generalizes the example we just did with the Changes-to-the-Minimum problem:

Theorem 1. *("**Markov's Inequality**") Let X be a random variable which is always nonnegative. Then $\forall\ a > 0$,*

$$\mathbf{Pr}[X \geq a] \leq \frac{\mathbf{E}[X]}{a}.$$

Proof. Let's first deal with some extreme technicalities: If $\mathbf{Pr}[X \geq a] = 0$ then we are done (because $\mathbf{E}[X]/a \geq 0$, because $a > 0$ and clearly $\mathbf{E}[X] \geq 0$). Otherwise, it is legal to condition on the event $X \geq a$. Relatedly, if $\mathbf{Pr}[X < a] = 0$ then $X \geq a$ always, and hence $\mathbf{E}[X] \geq a$; thus there is nothing to prove, since the right-hand side is at least 1. So we can assume $\mathbf{Pr}[X < a] \neq 0$, and it is legal to condition on the event $X < a$.

With these technicalities out of the way, we have the following easy observation:

$$\begin{aligned} \mathbf{E}[X] &= \mathbf{Pr}[X \geq a] \cdot \mathbf{E}[X \mid X \geq a] + \mathbf{Pr}[X < a] \cdot \mathbf{E}[X \mid X < a] \\ &\geq \mathbf{Pr}[X \geq a] \cdot a + \mathbf{Pr}[X < a] \cdot 0, \end{aligned}$$

where we've used $\mathbf{E}[X \mid X \geq a] \geq a$, and $\mathbf{E}[X \mid X < a] \geq 0$ because X is always nonnegative. Thus we have

$$\mathbf{E}[X] \geq a\mathbf{Pr}[X \geq a] \qquad \Rightarrow \qquad \mathbf{Pr}[X \geq a] \leq \frac{\mathbf{E}[X]}{a}.$$

□

Pretty straightforward. We can also deduce:

[2] Not to be confused with the "Markov Brothers Inequality", which Markov proved with his brother. Also, don't confuse Andrey Andreyevich Markov with his son, also a mathematician called Andrey Andreyevich Markov. Markov *fils* lived from 1903 to 1979.

Corollary 2. *Let X be a nonnegative random variable, and let $t > 0$. Then*

$$\mathbf{Pr}\left[X \geq t \cdot \mathbf{E}[X]\right] \leq \frac{1}{t}.$$

The proof of this corollary is easy: just take $t = a/\mathbf{E}[X]$ and use Markov's Inequality.

I personally tend to remember/use the Corollary much more than the original statement. You can think of it as saying, "The probability a nonnegative r.v. is much bigger than its expectation is small." Note that it's pointless to use a $t < 1$ in the Corollary.

A good example of Markov's Inequality is when X represents the running time of a randomized algorithm (which is of course always nonnegative). For example, suppose you prove your algorithm's running time satisfies $\mathbf{E}[X] = \ln n$. As we've seen before, you can often prove such a statement while knowing extremely little about the PMF of X; the linearity + expectation method works wonders. Nevertheless, by Markov's Inequality you automatically know:

$$\mathbf{Pr}[X \geq 2 \ln n] \leq 1/2,$$

$$\mathbf{Pr}[X \geq 10 \ln n] \leq 1/10,$$

$$\mathbf{Pr}[X \geq 100 \ln n] \leq 1/100,$$

etc. So we *do* learn from $\mathbf{E}[X] = \ln n$ that it is pretty rare for the running time X to be far greater than $\ln n$. Now this is not a *great* bound, because often the behavior is even better than this. But Markov's Inequality has the merit that it applies in extremely general situations — all you need is that the random variable is nonnegative.

Let's do an example illustrating that Markov's Inequality is not always very useful. Suppose $X \sim \text{Binomial}(n, 1/2)$ (remember, this means X counts the number of heads in n fair coin flips). By Markov's Inequality,

$$\mathbf{Pr}\left[X \geq \frac{3}{4}n\right] \leq \frac{\mathbf{E}[X]}{\frac{3}{4}n} = \frac{(1/2)n}{(3/4)n} = \frac{2}{3}.$$

This is indeed true, but it's a pretty horrible bound, since it's obvious from symmetry that even $\mathbf{Pr}[X > \frac{1}{2}n] \leq \frac{1}{2}$.

3 Variance

Let X be a random variable. Let's denote its mean, $\mathbf{E}[X]$, by μ. (This is a very common convention, by the way. μ is chosen because μ is the Greek version of m, which is the first letter in "mean".) A natural question to ask is, "What is typical amount by which X deviates from its mean?" I.e., what is

$$\mathbf{E}[|X - \mu|]?$$

Interesting and natural though this question is, it turns out to be very hard to analyze and work with, in general. Weirdly enough, things become much simpler and cleaner if you ask about the *square* of the deviation from the mean...

Definition 3. *The* variance *of a random variable X, denoted* $\mathbf{Var}[X]$*, is*

$$\mathbf{Var}[X] = \mathbf{E}[(X - \mu)^2], \tag{2}$$

where $\mu = \mathbf{E}[X]$.

Here are some of the most important facts about the variance of a random variable:

- $\mathbf{Var}[X] \geq 0$ always. This is because the thing inside the expectation, $(X-\mu)^2$, is always nonnegative.
- $\mathbf{Var}[X] = 0$ if and only if X is a constant random variable. The "if" is easy to see: If X is always equal to some number c, then $\mu = c$ of course, and hence $(X-\mu)^2 = (c-c)^2 = 0$ always. As for the "only if" direction, if $\mathbf{E}[(X-\mu)^2] = 0$, then the thing inside the expectation has to *always* be 0 (otherwise, the expectation would be strictly positive). In other words, $X = \mu$ with probability 1; i.e., X is a constant random variable.
- If c is a constant, $\mathbf{Var}[cX] = c^2\mathbf{Var}[X]$. Note that it is *not* $c\mathbf{Var}[X]$; don't be fooled by this common fallacy!

Here is the easy proof of this last item:

Proof.

$$\begin{aligned}
\mathbf{Var}[aX] &= \mathbf{E}\left[(aX - \mathbf{E}[aX])^2\right] \\
&= \mathbf{E}\left[(aX - a\mathbf{E}[X])^2\right] \\
&= \mathbf{E}\left[(a(X - \mathbf{E}[X]))^2\right] \\
&= \mathbf{E}\left[a^2(X - \mathbf{E}[X])^2\right] \\
&= a^2\mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \quad = \quad a^2\mathbf{Var}[X].
\end{aligned}$$

□

We never (or at least, rarely) calculate the variance of a random variable directly from the defining equation (2). Instead, we almost always use the following:

Formula: $\mathbf{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$.

Proof.

$$\begin{aligned}
\mathbf{Var}[X] = \mathbf{E}[(X-\mu)^2] &= \mathbf{E}[X^2 - 2\mu X + \mu^2] \\
&= \mathbf{E}[X^2] - 2\mu\mathbf{E}[X] + \mu^2 \qquad \text{(linearity of expectation)} \\
&= \mathbf{E}[X^2] - 2\mu\cdot\mu + \mu^2 \\
&= \mathbf{E}[X^2] - \mu^2 \quad = \quad \mathbf{E}[X^2] - \mathbf{E}[X]^2.
\end{aligned}$$

□

Corollary 4. *If X is any random variable, $\mathbf{E}[X^2] \geq \mathbf{E}[X]^2$.*

Proof. $\mathbf{E}[X^2] - \mathbf{E}[X]^2 = \mathbf{Var}[X] \geq 0$. □

Let's do a simple example, for practice.

Example: Suppose you win \$100 with probability 1/10 and you win \$0 otherwise. Let X denote your winnings in dollars. What is $\mathbf{Var}[X]$?

Answer: First, $\mathbf{E}[X] = (1/10) \cdot 100 = 10$. Therefore, $\mathbf{E}[X]^2 = 100$. Next, $\mathbf{E}[X^2] = (1/10) \cdot (100)^2 = 1000$. Therefore, $\mathbf{Var}[X] = 1000 - 100 = 900$.

3.1 Standard Deviation

Suppose that, like a physicist, we decided to carry around units in the preceding calculation. Then $\mathbf{E}[X]^2 = 100\$^2$, and $\mathbf{E}[X^2] = 1000\$^2$, so $\mathbf{Var}[X] = 900\$^2$. So although variance indeed measures spread/dispersion of the random variable, the scale is, in a way, off. It would be more logical to measure the spread/dispersion in units, not squared-units. Also, there is something a little funny about the formula $\mathbf{Var}[aX] = a^2\mathbf{Var}[X]$. This observation motivates the following definition:

Definition 5. *The* standard deviation *of a random variable X, denoted* $\mathbf{stddev}[X]$*, is*

$$\mathbf{stddev}[X] = \sqrt{\mathbf{Var}[X]}.$$

Much like μ is usually used to denote the mean of x, it is quite traditional to denote the standard deviation by σ (again, because σ is Greek for s, the first letter in 'standard deviation'). We can therefore conclude that it is traditional to denote the variance by σ^2.

So in the above gambling example, $\mathbf{stddev}[X] = \sqrt{900} = 30$ (with units \$).

The standard deviation actually has a pretty reasonable name; it kind of *does* give a reasonable quantification of the amount by which a random variable deviates from its mean.

4 Chebyshev's Inequality

Pafnuty Lvovich Chebyshev (1821–1894) was also an interesting character (with a very unusual name — Pafnuty?). He kind of invented the notions of random variables and expected value. He also lived before Markov did, so it's funny that his inequality comes second in the natural progression.

If you're able to compute the standard deviation σ of a random variable, you can give a stronger bound than Markov can for the probability that a random variable deviates from its mean (again, "The More You Know"...):

Theorem 6. *Let X be a random variable (it need not be nonnegative) with mean μ and standard deviation $\sigma > 0$. Then for any $t > 0$,*

$$\mathbf{Pr}[|X - \mu| \geq t\sigma] \leq \frac{1}{t^2}.$$

You should remember this theorem in words:

The probability a r.v. is at least t standard deviations away from its mean is at most $1/t^2$.

For example,

$$\mathbf{Pr}[|X - \mu| \geq 6\sigma] \leq \frac{1}{36}.$$

If you know about the business management strategy called "Six Sigma", you may now call into question how much sense its name makes. (Hint: none. Read the Wikipedia entry for a laugh.)

The proof of Chebyshev's Inequality is easy: use Markov's Inequality!

Proof. We apply Markov's Inequality to the random variable $(X-\mu)^2$. (This is legal, because this random variable is indeed always nonnegative.)

$$\begin{aligned}\mathbf{Pr}[|X-\mu| \geq t\sigma] &= \mathbf{Pr}[|X-\mu|^2 \geq t^2\sigma^2] \\ &\leq \frac{\mathbf{E}[(X-\mu)^2]}{t^2\sigma^2} \\ &= \frac{\mathbf{Var}[X]}{t^2\mathbf{Var}[X]} = \frac{1}{t^2}.\end{aligned}$$

Here the first step is because "$|X-\mu| \geq t\sigma$" and "$|X-\mu|^2 \geq t^2\sigma^2$" are *identical events*, so of course they have the same probability. The second step is Markov. The last step is by definition of Variance. □

Here is an equivalent statement of Chebyshev's Inequality (you can check the equivalence yourself). This is how it's more usually stated, but it's almost always *used* in the above form, so that's why we stated it thus.

Also Chebyshev's Inequality: $\forall\ a > 0$,

$$\mathbf{Pr}[|X-\mu| \geq a] \leq \frac{\mathbf{Var}[X]}{a^2}.$$

5 More on variance, covariance, and an example

We will now build up to an example of Chebyshev's Inequality in action, with our random variable being a Binomial random variable. Of course, to use Chebyshev's Inequality, you need to know the variance. So indeed, let's think about computing the variance of a Binomial random variable. It's natural to start with the $n = 1$ case of a Binomial random variable — i.e., a Bernoulli random variable:

Formula: Let $X \sim \text{Bernoulli}(p)$. Then $\mathbf{Var}[X] = p(1-p)$.

Proof. $\mathbf{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2 = (p \cdot 1^2 + (1-p)\cdot 0^2) - p^2 = p - p^2 = p(1-p)$. □

It's pointless to use Chebyshev's Inequality on Bernoulli random variables because, well, you already know all there is to know about Bernoulli random variables — they're 1 with probability p, 0 with probability $1-p$. Case closed!

Binomials are sums of Bernoullis, so what's going on with the variance of the sum of two random variables? Something very much like what's going on with the expectation of the product of two random variables, that's what.

Fallacy: $\mathbf{Var}[X+Y] = \mathbf{Var}[X] + \mathbf{Var}[Y]$. ("Linearity of variance" is *false*.)

Theorem 7. If X *and* Y *are independent, then* $\mathbf{Var}[X+Y] = \mathbf{Var}[X] + \mathbf{Var}[Y]$.

Proof. We have the following straightforward computation, using linearity of expectation:

$$\begin{aligned}
\mathbf{Var}[X+Y] &= \mathbf{E}[(X+Y)^2] - (\mathbf{E}[X+Y])^2 \\
&= \mathbf{E}[X^2 + 2XY + Y^2] - (\mathbf{E}[X] + \mathbf{E}[Y])^2 \\
&= \mathbf{E}[X^2] + 2\mathbf{E}[XY] + \mathbf{E}[Y^2] - \mathbf{E}[X]^2 - 2\mathbf{E}[X]\mathbf{E}[Y] - \mathbf{E}[Y]^2 \\
&= \mathbf{E}[X^2] - \mathbf{E}[X]^2 \quad + \quad \mathbf{E}[Y^2] - \mathbf{E}[Y]^2 \quad + \quad 2(\mathbf{E}[XY] - \mathbf{E}[X]\mathbf{E}[Y]) \\
&= \mathbf{Var}[X] + \mathbf{Var}[Y] + 2(\mathbf{E}[XY] - \mathbf{E}[X]\mathbf{E}[Y]) \qquad (*)
\end{aligned}$$

But now the last term in (*) is 0, because $\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y]$ by independence of X and Y. □

You can also easily prove the following extension:

Theorem 8. *If $X_1, \ldots, X_n$ are independent,* $\mathbf{Var}[X_1 + \cdots + X_n] = \mathbf{Var}[X_1] + \cdots + \mathbf{Var}[X_n]$.

Fallacy: If X and Y are independent, $\mathbf{stddev}[X+Y] = \mathbf{stddev}[X] + \mathbf{stddev}[Y]$.

The correct statement (by definition) is of course:

Fact: If X and Y are independent, $\mathbf{stddev}[X+Y] = \sqrt{\mathbf{stddev}[X]^2 + \mathbf{stddev}[Y]^2}$.

5.1 Covariance

Based on line (*) in the above proof, we see that $\mathbf{E}[XY] - \mathbf{E}[X]\mathbf{E}[Y]$ sort of measures the extent to which "linearity of variance" fails, for non-independent r.v.'s.

Definition 9. *The* covariance *of two random variables, X and Y, denoted* $\mathbf{Cov}[X,Y]$*, is*

$$\mathbf{Cov}[X,Y] = \mathbf{E}[XY] - \mathbf{E}[X]\mathbf{E}[Y].$$

Exercise: Prove for yourself that we also have

$$\mathbf{Cov}[X,Y] = \mathbf{E}[(X - \mathbf{E}[X])(Y - \mathbf{E}[Y])].$$

Obvious Fact: If X and Y are independent then $\mathbf{Cov}[X,Y] = 0$.

Fallacy: If $\mathbf{Cov}[X,Y] = 0$ then X and Y are independent.

We'll see counterexamples for the fallacy on the homework.

5.2 Variance of a Binomial r.v.

Finally, let's get back to the variance of a Binomial r.v.

Formula: Let $X \sim \text{Binomial}(p)$. Then $\mathbf{Var}[X] = np(1-p)$.

Proof. We can write $X = X_1 + \cdots + X_n$, where the X_i's are independent Bernoulli(p) random variables. Thus we have $\mathbf{Var}[X] = \sum_{i=1}^{n} \mathbf{Var}[X_i] = np(1-p)$, where we used independence. □

We can now give an interesting example of Chebyshev's Inequality.

Question: Suppose we flip n independent fair coins. Upper-bound the probability that the number of heads is $\frac{3}{4}n$ or more.

Answer: Let X be the number of heads, so $X \sim \text{Binomial}(n, 1/2)$. We have $\mu = \mathbf{E}[X] = n/2$. We've just seen that $\sigma^2 = \mathbf{Var}[X] = n/4$. Hence $\sigma = \mathbf{stddev}[X] = \sqrt{n}/2$. Using $t = \sqrt{n}/2$ in Chebyshev's Inequality, we have

$$\mathbf{Pr}[|X - n/2| \geq n/4] = \mathbf{Pr}[|X - \mu| \geq (\sqrt{n}/2)(\sqrt{n}/2)] = \mathbf{Pr}[|X - \mu| \geq t\sigma] \leq 1/t^2 = 4/n.$$

Note that the event $|X - n/2| \geq n/4$ is equal to

$$(X \geq (3/4)n) \quad \cup \quad (X \leq (1/4)n).$$

The two events $X \geq (3/4)n$ and $X \leq (1/4)n$ are disjoint, and it's clear by symmetry that they have the same probability. We just showed that the sum of their probabilities is at most $4/n$, so the event $X \geq (3/4)n$ must have probability at most $2/n$.

This is a pretty good bound: it goes to 0 as n gets large. (Remember that Markov's Inequality only told us that this probability was at most 2/3!) For example, it says that the probability of getting at least 750 heads when you flip 1000 coins is at most .002.

But Chebyshev isn't always *so* strong: as we'll see in the next lecture, the true probability of getting 750 or more heads in 1000 flips is at most 10^{-54}. Indeed, it's roughly

$$.000000000000000000000000000000000000000000000000000000000067.$$

In other words, it'll never happen in anyone's lifetime. In the universe's lifetime, even.

15-359: Probability and Computing
Fall 2009

Lecture 9: The Chernoff Bound

1 Chernoff Bounds

Today you will learn the Chernoff Bound. Or rather, *a* Chernoff Bound. There are actually very many slightly different statements that people call "Chernoff Bounds" You will be required to memorize *two* such bounds. The first of these is a special case we will call:

Chernoff Bound 1: Let $X \sim \text{Binomial}(n, 1/2)$. Then for any $0 \leq t \leq \sqrt{n}$,

$$\begin{aligned} \mathbf{Pr}\left[X \geq \frac{n}{2} + t\frac{\sqrt{n}}{2}\right] &\leq e^{-t^2/2}, \\ \text{and also } \mathbf{Pr}\left[X \leq \frac{n}{2} - t\frac{\sqrt{n}}{2}\right] &\leq e^{-t^2/2}. \end{aligned}$$

We'll tell you

Chernoff Bound 2 in the next lecture.

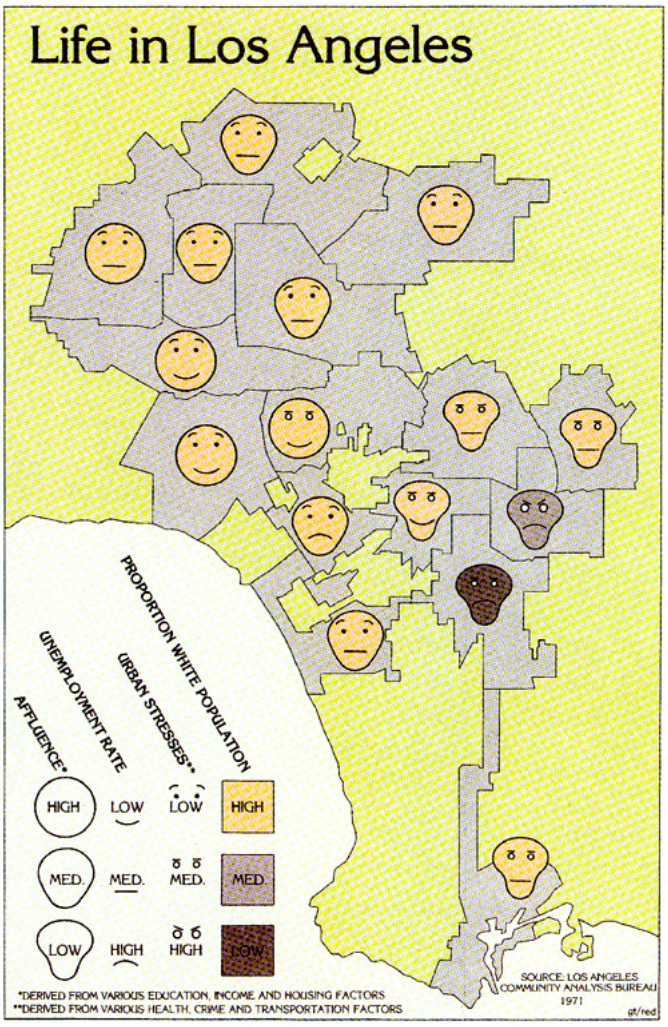
Chernoff Bounds extend the theme we saw in the last lecture: Trying to show that it is unlikely a random variable X is far away from its expectation. As we saw last class, you can make better and better statements to this effect the "more you know" about X. If you know nothing about X, you can't really say anything. If you know that X is nonnegative, Markov's Inequality tells you it's unlikely to be far bigger than its expectation. If you manage to compute $\mathbf{stddev}[X]$, Chebyshev's Inequality tells you the chance that X is t standard deviations or more from its expectation is at most $1/t^2$.

Chernoff Bounds can be used when you know that X is the sum of many ***independent*** *random variables.* The prototypical example of such an X is a $\text{Binomial}(n, 1/2)$ random variable, which is the sum of n independent Bernoulli random variables. As we'll see later in this class, whenever you add up many independent random variables, the sum acts like a "bell curve", and in particular is *extremely*, indeed *exponentially* unlikely to be t standard deviations away from its mean. We can see this in the statement of Chernoff Bound 1, where $\mathbf{E}[X] = \frac{n}{2}$ and $\mathbf{stddev}[X] = \frac{\sqrt{n}}{2}$, as we saw last time. Chernoff Bound 1 tells us that the probability that X is t standard deviations or more away from its mean is at most $2\exp(-t^2/2)$.[1]

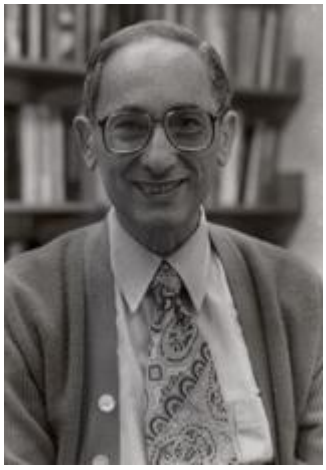
[1] By the way, we will sometimes use this "exp" notation: $\exp(x)$ means e^x, and is used whenever x is really complicated and therefore you'd rather not have to scrunch it up into a superscript.

1.1 Herman Chernoff

The Chernoff Bound is named after Herman Chernoff. You'd think that this extends the Russian theme of Markov and Chebyshev, but actually, Chernoff is American. He's a prof at Harvard. Oddly enough, he's perhaps better known now for his idea to use pictures of human faces to illustrate multivariate statistical data. Here's an example we found on the Internet:



Chernoff Faces



Chernoff's face

Actually, Chernoff wasn't even really the first to prove the Chernoff Bounds we'll see. Rather, Sergei Bernstein proved our "Chernoff Bounds" almost 30 years before Chernoff, in 1924. (Another Russian after all? Nope. Ukrainian.) Chernoff himself proved generalizations of Bernstein's work. But it seems the first computer scientists to use these bounds mistakenly attributed them to Chernoff rather than Bernstein. Subsequently, Chernoff Bounds got used like crazy, all the time, in computer science research — and the originally-bestowed name, "Chernoff Bounds", stuck.

2 Chebyshev bounds reviewed

Let $X \sim \text{Binomial}(n, 1/2)$. As we all remember/know by now, we have

$$\mathbf{E}[X] = n/2, \qquad \mathbf{Var}[X] = n/4 \qquad \Rightarrow \quad \mathbf{stddev}[X] = \sqrt{n}/2.$$

So Chebyshev's Inequality gives us, for example,

$$\mathbf{Pr}[|X - n/2| \geq 5\sqrt{n}] = \mathbf{Pr}[|X - \mathbf{E}[X]| \geq 10\mathbf{stddev}[X]] \leq 1/10^2 = 1/100$$

(here we used "t" equal to 10). In other words,

$$\mathbf{Pr}\left[\frac{n}{2} - 5\sqrt{n} \leq X \leq \frac{n}{2} + 5\sqrt{n}\right] \geq .99.$$

So, "most" of the time, a Binomial$(n, 1/2)$ random variable is within a few factors of $\sqrt{n}$ of its mean.

A few standard deviations is considered to be a "small deviation" from the mean. What about a large deviation? As we saw last time using Chebyshev's Inequality and $t = \sqrt{n}/2$ we have

$$\mathbf{Pr}\left[X \geq \frac{3}{4}n\right] \leq \frac{2}{n}.$$

This is indeed true, but it's actually far from being sharp. For example, it tells us that in 1000 coin flips ($X \sim$ Binomial$(1000, 1/2)$), the probability of getting at least 750 heads is at most 0.2%. But in fact, using Maple we calculated

$$\mathbf{Pr}[X \geq 750] = \sum_{u=750}^{1000} p_X(u) = \sum_{u=750}^{1000} \binom{1000}{u} 2^{-1000} \approx .67 \times 10^{-58}.$$

So 0.2% is extraordinarily far from the truth! So it behooves us to figure out a better bound, exploiting the fact that X is not just any old random variable — it's the sum of many independent random variables.

Chernoff Bound 1, given above, is just such a bound. By taking $n = 1000$ and $t = \sqrt{1000}/2$ in it, we get

$$\mathbf{Pr}[\text{Binomial}(1000, 1/2) \geq 750] \leq e^{-(\sqrt{1000}/2)^2/2} = e^{-1000/8} = .52 \times 10^{-54}.$$

That's more like it!

3 Intuition for the proof

For the rest of this lecture, we will see the proof of Chernoff Bound 1. This will be one of the hardest proofs we do in this class. Also, we are going to cheat *very* slightly near the very end; not in a fundamental way, just in a way to save some time by avoiding some painful arithmetic. In this (long) section of the notes, we will give intuition for the proof. The formal proof will come in the next section of the notes.

In fact, we just have to prove the first bound in the statement of Chernoff Bound 1. The other bound follows immediately by the symmetry of X; i.e., the fact that

$$p_X(u) = p_X(n - u) \qquad \text{for all } u = 0 \ldots n.$$

In a way, the proof is a lot like the proof of Chebyshev's Inequality — you take X, consider a *different* random variable based on X, and then apply Markov's Inequality to that random variable.

3.1 Switching to Rademachers

So let's begin with our random variable X,

$$X = X_1 + \cdots + X_n,$$

where the X_i's are independent Bernoulli(1/2) random variables, equally likely to be 0 or 1. Actually, since what we care about is the deviation of X from its mean, it's a bit nicer if this mean is 0, rather than $n/2$. Let's define

$$Y_i = -1 + 2X_i,$$

so Y_i is -1 with probability 1/2 and Y_i is $+1$ with probability 1/2. (These Bernoulli-like random variables are called *Rademacher* random variables, named after the German mathematician Hans Rademacher who taught at UPenn for most of his life.) Define

$$Y = Y_1 + \cdots + Y_n = (-1 + 2X_1) + \cdots + (-1 + 2X_n) = -n + 2X.$$

This is just a shifted version of X which is more convenient to work with. The key point is that the event we're interested in is:

$$X \geq n/2 + t\sqrt{n}/2 \qquad \Leftrightarrow \qquad -n + 2X \geq -n + 2(n/2 + t\sqrt{n}/2) = t\sqrt{n}.$$

We haven't done anything fundamental here, we've just changed the problem to upper-bounding

$$\mathbf{Pr}[Y \geq t\sqrt{n}],$$

where

$$Y = Y_1 + \cdots + Y_n, \qquad \text{where } Y_i\text{'s are independently } \pm 1 \text{ with prob. } 50\% \text{ each.}$$

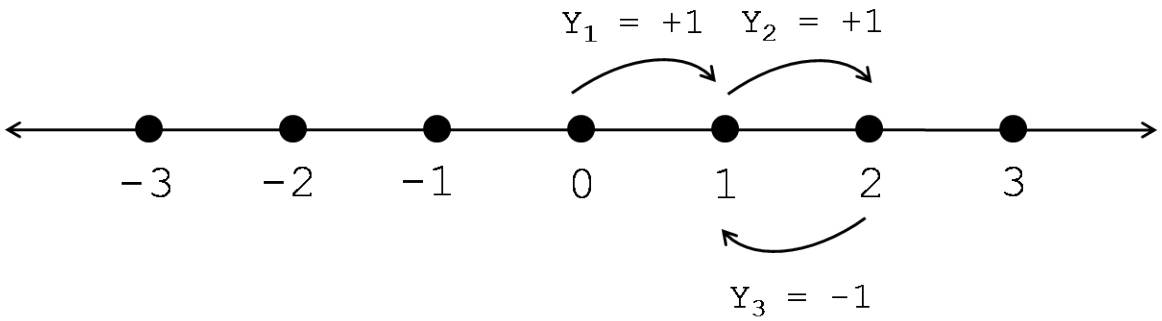
Notice the event "$Y \geq t\sqrt{n}$" is again the event that Y is at least t standard deviations above its mean: this is because it's easy to see

$$\mathbf{E}[Y_i] = 0 \quad \Rightarrow \quad \mathbf{E}[Y] = 0;$$

$$\mathbf{Var}[Y_i] = [(1/2) \cdot 1^2 + (1/2) \cdot (-1)^2] - 0^2 = 1 \quad \Rightarrow \quad \mathbf{Var}[Y] = n \quad \Rightarrow \quad \mathbf{stddev}[Y] = \sqrt{n}.$$

3.2 Intuition

Now that we've switched to adding up these Y_i's, let's reflect a little about what's going on. As we add up Y_1, Y_2, Y_3, ..., it's like we're taking a "random walk" on the integers.



We start at 0, and at each "step" we go either 1 unit to the right or 1 to the right, at the flip of a fair coin. The position we end up in after n steps gives the random variable Y. We're trying to bound the probability that Y ends up very high. Note that Y is *not* a nonnegative random variable, so we can't use Markov's Inequality.

Idea: Let's take our inspiration from one of the top mathematicians of our time:

"Near 1*, multiplication is the same as addition."*

— Fields Medalist Terence Tao, 2007

What Tao meant was the following:

"If λ_1 *and* λ_2 *are tiny numbers (positive or negative), then*

$$(1+\lambda_1)\times(1+\lambda_2)\approx 1+(\lambda_1+\lambda_2)."$$

This is simply because

$$(1+\lambda_1)(1+\lambda_2)=1+\lambda_1+\lambda_2+\lambda_1\lambda_2,$$

and if λ_1 and λ_2 are tiny, then $\lambda_1\lambda_2$ is *supertiny*, and thus, perhaps, negligible. Another way to see the validity of this maxim is to use The Most Useful Approximation Ever:

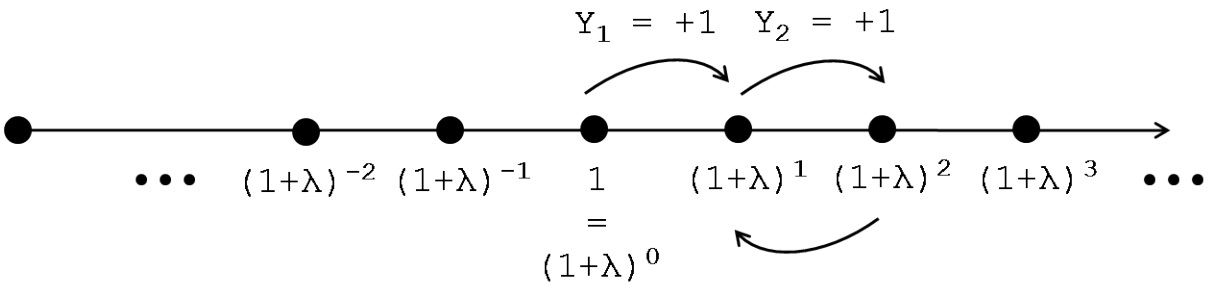
$$e^x\approx 1+x \qquad \text{if } x \text{ is tiny.}$$

Thus, using the approximation twice,

$$(1+\lambda_1)(1+\lambda_2)\approx e^{\lambda_1}e^{\lambda_2}=e^{\lambda_1+\lambda_2}\approx 1+(\lambda_1+\lambda_2).$$

3.3 The multiplicative random walk

Instead of starting at 0 and adding ± 1 at each step in the random walk, let's start at 1 and multiply/divide by $1+\lambda$ at each step. (Think of λ as a tiny number to be named later.)



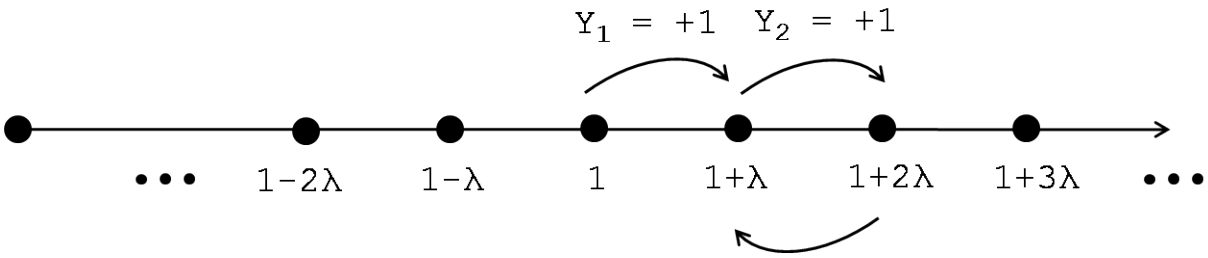
So now, it's like being at position $(1+\lambda)^u$ in this "multiplicative" random walk is equivalent to being at position u in the old "additive" random walk. Note that the final position of the multiplicative random walk is precisely

$$1\cdot(1+\lambda)^{Y_1}(1+\lambda)^{Y_2}\cdots(1+\lambda)^{Y_n}=(1+\lambda)^{Y_1+\cdots+Y_n}=(1+\lambda)^{Y}.$$

By Tao's maxim (equivalent, our approximation $e^x\approx 1+x$) we know that

$$(1+\lambda)^u\approx 1+u\lambda,$$

at least, as long as $u\lambda$ is small. I.e., if we zoom in, our walk kind of looks like this:



But this approximation breaks down once $u\lambda$ is no longer "small"; say, when $u\lambda$ is around 1. In fact, it's starts to go extremely haywire once $u\lambda$ gets much bigger than 1.

We know the "standard deviation" of Y is $\sqrt{n}$, so we expect that it's reasonably likely Y will end up around $\sqrt{n}$. Therefore, let's consider choosing λ so that the "haywire" point is right around $u = \sqrt{n}$; i.e., let's consider choosing $\lambda = 1/\sqrt{n}$.

(This is all still intuition for the proof, by the way.)

3.4 The haywire point

If we choose $\lambda = 1/\sqrt{n}$, then the final location of the multiplicative random walk is

$$(1 + 1/\sqrt{n})^{Y_1}(1 + 1/\sqrt{n})^{Y_2} \cdots (1 + 1/\sqrt{n})^{Y_n} = (1 + 1/\sqrt{n})^{Y}.$$

If Y itself ends up being, say, $\sqrt{n}$, this is

$$(1 + 1/\sqrt{n})^{\sqrt{n}} \approx (e^{1/\sqrt{n}})^{\sqrt{n}} = e.$$

So when Y ends up at this relatively reasonable random, the multiplicative random walk ends up near the number e, which is still reasonably in the vicinity of 1. Similarly, if Y ends up at, say, $-\sqrt{n}$, the multiplicative random walk ends up at

$$(1 + 1/\sqrt{n})^{-\sqrt{n}} \approx (e^{1/\sqrt{n}})^{-\sqrt{n}} = e^{-1},$$

also reasonably in the vicinity of 1.

But, if Y ends up somewhat bigger than this — say, at $5\sqrt{n}$ — that is equivalent to the multiplicative random walk ending up at

$$(1 + 1/\sqrt{n})^{5\sqrt{n}} \approx e^5 \approx 150.$$

And if Y were to end up at, say, $100\sqrt{n}$, that would be equivalent to the multiplicative random walk ending up at e^{100}, an unimaginably huge number.

This is the key for the whole proof — *Markov's Inequality* can upper-bound the probability that a random variable ends up unimaginably huge.

4 The formal* proof

(We put an asterisk by "formal" because, as mentioned, we'll cheat very slightly at one point.)

Let $Y = Y_1 + \cdots + Y_n$ be the sum of n independent Rademacher random variables (each is ± 1 with probability $1/2$ each). Our goal is to upper-bound the probability that Y is at least t standard deviations about its mean, $\mathbf{Pr}[Y \geq t\sqrt{n}]$.

Define $Z_i = (1+\lambda)^{Y_i}$, where λ is a small positive number we'll define later. Of course,

$$Z_i = \begin{cases} 1+\lambda & \text{with probability } 1/2, \\ \frac{1}{1+\lambda} & \text{with probability } 1/2. \end{cases}$$

Since the $Y_1, \ldots, Y_n$ are independent, we also have that $Z_1, \ldots, Z_n$ are independent. Define

$$Z = Z_1 Z_2 \cdots Z_n,$$

which is the final position of the "multiplicative random walk". All of the Z_i's are nonnegative random variables, and hence Z is also always nonnegative. Thus, we can use Markov's Inequality on it. To do this, we will need to calculate the expectation of Z. We have

$$\mathbf{E}[Z] = \mathbf{E}[Z_1 Z_2 \cdots Z_n] = \mathbf{E}[Z_1]\mathbf{E}[Z_2] \cdots \mathbf{E}[Z_n], \tag{1}$$

using the fact that the Z_i's are independent. As for the expectation of the Z_i's, we have

$$\mathbf{E}[Z_i] = (1/2)(1+\lambda) + (1/2)\frac{1}{1+\lambda} = \frac{(1+\lambda)^2+1}{2(1+\lambda)} = \frac{2+2\lambda+\lambda^2}{2+2\lambda} = 1 + \frac{\lambda^2}{2+2\lambda} \leq 1 + \frac{\lambda^2}{2}. \tag{2}$$

(The mean of Z_i is very very slightly bigger than 1, meaning that the "multiplicative random walk" "drifts" slightly to the right.) Substituting this into (1) yields

$$\mathbf{E}[Z] \leq \left(1 + \frac{\lambda^2}{2}\right)^n. \tag{3}$$

Now we use Markov. The event we are interested in is

$$Y \geq t\sqrt{n} \qquad \Leftrightarrow \qquad (1+\lambda)^Y \geq (1+\lambda)^{t\sqrt{n}} \qquad \Leftrightarrow \qquad Z \geq (1+\lambda)^{t\sqrt{n}}.$$

So

$$\mathbf{Pr}[Y \geq t\sqrt{n}] = \mathbf{Pr}[Z \geq (1+\lambda)^{t\sqrt{n}}] \leq \frac{\mathbf{E}[Z]}{(1+\lambda)^{t\sqrt{n}}},$$

using Markov's Inequality. Substituting (3), we get

$$\mathbf{Pr}[Y \geq t\sqrt{n}] \leq \frac{(1+\lambda^2/2)^n}{(1+\lambda)^{t\sqrt{n}}}.$$

So far we haven't actually specified what λ will be. We can take it to whatever is our best advantage. Before we thought about taking $\lambda = 1/\sqrt{n}$; turns out it's a bit better to take it to be $t/\sqrt{n}$. So continuing the above, taking $\lambda = t/\sqrt{n}$:

$$\mathbf{Pr}[Y \geq t\sqrt{n}] \leq \frac{(1+t^2/(2n))^n}{(1+t/\sqrt{n})^{t\sqrt{n}}}. \tag{4}$$

Now we do a slight cheat, by using The Most Useful Approximation Ever, $1 + x \approx e^x$. Now $1 + x \leq e^x$ for all x, so it's completely valid to use this approximation in the numerator of the above, but it's invalid to use it in the denominator. Well, let's do it anyway. We promise, you can make this proof correct.[2] So using our favorite approximation in (4) we get

$$\begin{aligned} \mathbf{Pr}[Y \geq t\sqrt{n}] &\leq \frac{(e^{t^2/2n})^n}{(e^{t\sqrt{n}})^{t\sqrt{n}}} \\ &= \frac{e^{t^2/2}}{e^{t^2}} \\ &= e^{-t^2/2}, \end{aligned}$$

which is what is claimed in Chernoff Bound 1.

[2]The trick is to keep around the exact expression from (2), and to choose $\lambda = e^{t/\sqrt{n}} - 1$, rather than $\lambda = t/\sqrt{n}$.

15-359: Probability and Computing
Fall 2009

Lecture 10: More Chernoff Bounds, Sampling, and the Chernoff + Union Bound method

1 Chernoff Bound 2

Last lecture we saw the following:

Chernoff Bound 1: Let $X \sim \text{Binomial}(n, 1/2)$. Then for any $0 \le t \le \sqrt{n}$,

$$\begin{aligned} \mathbf{Pr}\left[X \ge \frac{n}{2} + t\frac{\sqrt{n}}{2}\right] &\le e^{-t^2/2}, \\ \text{and also } \mathbf{Pr}\left[X \le \frac{n}{2} - t\frac{\sqrt{n}}{2}\right] &\le e^{-t^2/2}. \end{aligned}$$

This bound tells us that if X is the sum of many independent Bernoulli(1/2)'s, it's extremely unlikely that X will deviate even a little bit from its mean. Let's rephrase the above a little. Taking $t = \epsilon\sqrt{n}$ in Chernoff Bound 1, we get

$$\left.\begin{array}{l} \mathbf{Pr}[X \ge (1+\epsilon)(n/2)] \\ \mathbf{Pr}[X \le (1-\epsilon)(n/2)] \end{array}\right\} \le e^{-\epsilon^2 n/2}.$$

Okay, that tells us about deviations for Binomial$(n, 1/2)$. Turns out that similar bounds are true for Binomial(n, p). And similar bounds are true for, say,

$$U = U_1 + \cdots + U_n$$

when the U_i's are *independent* random variables that are

$$U_i = \begin{cases} 0 & \text{w.p. } 1/3, \\ 1/2 & \text{w.p. } 1/3, \\ 1 & \text{w.p. } 1/3. \end{cases}$$

Actually, there are a dozens of versions of Chernoff bounds, covering dozens of variations; in other texts, you might see any of them. But as promised, you will only be required to memorize one more :) This one covers 95% of cases pretty well.

Chernoff Bound 2:
Let $X_1, \ldots, X_n$ be *independent* random variables.
They need not have the same distribution.
Assume that $0 \le X_i \le 1$ always, for each i.
Let $X = X_1 + \cdots + X_n$.
Write $\mu = \mathbf{E}[X] = \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_n]$.

Then for any $\epsilon \geq 0$,

$$\begin{aligned} \mathbf{Pr}[X \geq (1+\epsilon)\mu] &\leq \exp\left(-\frac{\epsilon^2}{2+\epsilon}\mu\right) \\ \text{and, } \mathbf{Pr}[X \leq (1-\epsilon)\mu] &\leq \exp\left(-\frac{\epsilon^2}{2}\mu\right). \end{aligned}$$

(Remember the notation $\exp(x) = e^x$.)

1.1 What's up with the $\frac{\epsilon^2}{2+\epsilon}$?

Chernoff Bound 2 takes some memorizing, we admit it. But it's a truly indispensable tool, so it's very much worth it. Actually, the second statement, the probability of going below the expectation, isn't *so* bad to remember; it's clean-looking at least. The first statement is, admittedly, slightly weird. By symmetry you were probably hoping that we were going to write

$$\exp\left(-\frac{\epsilon^2}{2}\mu\right)$$

on the right side of the first statement, like in the second statement. Unfortunately we can't; turns out, that's simply not true. Notice that

$$\frac{\epsilon^2}{2} \text{ is slightly bigger than } \frac{\epsilon^2}{2+\epsilon},$$

and therefore

$$-\frac{\epsilon^2}{2}\mu \text{ is slightly more negative than } -\frac{\epsilon^2}{2+\epsilon}\mu$$

(since $\mu \geq 0$ since all the X_i's are nonnegative), and therefore

$$\exp\left(-\frac{\epsilon^2}{2}\mu\right) \text{ is slightly smaller than } \exp\left(-\frac{\epsilon^2}{2+\epsilon}\mu\right).$$

So the bound for $\mathbf{Pr}[X \geq (1+\epsilon)\mu]$ is slightly bigger than the bound for $\mathbf{Pr}[X \leq (1-\epsilon)\mu]$, but unfortunately, it cannot be improved to the smaller quantity.

Now, ϵ is a user-selected parameter, and *most* of the time you, the user, select ϵ to be pretty small, smaller than 1. If indeed $\epsilon \leq 1$, we have

$$\frac{\epsilon^2}{2+\epsilon} \text{ is bigger than } \frac{\epsilon^2}{3},$$

and so we could put this slightly simpler quantity into our bound. Indeed, sometimes people state a simplified Chernoff Bound 2 like this:

Chernoff Bound 2′: Suppose we are in the setting of Chernoff Bound 2. Then for all $0 \leq \epsilon \leq 1$,

$$\begin{aligned} \mathbf{Pr}[X \geq (1+\epsilon)\mu] &\leq \exp\left(-\frac{\epsilon^2}{3}\mu\right) \\ \text{and, } \mathbf{Pr}[X \leq (1-\epsilon)\mu] &\leq \exp\left(-\frac{\epsilon^2}{2}\mu\right). \end{aligned}$$

That's a little easier to remember. And indeed, for the event $X \leq (1-\epsilon)\mu$ you would never care to take $\epsilon > 1$ anyway! But for the event $X \geq (1+\epsilon)\mu$ sometimes you *would* care to take $\epsilon > 1$. And Chernoff Bound 2′ doesn't really tell you what to do in this case. Whereas Chernoff Bound 2 does; for example, taking $\epsilon = 8$, it tells you

$$\mathbf{Pr}[X \geq 9\mu] \leq \exp(-6.4\mu).$$

1.2 More tricks and observations

Sometimes you simply want to upper-bound the probability that X is *far* from its expectation. For this, one thing we can do is take Chernoff Bound 2, intentionally *weaken* the second bound to $\exp(-\frac{\epsilon^2}{2+\epsilon}\mu)$ as well, and then add, concluding:

Two-sided Chernoff Bound: In the setting of Chernoff Bound 2,

$$\mathbf{Pr}[|X-\mu| \geq \epsilon\mu] \leq 2\exp\left(-\frac{\epsilon^2}{2+\epsilon}\mu\right).$$

What if we don't have $0 \leq X_i \leq 1$? Another twist is that sometimes you don't have that your r.v.'s X_i satisfy $0 \leq X_i \leq 1$. Sometimes they might satisfy, say, $0 \leq X_i \leq 10$. Don't panic! In this case, the trick is to define $Y_i = X_i/10$, and $Y = Y_1 + \cdots + Y_n$. Then we *do* have that $0 \leq Y_i \leq 1$ always, the Y_i's are independent assuming the X_i's are, $\mu_Y = \mathbf{E}[Y] = \mathbf{E}[X]/10 = \mu_X/10$, and we can use the Chernoff Bound on the Y_i's;

$$\begin{aligned}\mathbf{Pr}[X \geq (1+\epsilon)\mu_X] = \mathbf{Pr}[X/10 \geq (1+\epsilon)\mu_X/10] &= \mathbf{Pr}[Y \geq (1+\epsilon)\mu_Y] \\ &\leq \exp\left(-\frac{\epsilon^2}{2+\epsilon}\mu_Y\right) = \exp\left(-\frac{\epsilon^2}{2+\epsilon}\frac{\mu_X}{10}\right).\end{aligned}$$

So you lose a factor of 10 inside the final exponential probability bound, but you still get something pretty good. This is a key trick to remember!

Finally, one more point: Given that we have this super-general Chernoff Bound 2, why bother remembering Chernoff Bound 1? The reason is, Chernoff 2 is weaker than Chernoff 1. If $X \sim \text{Binomial}(n, 1/2)$, Chernoff Bound 1 tells us

$$\mathbf{Pr}[X \leq n/2 - t\sqrt{n}/2] \leq \exp(-t^2/2). \tag{1}$$

However, suppose we used Chernoff Bound 2. We have

$$\text{“}X \leq n/2 - t\sqrt{n}/2\text{”} \qquad \Leftrightarrow \qquad X \leq (1 - t/\sqrt{n})(n/2),$$

and so Chernoff Bound 2 gives us

$$\mathbf{Pr}[X \leq n/2 - t\sqrt{n}/2] \leq \exp\left(-\frac{(t/\sqrt{n})^2}{2} \cdot \frac{n}{2}\right) = \exp(-t^2/4).$$

This is only the *square-root* of the bound (1).

1.3 The proof

We will not prove Chernoff Bound 2. However the proof is not much more than an elaboration on our proof of Chernoff Bound 1. Again, the idea is to let $\lambda > 0$ be a small "scale", pretty close to ϵ, actually. You then consider the random variable $(1+\lambda)^X$, or relatedly, $e^{\lambda X}$. Finally, you bound, e.g.,

$$\mathbf{Pr}[X \geq (1+\epsilon)\mu] = \mathbf{Pr}[e^{\lambda X} \geq e^{(1+\epsilon)\mu}] \leq \frac{\mathbf{E}[e^{\lambda X}]}{e^{(1+\epsilon)\mu}},$$

using Markov's Inequality, and you can compute $\mathbf{E}[e^{\lambda X}]$ as

$$\mathbf{E}[e^{\lambda X}] = \mathbf{E}[e^{\lambda X_1 + \cdots + \lambda X_n}] = \mathbf{E}[e^{\lambda X_1} \cdots e^{\lambda X_n}] = \mathbf{E}[e^{\lambda X_1}] \cdots \mathbf{E}[e^{\lambda X_n}],$$

using the fact that $X_1, \ldots, X_n$ are independent.

2 Sampling

The #1 use of the Chernoff Bound is probably in Sampling/Polling. Suppose you want to know what fraction of the population approves of the current president. What do you do?

Well, you do a poll. Roughly speaking, you call up n random people and ask them if they approve of the president. Then you take this empirical fraction of people and claim that's a good estimate of the true fraction of the entire population that approves of the president. But is it a good estimate? And how big should n be?

Actually, there are *two* sources of error in this process. There's the probability that you obtain a "good" estimate. And there's the extent to which your estimate is "good". Take a close look at all those poll results that come out these days and you'll find that they use phrases like,

"This poll is accurate to within $\pm 2\%$*,* 19 *times out of* 20*."*

What this means is that they did a poll, they published an estimate of the true fraction of people supporting the president, and they make the following claim about their estimate: There is a $1/20$ chance that their estimate is just completely way off. Otherwise, i.e., with probability $19/20 = 95\%$, their estimate is within $\pm 2\%$ of the truth.

This whole 95% thing is called the "confidence" of the estimate, and its presence is inevitable. There's just no way you can legitimately say, "My polling estimate is 100% guaranteed to be within $\pm 2\%$ of the truth." Because if you sample n people at random, you know, there's a chance they all happen to live in Massachusetts, say (albeit an unlikely, much-less-than-5% chance), in which case your approval rating estimate for a Democratic president is going to much higher than the overall country-wide truth.

To borrow a phrase from Learning Theory, these polling numbers are "Probably Approximately Correct" — i.e., probably (with chance at least 95% over the choice of people), the empirical average is approximately (within $\pm 2\%$, say) correct (vis-a-vis the true fraction of the population).

2.1 Analysis

How do pollsters, and how can we, make such statements?

Let the true fraction of the population that approves of the president be p, a number in the range $0 \leq p \leq 1$. This is the "correct answer" that we are trying to elicit.

Suppose we ask n uniformly randomly chosen people for their opinion, and let each person be chosen *independently*. We are choosing people "*with* replacement". (I.e., it's possible, albeit a very slim chance, that we may ask the same person more than once.) Let X_i be the indicator random variable that the ith person we ask approves of the president. Here is the key observation:

Fact: $X_i \sim \text{Bernoulli}(p)$, and $X_1, \ldots, X_n$ are independent.

Let $X = X_1 + \cdots + X_n$, and let $\overline{X} = X/n$. The empirical fraction $\overline{X}$ is the estimate we will publish, our guess at p.

Question: How large does n have to be so that we get good "accuracy" with high "confidence"? More precisely, suppose our pollster boss wants our estimate to have accuracy θ and confidence $1 - \delta$, meaning

$$\mathbf{Pr}\left[|\overline{X} - p| \leq \theta\right] \geq 1 - \delta.$$

How large do we have to make n?

Answer: Let's start by using the Two-sided Chernoff Bound on X. Since $X \sim \text{Binomial}(n, p)$, we have $\mathbf{E}[X] = np$. So for any $\epsilon \geq 0$, we have

$$\begin{aligned} \mathbf{Pr}[|X - pn| \geq \epsilon pn] &\leq 2\exp\left(-\frac{\epsilon^2}{2+\epsilon} \cdot pn\right) \\ \Leftrightarrow \mathbf{Pr}[|\overline{X} - p| \geq \epsilon p] &\leq 2\exp\left(-\frac{\epsilon^2}{2+\epsilon} \cdot pn\right). \end{aligned}$$

Here the two events inside the $\mathbf{Pr}[\cdot]$ are the same event; we just divided by n.

We want accuracy θ; i.e., we want $\overline{X}$ to be within θ of p with high probability. (In our original example, $\theta = 2\% = .02$.) We need to get $\theta = \epsilon p$, so we should take $\epsilon = \theta/p$.[1] Doing so, we get

$$\mathbf{Pr}[|\overline{X} - p| \geq \theta] \quad \leq \quad 2\exp\left(-\frac{\theta^2/p^2}{2+\theta/p} \cdot pn\right) \quad = \quad 2\exp\left(-\frac{\theta^2}{2p+\theta} \cdot n\right).$$

Okay, what about getting confidence $1 - \delta$? Let's look at that bound on the right. Having that n inside the $\exp(-\cdot)$ is great — it tells us the bigger n is, the less chance that our estimate is off by more than θ. As for the $\frac{\theta^2}{2p+\theta}$, well, the bigger that term is, the better. The bigger p is, the smaller that factor is, but the biggest p could be is 1. I.e.,

$$\frac{\theta^2}{2p+\theta} \geq \frac{\theta^2}{2+\theta},$$

and therefore we have

$$\mathbf{Pr}[|\overline{X} - p| \geq \theta] \quad \leq \quad 2\exp\left(-\frac{\theta^2}{2+\theta} \cdot n\right).$$

[1] Worried about $p = 0$? In that case, $\overline{X}$ will correctly be 0 100% of the time!

So if we want confidence $1-\delta$ in the estimate (think, e.g., $\delta = 1/20$), we would like the right-hand side in the above to be at most δ.

$$\begin{aligned} \delta \geq 2\exp\left(-\frac{\theta^2}{2+\theta}\cdot n\right) \quad &\Leftrightarrow \quad \exp\left(\frac{\theta^2}{2+\theta}n\right) \geq \frac{2}{\delta} \\ &\Leftrightarrow \quad \frac{\theta^2}{2+\theta}n \geq \ln\frac{2}{\delta} \\ &\Leftrightarrow \quad n \geq \frac{2+\theta}{\theta^2}\ln\frac{2}{\delta}. \end{aligned}$$

We have thus proved the following very important theorem. (NB: As is traditional, we've called the accuracy "ϵ" in the below, rather than "θ".)

Sampling Theorem: Suppose we use independent, uniformly random samples to estimate p, the fraction of a population with some property. If the number of samples n we use satisfies

$$n \geq \frac{2+\epsilon}{\epsilon^2}\ln\frac{2}{\delta},$$

then we can assert that our estimate $\overline{X}$ satisfies

$$\overline{X} \in [p-\epsilon, p+\epsilon] \text{ with probability at least } 1-\delta.$$

Some comments:

- That range $[p-\epsilon, p+\epsilon]$ is sometimes called the *confidence interval*.
- Due to the slightly complicated statement of the bound, sometimes people will just write the slightly worse bounds
$$n \geq \frac{3}{\epsilon^2}\ln\frac{2}{\delta},$$
or even
$$n \geq O\left(\frac{1}{\epsilon^2}\ln\frac{2}{\delta}\right).$$
- One beauty of the Sampling Theorem is that the number of samples n you need *does not depend on the size of the total population*. In other words, it doesn't matter how big the country is, the number of samples you need to get a certain accuracy and a certain confidence only depends on that accuracy and confidence.
- In the example we talked about earlier we were interested in accuracy $\epsilon = 2\%$ and confidence 95%, meaning $\delta = 1/20$. So the Sampling Theorem tells us we need at least
$$n \geq \frac{2+.02}{(.02)^2}\ln\frac{2}{1/20} = 5050\ln 40 \approx 18600.$$
Not so bad: you only need to call 18600 or so folks! Er, well, actually, you need to get 18600 folks to respond. And you need to make sure that the events "person responds" and "person approves of the president" are independent. (Hmm... maybe being a pollster is not as easy as it sounds...)
- As you can see from the form of the bound in the Sampling Theorem, the really costly thing is getting high accuracy: $1/\epsilon^2$ is a fairly high price to have to pay in the number of samples. On the other hand, getting really high confidence is really cheap: because of the ln, it hardly costs anything to get δ really tiny.

3 The Chernoff + Union Bound method

Just as Linearity of Expectation and Indicator random variables often come together in glorious marriage, so too do the Chernoff Bound and the Union Bound. You remember the humble Union Bound, right?

Union Bound: $\mathbf{Pr}[B_1 \cup B_2 \cup \cdots \cup B_n] \leq \sum_{i=1}^{n} \mathbf{Pr}[B_i]$.

The idea behind the Chernoff + Union Bound method is the following: The Chernoff Bound is extraordinarily strong, usually showing that the probability a certain "bad event" happens is extremely tiny. Thus, even if very many different bad events exist, if you bound each one's probability by something extremely tiny, you can afford to just add up the probabilities. I.e.,

$$\mathbf{Pr}[\text{anything bad at all}] = \mathbf{Pr}[\text{Bad}_1 \cup \cdots \cup \text{Bad}_{\text{Large}}] \leq \sum_{i=1}^{\text{Large}} \mathbf{Pr}[\text{Bad}_i]$$

(this is a high-level picture :) and if the Chernoff Bound implies $\mathbf{Pr}[\text{Bad}_i] \leq$ minuscule for each i, we get

$$\mathbf{Pr}[\text{anything bad at all}] \leq \sum_{i=1}^{\text{Large}} \text{minuscule} = (\text{Large}) \times (\text{minuscule}) = \text{small}.$$

(Told you it was high-level.)

Let's do an example to make this concrete.

3.1 Random load balancing

Suppose you are a content delivery network — say, YouTube. Suppose that in a typical five-minute time period, you get a million content requests, and each needs to be served from one of your, say, 1000 servers. How should you distribute the requests (let's call them 'jobs') across your servers to balance the load? You might consider a round-robin policy, or a policy wherein you send each job to the server with the lowest load. But each of these requires maintaining some state and/or statistics, which might cause slight delays. You might instead consider the following extremely simple and lightweight policy, which is surprisingly effective: assign each job to a random server.

Let's abstract things slightly. Suppose we have k servers and n jobs. Assume all n jobs arrive very quickly, we assign each to a random server (independently), and the jobs take a while to process. What we are interested in the *load* of the servers.

ASSUMPTION: n is much bigger than k.

E.g., our YouTube example had $n = 10^6$, $k = 10^3$.

Question: The average "load" — jobs per server — will of course be n/k. But how close to perfectly balanced will things be? In particular, is it true that the *maximum* load is not much bigger than n/k, with high probability?

Answer: Yes! Let's do the analysis.

Let X_i denote the number of jobs assigned to server i, for $1 \leq i \leq k$.

Question: What is the distribution of the random variable X_i?

Answer: If you think a little bit carefully, you see that X_i is a binomial random variable: $X_i \sim \text{Binomial}(n, 1/k)$. To see it, just imagine staring at the ith server. For each of n trials/jobs, there is a $1/k$ chance that that job gets thrown onto this ith server.

Don't be confused by notation, by the way — we used to use subscripted X's like X_i to denote Bernoulli random variables, and their sums were Binomial random variables denoted X. Here we have that each X_i itself is a Binomial random variable.

Question: Are $X_1, \ldots, X_k$ independent random variables?

Answer: No! Here is one non-rigorous but intuitive reason: we *know* that it will always be the case that

$$\sum_{i=1}^{k} X_i = n.$$

So in particular, if I tell you the value of $X_1, \ldots, X_{k-1}$, you know exactly what X_k is. Bear in mind that this reasoning does not *formally* prove that $X_1, \ldots, X_k$ are not independent. But it's not hard to do that either. Here's one way: if $X_1, \ldots, X_k$ *were* independent, we'd have

$$\mathbf{Pr}[X_1 = n \cap X_2 = n \cap \cdots \cap X_k = n] = \mathbf{Pr}[X_1 = n] \cdot \mathbf{Pr}[X_2 = n] \cdots \mathbf{Pr}[X_k = n].$$

Now the left-hand side above is 0, since there's no way each server can have all n jobs! But the right-hand side is *nonzero*; each $\mathbf{Pr}[X_i = n] = (1/k)^n$.

But you know what? It's cool. We're going to be using the Union Bound, and one beauty of the Union Bound is that (like Linearity of Expectation), it does not care whether the events it involves are independent or not.

The average load after doing random load balancing is clearly n/k, just because there are n total jobs and k servers. Further, we have

$$\mathbf{E}[X_i] = n/k$$

for each i (by the formula $\mathbf{E}[\text{Binomial}(n, p)] = np$, e.g.). So, as is intuitive, each server is expected to have n/k jobs. But we are interested in the *maximum* load among the k servers...

Let

$$M = \max(X_1, X_2, \ldots, X_k),$$

a random variable representing the maximum load. Our goal is to show a statement of the form

$$\mathbf{Pr}[M \geq n/k + \text{c}] \leq \text{small}$$

where c is not too large. Then we'll be able to say, "With high probability, the maximum load is at most $n/k + c$." We'll let you in on a secret: we're eventually going to be able to take

$$c = 3\sqrt{\ln k}\sqrt{n/k}.$$

You might say to yourself, "uh, is that good?" Yeah, it's pretty good, actually. The $3\sqrt{\ln k}$ part is really quite small, and the rest of the deviation, $\sqrt{n/k}$, is only the square-root of the average load. So, in a concrete setting with $n = 10^6$ and $k = 10^3$, the average load is

$$n/k = 1000,$$

and our deviation here is

$$c = 3\sqrt{\ln 1000}\sqrt{1000} \approx 250.$$

So our result will show, "With high probability, the *maximum* load is at most 1250." We'll see what "high probability" means next lecture.

15-359: Probability and Computing
Fall 2009

Lecture 11: Load balancing concluded, Balls and Bins, Poisson r.v.'s

1 Chernoff + Union Bound Method: Load balancing concluded

In the last lecture we discussed the "Chernoff + Union Bound Method", and in this lecture we'll see a very typical use of it. Remember the "load balancing" problem from last time. There are n jobs that need to be assigned to k servers. We assume that $n \gg k$, with a typical example being $n = 10^6$, $k = 10^3$. We use the *random* load balancing policy: each job is independently assigned to one of the k servers uniformly at random.

Let X_i denote the number of jobs assigned to server i, for $i = 1 \ldots k$. Recall that $X_i \sim \text{Binomial}(n, 1/k)$. Since we always have n jobs on k servers, the *average load* will be exactly n/k. What about the *maximum load*?

Let $M = \max(X_1, X_2, \ldots, X_k)$, a random variable. We will show that with high probability, M is not much bigger than n/k.

Theorem 1.

$$\mathbf{Pr}[M \geq n/k + 3\sqrt{\ln k}\sqrt{n/k}] \leq 1/k^2.$$

In the specific case of $n = 10^6$, $k = 10^3$, this translates into

$$\mathbf{Pr}[M \geq 1000 + 249.34] \leq 10^{-6};$$

i.e., the maximum load will be less than 1250 except with probability one in a million. A pretty comforting result! By the way, since each X_i is Bernoulli$(n, 1/k)$, we have

$$\mathbf{Var}[X_i] = n(1/k)(1 - 1/k) \approx n/k \quad \Rightarrow \quad \mathbf{stddev}[X_i] \approx \sqrt{n/k}.$$

So even for one *particular* server we kind of expect the number of jobs to fluctuate from the mean, n/k, by a few factors of $\sqrt{n/k}$. Indeed, if we just applied Chebyshev to a particular server, we'd only conclude

$$\mathbf{Pr}[X_i \geq n/k + 3\sqrt{\ln k}] \leq 1/(3\sqrt{\ln k})^2 = 1/(9 \ln k),$$

and $1/(9 \ln k) \approx 1.6\%$ when $k = 10^3$. So Chebyshev can only tell us that the probability, say, server #7 has less than 1250 jobs is at most 1.6%. The Chernoff + Union Bound Method tells us that *all* 1000 *servers* have less than 1250 jobs except with probability one in a million!

Proof. Let $c = 3\sqrt{\ln k}\sqrt{n/k}$. (We selected this quantity with considerable foresight! Indeed, when you solve problems like this, you'll need to let c remain a variable, and then select it at the very end.)

As promised, we use the Chernoff + Union Bound method. The first step when doing this is to decide what the "bad events" are. Here it's bad whenever $X_i \geq n/k + c$. So let's define k "bad events", $B_1, \ldots, B_k$, where

$$B_i = \text{“}X_i \geq n/k + c\text{”}.$$

(This is a lot of bad events, by the way. But that's the beauty of the Chernoff + Union Bound Method; you can overcome even 1000's of ways of things going wrong!) Let's further define

$$B = B_1 \cup B_2 \cup \cdots \cup B_k,$$

our overall "bad event". This is the event that *at least* one of the k servers gets load at least $n/k+c$. This is exactly the event we care about. I.e.,

$$B = \text{“}M \geq n/k + c\text{”} = \text{“}m \geq n/k + 3\sqrt{\ln k}\sqrt{n/k}\text{”}$$

Our goal is to show that B occurs with probability at most $1/k^2$. We will use the Union Bound for this, telling us that

$$\mathbf{Pr}[B] \leq \sum_{i=1}^{k} \mathbf{Pr}[B_i].$$

Now if you think for a second, you'll see that all these probability $\mathbf{Pr}[B_i]$ are the same, since the situation is the same with respect to all the servers. We will show

$$\mathbf{Pr}[B_1] \leq 1/k^3.$$

Thus $\mathbf{Pr}[B_i] \leq 1/k^3$ for all i, and so

$$\mathbf{Pr}[M \geq n/k + c] = \mathbf{Pr}[B] \leq \sum_{i=1}^{k} \mathbf{Pr}[B_i] \leq \sum_{i=1}^{k} (1/k^3) = k/k^3 = 1/k^2.$$

So indeed it remains to show $\mathbf{Pr}[B_1] \leq 1/k^3$. For this we use the Chernoff Bound.

Specifically, we use Chenoff Bound 2. For this we need to be analyzing a random variable which is the sum of many independent random variables. Why are we in this situation? Remember, $X_1 \sim \text{Bernoulli}(n, 1/k)$. In particular, we could write

$$X_1 = I_1 + I_2 + \cdots + I_n,$$

where I_j is the indicator random variable that job j gets assigned to server #1. The I_j's are independent, and they are Bernoulli$(1/k)$ which means they are between 0 and 1 as required by Chernoff. We have

$$\mu = \mathbf{E}[X_1] = n/k.$$

Hence, for any $\epsilon > 0$, the Chernoff Bound implies

$$\mathbf{Pr}[X_1 \geq (1+\epsilon)(n/k)] \leq \exp\left(-\frac{\epsilon^2}{2+\epsilon}\mu\right). \tag{1}$$

We are concerned with the event $X_1 \geq n/k + c$, so we need to take ϵ such that

$$\epsilon(n/k) = c = 3\sqrt{\ln k}\sqrt{n/k} \quad \Leftrightarrow \quad \epsilon = 3\sqrt{\ln k}/\sqrt{n/k}.$$

We're assuming that $n \gg k$, which should imply $\epsilon \ll 1$. Specifically, let's assume[1] that $n \geq 9k \ln k$, which you can check implies $\epsilon \leq 1$. So

$$\frac{\epsilon^2}{2+\epsilon} \geq \frac{\epsilon^2}{3} = \frac{(3\sqrt{\ln k}/\sqrt{n/k})^2}{3} = \frac{9 \ln k}{3(n/k)} = \frac{3 \ln k}{\mu}.$$

Substituting this into (1) implies

$$\mathbf{Pr}[X_1 \geq n/k + c] \leq \exp\left(-\frac{3 \ln k}{\mu}\mu\right) = e^{-3 \ln k} = k^{-3} = 1/k^3,$$

as desired. □

2 Balls & Bins

Balls.

• • • • • • • • — n of them

Bins.

⊔ ⊔ ⊔ ⊔ ⊔ ⊔ ⊔ — m of them

Each ball is independently "thrown" into a uniformly random bin.

This is the scenario we mean when we say "Balls and Bins" — and say it we will.

2.1 Examples

Balls and Bins is one of the most commonly discussed and most important paradigms in all of probability. And computing. It's the model for hundreds of situations and the source of hundreds of interesting problems. Here are some example situations it models:

- **Load balancing:** Balls = jobs, Bins = servers. Now Balls and Bins models the random load balancing scenario we just discussed.
- **Data storage:** Balls = files, Bins = disks.
- **Hashing:** Balls = data keys, Bins = hash table slots. We will discuss this application in detail next lecture.
- **Routing:** Balls = connectivity requirements, Bins = paths in a network.
- **Beloved probability chestnuts:**

  a) Balls = students, Bins = birthdays ⇒ the Birthday Problem.

  b) Balls = purchased coupons, Bins = coupon types

  ⇒ the "Coupon Collector Problem" (AKA "RetailMeNot" problem on the homework)

And here are some example questions one is often interested in:

- **Question:** What is the probability of a *collision* — i.e., getting more than 1 ball in some bin? This is exactly what we're interested in in the Birthday Problem.

[1] This should really go into the hypothesis of the theorem.

- **Question:** How many balls are thrown before each bin has at least 1 ball? This is exactly what we're interested in in the Coupon Collector Problem.
- **Question:** How many balls end up in the bin with maximum load? This is what we just finished analyzing in the discussion of Randomized Load Balancing.

2.2 How to analyze — rules of thumb

Let X_i be the random variable counting the number of balls in the ith bin, $1 \leq i \leq m$. Just as we saw last before,

$$X_i \sim \text{Binomial}(n, 1/m).$$

This is because each of the n balls has a $1/m$ chance of being thrown into the ith bin. As we noted last time,

$$X_1, X_2, \ldots, X_m \text{ are } \textit{not} \text{ independent random variables.}$$

We of course have $\mathbf{E}[X_i] = n/m$ for each i. We will give a name to this:

Definition 2. *The* average load *in a Balls and Bins scenario is*

$$\lambda = \frac{n}{m}.$$

How do we analyze the various questions mentioned above? In some sense, we know everything is to know; we know the distribution of each X_i and in some sense we know how they are jointly distributed. However, the situation is a bit complicated, and to get a handle on it you often have to use various approximations and bounding inequalities. Here is the key *rule of thumb*:

Rule of Thumb:
a) If $n \gg m$, i.e., λ is huge, then it's a good idea to do analysis with Chernoff Bounds.
b) If $n \approx m$, i.e., λ is on the order of a "constant", Chernoff Bounds are not so powerful — instead, use the *Poisson Approximation*.

We'll discuss what exactly the Poisson Approximation is shortly. Just to say a few more words here: It's not like in part (b) that the Chernoff Bounds are *wrong*. It's just that they tend not to give very powerful statements — i.e., bounds that are close to being sharp. The reason is that if you apply Chernoff Bound 2 to some X_i, you get a bound like

$$\exp\left(-\frac{\epsilon^2}{2+\epsilon} \cdot \mu\right),$$

say, and here $\mu = \lambda$. If λ is very large, this is a very strong (i.e., small) upper-bound. But if λ is more like a constant... well, again, it's not that it's wrong, it's just that it's not very powerful.

3 Approximating the number of balls in a bin

Let's start by investigating a few simple questions in the Balls and Bins scenario. Remember, we have n balls, m bins, the average load is $\lambda = n/m$, and X_i denotes the number of balls landing in bin i. Our main assumption for now is:

ASSUMPTION: The number of bins, m, is "large". Also, the number of balls n is "large" but comparable to m: i.e., λ is "constant".

Using this assumption, let's try to get a handle on the distribution of the number of balls in the ith bin. You might protest that we're already done: $X_i \sim \text{Binomial}(n, 1/m)$, we can write down the PMF of X_i, and there's not much more to know. True, but Binomial random variables can be somewhat complicated to work with (think about their PMFs). Also, they act in a bit of a strange way when the "p" parameter is really close to 0, and this is the case since our "p" is $1/m$. Let's see what happens when we use the Most Useful Approximation Ever,

$$e^x \approx 1 + x \text{ when } x \text{ is tiny.}$$

Question: What is the probability the ith bin ends up empty?

Answer: We want to know $\mathbf{Pr}[X_i = 0]$. Since we know $X_i \sim \text{Binomial}(n, 1/m)$ we can simply write this down from the PMF of X_i. The answer is $(1 - 1/m)^n$. This should be clear: each of n balls has to evade the ith bin (which happens with probability $1 - 1/m$ for each ball).

Question: How can we approximate $\mathbf{Pr}[X_i = 0]$?

Answer: We use the Most Useful Approximation Ever with $x = -1/m$; this is valid as we are assuming m is large.

$$\mathbf{Pr}[X_i = 0] = (1 - 1/m)^n \approx (e^{-1/m})^n = e^{-n/m} = e^{-\lambda}.$$

Question: What is the expected number of empty bins, precisely and approximately?

Answer: Let U_i be the indicator random variable for the ith bin being empty. $U = \sum_{i=1}^{m} U_i$ denotes the number of empty bins. By linearity of expectation,

$$\mathbf{E}[U] = \sum_{i=1}^{m} \mathbf{E}[U_i] = \sum_{i=1}^{m} \mathbf{Pr}[X_i = 0] = m(1 - 1/m)^n \approx e^{-\lambda} m.$$

For example, if there are an equal number of balls and bins, $n = m$, then $\lambda = 1$, and we expect about a $1/e$ fraction of bins to be empty.

Question: What is the probability that the ith bin contains exactly 1 ball?

Answer: We want to know $\mathbf{Pr}[X_i = 1]$, and again, we can just write this down from the PMF:

$$\mathbf{Pr}[X_i = 1] = n \cdot (1/m) \cdot (1 - 1/m)^{n-1}.$$

Recall, this is because there are n choices for which will be the lone ball in bin i, this ball has to go into the bin (probability $1/m$) and the remaining $n - 1$ balls have to miss the bin (probability $1 - 1/m$ each).

Question: How can we approximate $\mathbf{Pr}[X_i = 1]$?

Answer: Rewrite the above as

$$\mathbf{Pr}[X_i = 1] = (n/m) \cdot (1 - 1/m)^n \cdot (1 - 1/m)^{-1}.$$

As before we use $(1-1/m)^n \approx e^{-n/m}$. We also use $1-1/m \approx 1$, which is valid since we're assuming m large. So we have

$$\mathbf{Pr}[X_i = 1] \approx \lambda e^{-\lambda}.$$

Amusingly, from this we see that if $n = m$ and so $\lambda = 1$, then $\mathbf{Pr}[X_i = 1] \approx 1/e$ again, and we expect about a $1/e$ fraction of bins to have exactly 1 ball.

Question: What is the probability that the ith bin contains exactly 2 balls? What is it approximately?

Answer: Again, we start by simply writing the Binomial PMF for 2:

$$\begin{aligned}
\mathbf{Pr}[X_i = 2] &= \binom{n}{2}(1/m)^2(1-1/m)^{n-2} \\
&= \frac{n(n-1)}{2!} \cdot \frac{1}{m} \cdot \frac{1}{m} \quad \cdot \quad (1-1/m)^{n-2} \\
&= \frac{1}{2!} \cdot \frac{n}{m} \cdot \frac{n-1}{m} \quad \cdot \quad (1-1/m)^n(1-1/m)^{-2} \\
&= \frac{1}{2!} \cdot \frac{n}{m} \cdot \frac{n}{m}\left(\frac{n-1}{n}\right) \quad \cdot \quad (1-1/m)^n(1-1/m)^{-2}
\end{aligned}$$

Now we use $\lambda = n/m$, the same approximations as before, and also

$$\frac{n-1}{n} = 1 - \frac{1}{n} \approx 1.$$

Hence

$$\mathbf{Pr}[X_i = 2] \approx \frac{1}{2!}\lambda^2 \cdot 1 \cdot e^{-\lambda} \cdot 1^{-2} = \frac{e^{-\lambda}\lambda^2}{2!}.$$

Hmm. A little more complicated, yes, but think about an example, say, $n = 2000$, $m = 1000$: Would you rather try to compute

$$\binom{2000}{2}(1/1000)^2(1-1/1000)^{1998} \quad \text{or} \quad \frac{e^{-2}2^2}{2!} \ ?$$

The approximation is excellent, by the way; the true quantity on the left is .2708, the quantity on the right is .2707.

Let's do one more:

Question: What is the probability that the ith bin contains exactly 3 balls? What is it approximately?

Answer: Using the same tricks and approximations:

$$\begin{aligned}
\mathbf{Pr}[X_i = 3] &= \binom{n}{3}(1/m)^3(1-1/m)^{n-3} \\
&= \frac{n(n-1)(n-2)}{3!} \cdot \frac{1}{m} \cdot \frac{1}{m} \cdot \frac{1}{m} \quad \cdot \quad (1-1/m)^{n-2} \\
&= \frac{1}{3!} \cdot \frac{n}{m} \cdot \frac{n-1}{m} \cdot \frac{n-2}{m} \quad \cdot \quad (1-1/m)^{n}(1-1/m)^{-3} \\
&= \frac{1}{3!} \cdot \frac{n}{m} \cdot \frac{n}{m}\left(1-\frac{1}{n}\right) \cdot \frac{n}{m}\left(1-\frac{2}{n}\right) \quad \cdot \quad (1-1/m)^{n}(1-1/m)^{-3} \\
&\approx \frac{1}{3!}\lambda \cdot \lambda \cdot 1 \cdot \lambda \cdot 1 \cdot e^{-\lambda} \cdot 1^{-3} \\
&= \frac{e^{-\lambda}\lambda^3}{3!}.
\end{aligned}$$

Again, when $n = 2000$, $m = 1000$, the truth and approximation are

$$\binom{2000}{3}(1/1000)^3(1-1/1000)^{1997} \approx .1805,$$

$$\frac{e^{-2}2^3}{3!} \approx .1804.$$

The approximation is pretty good, as you can see, and way easier to compute.

If you continue the above reasoning, you'll find the following:

Summary: Assuming $n/m = \lambda$ and that $n, m \gg r$, then

$$\mathbf{Pr}[X_i = r] = \mathbf{Pr}[\text{Binomial}(n, 1/m) = r] \approx \frac{e^{-\lambda}\lambda^r}{r!}.$$

4 The Poisson Distribution

Something a little funny is going on. We know the true PMF of X_i:

$$\mathbf{Pr}[X_i = r] = p_{X_i}(r) = \binom{n}{r}(1/m)^r(1-1/m)^{n-r}, \qquad r = 0, 1, 2, \ldots, n.$$

However, if r is a smallish natural number, n and m are "large", and $\lambda = n/m$ is "constant", we just decided that

$$\mathbf{Pr}[X_i = r] \approx \frac{e^{-\lambda}\lambda^r}{r!}.$$

And check this out:

$$\begin{aligned}
\sum_{r=0}^{\infty} \frac{e^{-\lambda}\lambda^r}{r!} &= e^{-\lambda}\left(1 + \lambda + \frac{\lambda^2}{2!} + \frac{\lambda^3}{3!} + \cdots\right) \\
&= e^{-\lambda}e^{\lambda} \qquad \text{(Taylor expansion/definition of } e^{\lambda}) \\
&= 1.
\end{aligned}$$

Whoa! What an amazing miracle! We have a sequence of nonnegative numbers, $e^{-\lambda}\lambda^r/r!$, one for each natural number r, and we just showed that they add up to 1. **Therefore, they form a valid PMF!** Let's give the definition:

Definition 3. *For any positive real number λ, random variable Y is said to have the "Poisson(λ)" distribution if its PMF is*

$$p_Y(r) = \frac{e^{-\lambda}\lambda^r}{r!} \qquad r = 0, 1, 2, 3, \ldots.$$

There's a lot to say about this definition!

- This is a prime instance of *introducing a random variable by giving its PMF.*
- In a sense we've said, "Forget about the Balls and Bins for a second. Consider this random variable Y."
- What we have seen is that in a Balls and Bins scenario with large m and n and average load λ, the distribution on the number of balls in a particular bin is *well-approximated* by a Poisson(λ) random variable Y.
- *There is no natural experiment (randomized code) that generates a Poisson random variable.*
- That's not to say that you can't generate a Poisson random variable with randomized code (think about how you might do it...).
- A Poisson random variable takes values 0, 1, 2, 3, ...
- The distribution is named after the French mathematician Siméon Denis Poisson (1781–1840). He invented it in an article called, no joke, "Research on the Probability of Judgments in Criminal and Civil Matters".[2]

4.1 The Poisson Approximation

The following is the "Poisson Approximation": Suppose $X \sim \text{Binomial}(n, 1/m)$, where n is "large" and $n/m = \lambda$ is "constant". Then for each $r \in \mathbb{N}$, we have

$$\mathbf{Pr}[X = r] \approx \frac{e^{-\lambda}\lambda^r}{r!}$$

is a good approximation. More precisely,

$$\lim_{\substack{n,m\to\infty \\ n/m=\lambda}} \mathbf{Pr}[X = r] = \frac{e^{-\lambda}\lambda^r}{r!}.$$

More generally, if $X \sim \text{Binomial}(n, p)$, n is "large" and np is "constant", we have that $Y \sim \text{Poisson}(np)$ is a good approximation to X, in the above sense.

There's an extension to this. In the Balls and Bins scenario, where you have random variables $X_1, \ldots, X_m$ counting the number of balls in each bin, the Poisson Approximation says that pretending they are *independent* Poisson(λ) random variables is a close approximation of the truth. We don't want you to use this fact at will, though, because it's "less true" than what we've said above. There are some rigorous theorems partially justifying it, but we don't want to get into them. We thought we'd let you in on the secret, though.

[2] Well, actually, "Recherches sur la probabilité des jugements en matière criminelle et en matière civile".

5 Basics of Poisson r.v.'s

As always when introducing a new family of distributions, we should figure out their basic properties. The first thing is always the PMF of Y, but we know this already — indeed, it's built into the definition. Next thing is always expectation:

Theorem 4. *Let $Y \sim$ Poisson(λ). Then $\mathbf{E}[Y] = \lambda$.*

Intuition: Y approximates $X_i \sim \text{Binomial}(n, 1/m)$ with $n/m = \lambda$. Since $\mathbf{E}[X_i] = n/m = \lambda$, it makes sense that $\mathbf{E}[Y] = \lambda$.

Proof. Formally,

$$\begin{aligned}
\mathbf{E}[Y] &= \sum_{r=0}^{\infty} \frac{e^{-\lambda}\lambda^r}{r!} \cdot r \\
&= e^{-\lambda}\left(\lambda + \frac{2\lambda^2}{2!} + \frac{3\lambda^3}{3!} + \frac{4\lambda^4}{4!} + \cdots\right) \\
&= e^{-\lambda}\left(\lambda + \frac{\lambda^2}{1!} + \frac{\lambda^3}{2!} + \frac{\lambda^4}{3!} + \cdots\right) \\
&= e^{-\lambda}\lambda\left(1 + \lambda + \frac{\lambda^2}{2!} + \frac{\lambda^3}{3!} + \cdots\right) \\
&= e^{-\lambda}\lambda e^{\lambda} \\
&= \lambda.
\end{aligned}$$

□

Next thing is always variance:

Theorem 5. *Let $Y \sim$ Poisson(λ). Then $\mathbf{Var}[Y] = \lambda$.*

Intuition: Again, Y approximates $X_i \sim \text{Binomial}(n, 1/m)$ with $n/m = \lambda$. Since $\mathbf{Var}[X_i] = n(1/m)(1 - 1/m) = \lambda(1 - 1/m) \approx \lambda$, it makes sense that $\mathbf{Var}[Y] = \lambda$.

Proof. On the homework :) □

5.1 Poissons are additive

Finally, we have the following important result:

Theorem 6. *Let $Y \sim$ Poisson(λ), let $Z \sim$ Poisson(λ'), and assume Y and Z are independent. Then the random variable $Y + Z$ has the distribution Poisson$(\lambda + \lambda')$.*

Intuition: Y approximates the number of balls in bin 1 in a Balls and Bins process with m bins, λm balls. Z approximates the number of balls in bin 1 in an independent Balls and Bins process with m bins, $\lambda' m$ balls. Now just imagine you use the same bins for both processes! So $Y + Z$ approximates the number of balls in bin 1 when there are m bins and $(\lambda + \lambda')m$ balls — i.e., it ought to be (approximately) distributed like Poisson$(\lambda + \lambda')$.

Proof. Again, on the homework :) □

6 "Law of Rare Events"

Poisson(λ) random variables might look a little artificial at first — from the definition, it seems like we just plunked down a strange sequence of numbers that happened to add up to 1. However, they actually occur quite frequently "in nature". Here are some things which empirically have a Poisson distribution, according to the Internet:

- Per-year deaths by horse/mule kick in the late-19th-century Prussian army.
- Number of white blood cells in a blood suspension.
- Typos per page in printed books.
- Number of imperfections per square-inch in a piece of metal.
- Number of roadkill per mile on a highway.
- Number of bomb hits per .25km^2 in South London during WWII.
- Geiger counter blips per minute when measuring a meteorite.

(Don't know why the examples have so much negativity; sorry about that :)

The general idea for all of these is that if you have a large number n of objects that get distributed randomly over some stretch of time/space, the number you see in a particular $1/m$ substretch is probably/possibly distributed like Binomial$(n, 1/m)$. And if things are scaled so that the average number per substretch is some constant λ, then it's reasonable to think that the number in a particular substretch has a Poisson(λ) distribution.

Take, for example, the number of typos per page in a book. Suppose you are editing a 200 page book; you proofread the manuscript super-carefully and find 500 total typos. So the average number of typos per page is 2.5. Given this information, it's reasonable to think that the typos are distributed randomly, and therefore the number of typos on a particular page may well be distributed like a Poisson(2.5) random variable.

By way of illustration, here is part of the PMF of a Poisson(2.5) random variable:

u	0	1	2	3	4	5	6	7	8	9	10
$p_Y(u)$	.0820	.2052	.2565	.2137	.1336	.0668	.0278	.0099	.0031	.0008	.0002

15-359: Probability and Computing
Fall 2009

Lecture 12: Hashing, Fingerprinting, Bloom Filters

1 Hash Tables

The Hash Table is one of the greatest data structures. Hash Tables are "dictionary", or "associative array", data structures that *can* have constant lookup time. ("Can" here is the key word; there are sometimes caveats.) Remember that a "dictionary" data structure is just something you use to store n data items, indexing them by a "key". We all know various balanced binary search tree data structures (Red-Black trees, AVL trees, etc.) which support lookup in $O(\log n)$ time. But with appropriate use of Hash Tables, you can have $O(1)$ time lookup and insertion.

1.1 Example — unacceptable passwords

One of the more common methods for password cracking is to just try each password on a list of common passwords. This is (coincidentally) called the "Dictionary Attack". It's actually pretty ridiculous how well it works. To guard against it, a lot of password systems prevent you from using such "easy" passwords in the first place.

Assume we are trying to implement such a system and we have a list of n unacceptable passwords. When a user goes to change their password, or create a new one, we need to check their proposed password against our list of unacceptable ones. How should we implement this?

Idea 1: Store the list in a sorted array. In this case we can do a lookup with binary search. This will take $O(\log n)$ time.

Idea 2: Store the list in a "trie". This is a popular data structure solution for the case of a dictionary of strings. We will not discuss it, but see, e.g., Wikipedia for a nice description. The main downside of a "trie" is that for, say, 32-byte passwords, a lookup will involve accessing up to 32 random memory locations (which can be bad if access to memory/disk is slow).

Idea 3: Use a hash table. For this we assume we have a "hash function" h, mapping strings into numbers,

$$h : \{\text{strings}\} \to \{1, 2, \ldots, m\}$$

which is somehow "random-seeming". More on that later.

A "hash table" is then just an array with m cells. For each unacceptable password x, our application will store x in the cell with index $h(x)$. For example, `"password"` is a really bad password, hence it would probably be on our list. Our hash function h might have $h($`"password"`$) = 132$; so then it stores this string in cell 132 of the table.

Problem: Collisions. This is an obvious problem. If we're trying to conserve space, we would probably want to make m not too large (compared to n). But then there's a chance that two passwords on our list might "hash" to the same value. E.g., perhaps by chance $h(\texttt{"secret"}) = 132$ as well.

Solution: Actually, there are many solutions to this problem. For now we'll just consider what's probably the simplest possible solution. It is called *chaining*. It just means that each cell of the hash table actually stores a linked list. With chaining, the hash operations are implemented like this:

Insert(x)**:**

- Compute $h(x)$.
- Insert x at the head of the linked list stored in cell $h(x)$.

Lookup(x)**:**

- Compute $h(x)$.
- Sequentially search for x in the linked list in cell $h(x)$.

How long do these operations take? Well, we tend to make the following assumptions:

Assumptions:

- Computing $h(x)$ takes $O(1)$ time. This is a pretty reasonable assumption; most hash functions used in practice do some kind of basic bit-level arithmetic, and output a single "word" (4 bytes, say). We're assuming also here that x itself is of constant size.
- Accessing a given location in an array takes $O(1)$ time. This is a standard assumption for random access machines.

With these assumptions it's clear that doing an Insert takes $O(1)$ time. As for doing a Lookup, the time required depends on how long the linked list in the cell indexed by $h(x)$ is. The time required is

$$O(1) + O(\text{load of that cell}).$$

Naturally, then, we really hope that all cells have small load!

2 "Random-seeming" hash functions, and SUHA

Before we get into hash table analysis, we need to clarify what we meant by saying that the hash function is "random-seeming". Specifically, we want two things out of a hash function: for each x, $h(x)$ should act like a uniformly random number in $\{1, 2, \ldots, m\}$. And, $h(x)$ should be easy to compute — "constant time". Here are two things you could try:

Try 1: Make h a truly random function. Specifically, for each possible string x, choose $h(x) \leftarrow \texttt{RandInt}(m)$. Of course, this makes all analysis based on h being random perfectly correct. However, it's completely infeasible for practice, since it requires storing in advance an integer for every possible string x!

Try 2: Some ad-hoc, randommy, scrambly, xor-this, mod-that function. For example, here's how python hashes a string:

```
h = ord(x[0]) << 7
for char in x:
  h = c_mul(1000003, h) ^ ord(char)
h = h ^ len(x)
```

This kind of thing has the advantage of being fast to compute. But the hash values really aren't random. Maybe they're "random enough", but you have to worry about what happens if some evil person is choosing the strings to be hashed... Such a person might try to attack your program and make it run slowly by giving it a bunch of strings that all hash to the same value.

Hmm. What to do? Well, the thing most people to is just cheat :) We'll cheat too and use the "Simple Uniform Hashing Assumption":

Simple Uniform Hashing Assumption (SUHA): Do "Try 2", but analyze it as though it was Try 1.

There is a great deal of work done by researchers on how to construct hash functions that are efficient to evaluate but have sufficient "random-like" behavior. There's also some recent rigorous work on why the SUHA is actually essentially valid under certain assumptions about "real-world data". We may say more about that on the blog. But for now, let's just assume SUHA :)

By the way, there's an intermediate "Try 3" you might think about — that's to use a cryptographic hash function like SHA-1. We feel very strongly that SHA-1 acts almost indistinguishably from a truly random function, so that's good. But we still don't use this solution very much because computing SHA-1 is actually fairly slow. And in many uses of hash tables — e.g., in routers — we really require blazing speed for the computation of the hash function.

2.1 Using SUHA

Under SUHA, the analysis of hash tables is just Balls and Bins! Say you insert n keys into a hash table of size m. Under SUHA, each key hashes to a uniformly random cell. So balls = keys and bins = cells.

Let's go to the password example. Assume we've stored all n unacceptable passwords in a hash table of size m. Now we can think about the *expected* time of looking up a user's proposed new password.

First, let's think about an "unsuccessful find" (meaning the user's password is actually acceptable). Remember, lookups take $O(1) + O(\text{load})$ time. Under SUHA, if we hash a string which is not in the hash table, this leads you to a uniformly random cell. So the expected value of that cell's load is just $\lambda = n/m$. In other words:

$$\mathbf{E}[\text{time of an unsuccessful find}] = O(1 + \lambda).$$

Now what about a "successful find" (meaning the user's password is on the blacklist)? What is the *expected* time for this lookup? Focus on this particular bad password x. Its hash $h(x)$ is some cell in the table. Conditioned on that; say it is cell i. Now how many *more* bad passwords do we

expect to hash to cell i? Well, there are $n-1$ more bad passwords, and by SUHA we have that each hashes to cell i with probability $1/m$. So the *expected* load of x's cell is $1 + \frac{n-1}{m} \leq 1 + \lambda$. So we also have

$$\mathbf{E}[\text{time of a successful find}] = O(1 + \lambda).$$

2.2 The tradeoff

Thus we get a natural tradeoff between space utilization and Lookup time: Taking λ large we get a smaller table, since the table size is $m = n/\lambda$. However, this leads to slower Lookup times on average. A popular choice is $m = n$; i.e., $\lambda = 1$.

In several applications you get *really* concerned about Lookup time. In these cases, it makes sense to also worry about the *maximum load*, since this dictates the *worst-case* time of a Lookup. You might remember that we analyzed the maximum load in a balls and bins scenario in Lectures 8 and 9: n "jobs", k "servers". That, however, was in the case where $n \gg k$; in the hashing scenario the number of balls and bins is roughly equal (i.e., λ is a constant). As it turns out, when $m = n$, the maximum load is actually $\Theta(\log n / \log \log n)$, with high probability. You will prove this on the homework!

3 Saving space: Fingerprinting

Let's go back and talk about the bad-passwords example. There we saw that with a hash table, we could get Lookups in expected time $O(1)$ by using a table of size $m = n$. However, in some ways this is not such a great use of space. There are two problems:

- First, you have a lot of empty hash table slots. As we saw in the last lecture, if you have a Balls and Bins process with $n = m$ and hence $\lambda = 1$, the expected fraction of *empty* bins is $1/e$.
- Second, you are storing the whole password in each bin.

As an example, if you have n passwords, each 32 bytes long, then you're using up at least $256n$ bits of space here (more, actually, due to linked list overhead). Suppose you care greatly about using very little space, and you're not *too* worried about the time issue — think of implementing the system a cell phone, for example. Then **fingerprinting** might be the solution for you.

To explain fingerprinting, first notice that if space is a big concern but time isn't, then you probably wouldn't use a hash table at all. For one, it's probably a bit better to go back to the simple solution of just storing all the bad passwords in a sorted array and doing lookups via binary search. Then the time for lookups is not too bad, $O(\log n)$, and the space usage is precisely $256n$ bits.[1]

However, with fingerprinting you can save a factor of, say, 8 on space. It might seem impossible at first to save any space here, but we'll do it by trading space for *error*.

[1] This might be improvable by a small factor with a trie, depending on the exact set of passwords.

3.1 Saving a factor of 8

Specifically, we'll *fingerprint* each string, meaning we'll hash it from a 256-bit string to a 32-bit string, called the "fingerprint". We'll then store the *fingerprints* in a sorted array. We will *not* store the actual 256-bit bad passwords themselves. This reduces space down to $32n$ bits, a factor of 8 as promised.

To check a user-proposed password, we hash it to a fingerprint and use binary search to check whether this fingerprint is in our sorted list. If it is, we reject the password.

Problem: false positives. Something has to give, of course, and it is false positives — it may be that a perfectly acceptable password happens to have the same fingerprint as a bad password. This will cause it to be improperly rejected.

But you know what, that may not be so bad. Okay, the user gets a bit annoyed and has to pick a new password. But you're saving space by a factor of 8! Well, before we get too excited we should analyze the *probability* of a false positive; if it's really high then this will be a bad idea. But as it turns out, it's not very high at all.

How can we compute the probability that a user's *good* password will be improperly rejected? At first this looks slightly difficult to analyze, but with a simple trick it becomes easy. Instead of imagining all the bad passwords are first fingerprinted, and *then* the user selects a good password, think of the random experiment happening in the opposite (but equivalent) order. Imagine you start with the good password. It gets fingerprinted, and by SUHA it goes to *some* 32-bit string — call that y. Now the probability of a false positive is the probability that one of the bad passwords gets fingerprinted to the same string y.

By SUHA, each bad password has a $1/2^{32}$ chance of having its hash value be y, and these events are independent. So

$$\mathbf{Pr}[\textit{not a false positive}] = \mathbf{Pr}[\text{everything has a fingerprint} \neq y] = (1 - 1/2^{32})^n.$$

Perhaps a plausible value for n is $2^{14} = 16384$. Then we would have

$$\mathbf{Pr}[\textit{not a false positive}] = (1 - 1/2^{32})^{2^{14}} \geq 1 - 2^{14}/2^{32} = 1 - 1/2^{18},$$

and hence

$$\mathbf{Pr}[\text{false positive}] \leq 1/2^{18} \approx 4 \times 10^{-6}.$$

which is nice and tiny. Here in the previous line we used the fact that $(1-x)^t \geq 1 - xt$ whenever $x \leq 1$ ("Bernoulli's Inequality", it's called). This is close to be an equality when x is small, by the way, by the Most Useful Approximation Ever:

$$(1-x)^t \approx (e^{-x})^t = e^{-xt} \approx 1 - xt.$$

3.2 In general: bits per item vs. false positive rate

More generally, suppose we have a set of n items (strings, e.g.) we want to store. We don't mind having lookup time $O(\log n)$, and we don't mind having a small chance of a false positive.

Question: With fingerprinting, if we're willing to use b bits of space per item, how low can we get the false positive rate?

Answer: We will be hashing each item to a fingerprint of length b bits. So as in the calculation above, if we are looking up some item, the probability we *don't* have a false positive is

$$(1 - 1/2^b)^n \geq 1 - n/2^b,$$

and hence the probability of a false positive is at most $n/2^b$. Clearly we will need to take b to be at least $\log_2 n$, or else this bound is bigger than 1! On the other hand, with each extra bit we use beyond $\log_2 n$, we get an exponential decrease in the probability of a false positive.

Summary: By fingerprinting to $\log_2 n + c$ bits per item, you can store a set of n items and have

$$\mathbf{Pr}[\text{false positive on lookup}] \leq 2^{-c}.$$

4 Saving even *more* space: Bloom Filters

Although fingerprinting is a nice idea, it's more of a warmup for an even nicer idea: *Bloom Filters*. As we just saw, fingerprinting is quite useless unless you store at least $\log_2 n$ bits per key. Can we get away with even *less*?! Could we somehow store just a *constant* number of bits per key, 5 or 10 say, and still have low false-positive rate? It turns out that Bloom Filters let you do precisely this.

Bloom Filters are a very interesting data structure, for a couple of reasons. First, they are far less widely known than they ought to be, given that they're very simple and very useful in practice. Here is an example straight from real-life: it's how Google's "Bigtable" works (or HBase in Hadoop). Suppose you are managing a truly gigantic spreadsheet, so gigantic that you have to store it on many many disks. The spreadsheet is somewhat sparsely populated though, so if a user requests an entry from a certain cell, that cell may well be empty. Because hitting the disk is so slow, you'd like to keep track (in memory, say) of the locations of all the nonempty cells. Naturally, space is at a premium here, but also it's not the end of the world if you have a false positive. So indeed, Google's Bigtables use Bloom Filters to store the nonempty cell locations. Bloom Filters also get used a lot in web caching, and many other networking applications.

The other, even more interesting thing about Bloom Filters is that their eponym is truly a man of mystery. Bloom Filters were first described in a 1970 paper in *Communications of the ACM* by one Burton H. Bloom. In addition to his name, the paper contains an affiliation:

Computer Usage Company, Newton Upper Falls, Mass.

Bloom apparently wrote one other paper, 7 years earlier, in AI Memo 47 on heuristics for computer chess. He is otherwise a complete mystery. No one knows exactly who he is, or indeed whether he's alive or not. No less an authority than Don Knuth has publicly pleaded for someone who knows him to come forward. (Knuth probably really just wants to know what the H stands for.) Even "Computer Usage Company" seems to be little known; it's claimed on the Internet that it was the world's first software company, founded in 1955. It IPO'd in 1960 for $186,681, but folded in 1986.

4.1 Bloom Filters

So what *are* Bloom Filters? We can at least answer this :)

As the user, we first select k, a small constant integer such as 6. We then set (with some foresight that will be explained later)

$$m = \frac{1}{\ln 2} kn.$$

Well, we round this number off, since m should be an integer too. This m is the number of storage bits we will use; therefore we will be using essentially

$$b = \frac{1}{\ln 2} k \approx 1.44k \quad \text{bits per item.}$$

In our example with $k = 6$, this is just 8.66 bits per item.

We next make the Bloom Filter:

- Allocate an array $A[\,]$ (the Bloom Filter) of m bits. Initialize all bits to 0.
- Choose k "independent" hash functions,

$$h_1, \ldots, h_k : \{\text{items}\} \to \{1, 2, \ldots, m\}.$$

- "Insert" each of the n items into the Bloom Filter as follows: to insert x,

$$\text{set } A[h_1(x)] \leftarrow 1, A[h_2(x)] \leftarrow 1, \ldots, A[h_k(x)] \leftarrow 1.$$

  Note that in the course of doing this, some of these bits may *already* be set to 1. That's okay.

That's it. Note that creating the Bloom Filter takes time $O(n)$, assuming k is constant.

Here's how to do a Lookup:

- Lookup(y): Check whether $A[h_1(y)], \ldots, A[h_k(y)]$ are all set to 1. If so, report that y is in the set. If not, report that it's not.

Analysis: First, note that this algorithm has no false negatives; if y is really in the set, the Bloom Filter will really report it as in there. Next, note that Bloom Filters have $O(1)$ lookup time (again, assuming that k is constant). So they're even faster than fingerprinting! The main question is, of course, what is the probability of a false positive?

In answering this, we will cheat a little bit. Sorry, but to do it carefully requires some approximation arguments which are too painful to give in a single lecture.

Let's start by thinking about the following:

Question: What *fraction* of bits in A will be 1 after we insert the n items?

Answer: Inserting the n items is (under SUHA[2]) like throwing kn balls into m bins. The average load here is

$$\lambda = \frac{kn}{m} = \frac{kn}{(1/\ln 2)kn} = \ln 2,$$

by our choice of λ. Under the Poisson Approximation, the distribution of the number of balls in bin i is thus Poisson($\ln 2$). Since bit $A[i]$ is set in the Bloom Filter if and only if the corresponding bin i gets at least 1 ball, we conclude that

$$\mathbf{Pr}[A[i] = 1] \approx 1 - \mathbf{Pr}[\text{Poisson}(\ln 2) = 0] = 1 - \exp(-\ln 2) = 1 - 1/2 = 1/2.$$

So the expected number of bits set to 1 in the Bloom Filter is exactly 1/2 (assuming the Poisson Approximation).

In fact, we claim that there is an extremely high probability that the fraction of bits set to 1 is extremely close to 1/2. *If* the events $A[i] = 1$ were independent, we could prove this easily using the Chernoff Bound. Unfortunately, they're not.[3] It is possible to overcome this and still get a Chernoff-like bound; however, it's a little complicated and we will avoid getting into it.[4] So let us simply assume (with some cheating) that the fraction of bits $A[i]$ which are 1 is precisely 1/2.

Now think about what happens when we do a Lookup on some item y which is *not* in the set we stored. The positions we look up, $h_1(y), \ldots, h_k(y)$, are independent random numbers in $\{1, \ldots, m\}$ (by SUHA). Since we're assuming exactly 1/2 the bits in $A[\ ]$ are 1, the probability that all k positions contain a 1 is just 2^{-k}.

Summary: Using a Bloom Filter with k hash functions, you can store a set of n items using $b = \frac{1}{\ln 2}k \approx 1.44k$ bits per item, and have

$$\mathbf{Pr}[\text{false positive on lookup}] \leq 2^{-k}.$$

This is great because it's exponentially small in k (and in b)! If you want to halve the probability of a false positive, you only need to increase your storage by about $1.44n$ bits. In our example with $k = 6$, you use $8.66n$ bits of storage and have false-positive probability $2^{-6} = 1.6\%$. So you see it's still a tradeoff versus fingerprinting: less storage by a significant factor, faster storage and lookup, but higher false-positive probability by quite a bit. (And Bloom Filters do not support deletions!)

[2] Actually, an extension which allows us to get multiple "independent" hash functions.

[3] Although they are if we use the "extended" Poisson Approximation mentioned last lecture, wherein we pretend that the numbers of balls in each bin are independent ;)

[4] Here is one sketch for why it's okay: Really, we don't care that the fraction of 1's is *exactly* 1/2, we just care that it's *at most* 1/2. That's because we are trying to show that the chance of a false positive is low, so we are hoping that $A[\ ]$ has *few* bits set to 1. The events $A[i] = 1$ are not independent, which is too bad, since if they were we could use Chernoff. But actually, intuitively speaking, they are *negatively correlated*. E.g., if I tell you that $A[7] = 1$, it means that some balls went into bin 7, which makes you feel like it's even *less* likely than usual that, say, $A[36] = 1$. Intuitively speaking, having this negative correlation between the events $A[i] = 1$ should be even *better* than having independence, for the purposes of proving that not more than 1/2 of the events happen. This intuition can be made rigorous.

15-359: Probability and Computing
Fall 2009

Lecture 13: Random Graphs

In this lecture we will discuss three random graph models: (i) the Erdős–Rényi model $\mathcal{G}_{n,p}$; (ii) the Given Degree Sequence model of Bollobás; (iii) the Barabási–Albert Preferential Attachment model.

1 Gigantic Graphs

Gigantic graphs are everywhere these days. Here are some examples:

- The Web: nodes = pages, arcs[1] = links.
- Social networks: nodes = people, edges = acquaintanceships. (This may be informal, or formalized, as in Facebook.)
- Collaboration networks: nodes = researchers, edges = coauthorships.
- Movies: nodes = actors, edges = costars.
- The Internet: nodes = ISPs, edges = connections.
- Peer-to-peer file sharing systems: nodes = computers, edges = connections.
- Chain mail: nodes = recipients, arcs = forwards.
- Telephone grids: nodes = phones, edges = lines.
- Power (electricity) grids: nodes = transmission towers, edges = transmission lines.
- Neural networks: nodes = neurons, nodes = synapses.

These graphs have a number of vertices, "n", and number of edges "m" ranging from the thousands to the billions (or more). It's not surprising that many of the examples are computer science-related, since it's hard to keep track of gigantic masses of data without computers.

What do these graphs "look like"? They're clearly far to big to draw, so it's hard to actually "look" at them. It's not easy to say what this question even means. One thing you could at least do is look at various statistics; e.g.:

- What is the "density" of edges, m/n? (Relatedly, what is the "average degree", $2m/n$?) Is it "sparse", meaning the density is like a small constant? Is it "dense", meaning that $m = \Omega(n^2)$? Something in between?

[1] I.e., directed edges

- Is the graph "connected"? I.e., is there a path between every pair of vertices? Is it "very connected"? I.e., are there *many* paths between every pair of vertices?
- If the graph isn't connected, how many vertices are in the largest component?
- If the graph *is* connected, what is the "diameter", i.e., the maximum distance between a pair of nodes?
- What is the "degree distribution"? How many vertices have degree 0, have degree 1, have degree 2, etc.?
- What is the "girth", i.e., the length of the shortest cycle? How many "triangles" (i.e., 3-cycles) does the graph have?

If you just have a single graph you're interested in (the Movie network, the Web) then perhaps you could try to determine these statistics and be done. But that's not very scientific! It's important to ask, do various gigantic graphs out there have common statistical features? What is it about how the graph is created that determines these features? Can you predict how a newly created giant graph will behave? Can you predict one statistical feature based on some others?

Perhaps most interestingly, how do these features affect algorithms? Think, e.g., of a routing protocol for the Internet, or a viral marketing strategy for a social network. What properties of gigantic graphs make various algorithms do better or worse?

To try to answer these questions, we need some models for gigantic graphs — and there's no better model than *random graphs*.

2 Random Graphs

Really, what we're interested in is a fixed block of randomized code whose *output* is a graph (an *undirected* graph, say). This randomized code is sometimes called a "random graph *process*". Almost always, the number of *vertices* in the graph is decided on in advance. Here is the simplest possible random graph process: it's called $\mathcal{G}_{n,1/2}$:

$\mathcal{G}_{n,1/2}$: input is n, the desired number of vertices:
  `for` $i \leftarrow 1 \ldots n$
    `for` $j \leftarrow i+1 \ldots n$
      $A[i,j], A[j,i] \leftarrow$ `Bernoulli`$(1/2)$.

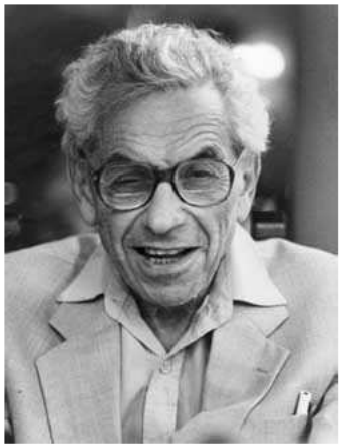
Here $A[\ ]$ gives the adjacency matrix of the graph. In plain words, $\mathcal{G}_{n,1/2}$ randomly makes a graph by taking each "potential edge" (i,j) to be present with probability $1/2$ and absent with probability $1/2$, independently for each potential edge. (We often say that $\mathcal{G}_{n,1/2}$ generates a "random graph", but remember that this is a bit of a misnomer; rather, $\mathcal{G}_{n,1/2}$ randomly generates a graph.)

The "1/2" in the name $\mathcal{G}_{n,1/2}$ indeed comes from the fact that each "potential edge" is present with probability $1/2$. We can of course generalize this to any probability $p \in [0,1]$. The resulting random graph process is called the *Erdős–Rényi random graph model* because, of course, it was first introduced by Edgar Gilbert in 1959 :) Gilbert worked at Bell Labs and was motivated by the

phone network. Around the same time, Austin, Fagen, Penney and Riordan introduced (basically) the related $\mathcal{G}_{n,m}$ model described below. Anyway, Paul Erdős and Alfréd Rényi were Hungarian mathematicians who independently introduced the model in 1960; they get the naming credit because they studied it much, much more extensively.

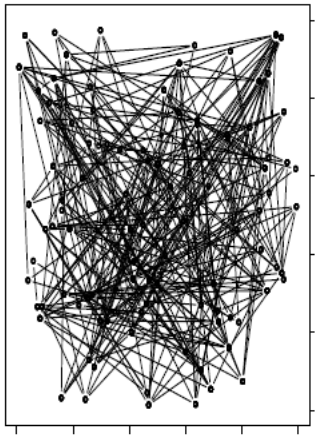
Definition 1. *The Erdős–Rényi random graph model, denoted $\mathcal{G}_{n,p}$, is to randomly make a graph on n vertices by choosing each potential edge (i, j) to be present with probability p and absent with probability $1 - p$, independently for all $\binom{n}{2}$ potential edges.*

Here is an illustration.[2]



Erdös

Rényi



Erdös–Rényi

There is a related Erdős–Rényi random graph model, called (slightly weirdly) $\mathcal{G}_{n,m}$. In the Erős–Rényi $\mathcal{G}_{n,m}$ model, you randomly make a graph on n vertices with *exactly* m edges, by choosing each of the $\binom{\binom{n}{2}}{m}$ possibilities with equal probability. The $\mathcal{G}_{n,m}$ model produces graphs which are extremely similar to those produced by $\mathcal{G}_{n,p}$ with $p = m/\binom{n}{2}$. However it's more of a pain to work with $\mathcal{G}_{n,m}$, so we will stick $\mathcal{G}_{n,p}$.

2.1 More on $\mathcal{G}_{n,1/2}$

We'll talk about the $\mathcal{G}_{n,p}$ model more later in the lecture, but now let's start with the simplest case, $\mathcal{G}_{n,1/2}$. Actually, this is a particularly special and simple case, because of the following:

Observation: Choosing a graph from $\mathcal{G}_{n,1/2}$ is equivalent to choosing a graph on vertices 1, 2, ..., n uniformly at random from all $2^{\binom{n}{2}}$ possible such graphs.

You should be able to easily convince yourself of this.

So what do $\mathcal{G}_{n,1/2}$ graphs look like? Or, perhaps more accurately, what do they "tend" to look like? Let $G \sim \mathcal{G}_{n,1/2}$ denote a "random graph" drawn from the model $\mathcal{G}_{n,1/2}$.

Question: What is the expected number of edges in G?

[2]Thanks to Prof. Lafferty for this.

Answer: For each potential edge (i, j), let $X_{i,j}$ be the indicator that this edge is present in G. The total number of edges is thus $\sum_{i<j} X_{i,j}$, so by linearity of expectation, the expected number of edges is $\binom{n}{2} \cdot (1/2) = n(n-1)/4 \approx .25n^2$.

You can use a Chernoff Bound to show that the number of edges will be between, say, $.24n^2$ and $.26n^2$ with extremely high probability. (The probability of deviating outside this range will be exponentially small in n. We'll do a related calculation shortly.) So these graphs tend to be extremely dense.

Question: What is the expected degree of vertex i in G?

Answer: Vertex i has $n-1$ potential neighbors, and each is an *actual* neighbor with probability $1/2$. Hence the degree of vertex i has distribution Binomial$(n-1, 1/2)$ and hence expectation $(n-1)/2$.

In fact, *every vertex* will have degree extremely close to $n/2$ "with high probability":

Theorem 2. *Let $G \sim \mathcal{G}_{n+1,1/2}$, where we've used "$n+1$" vertices instead of "$n$" for future typographic simplicity. Then "with high probability",* all *vertices have degree between $n/2 - \sqrt{n \ln n}$ and $n/2 + \sqrt{n \ln n}$. Here "with high probability" means with probability at least $1 - O(1/n)$.*

Proof. We use the Chernoff + Union Bound method. Let D_i denote the degree of vertex i. As we saw, the random variable D_i has distribution Bernoulli$(n, 1/2)$. Using Chernoff Bound 1 we have

$$\begin{aligned}\mathbf{Pr}[D_i \geq n/2 + \sqrt{n \ln n}] &= \mathbf{Pr}[D_i \geq n/2 + (2\sqrt{\ln n})\sqrt{n}/2] \\ &\leq \exp(-(2\sqrt{\ln n})^2/2) = \exp(-2 \ln n) = 1/n^2.\end{aligned}$$

Similarly,

$$\mathbf{Pr}[D_i \leq n/2 - \sqrt{n \ln n}] \leq 1/n^2.$$

Thus

$$D_i \in [n/2 - \sqrt{n \ln n}, n/2 + \sqrt{n \ln n}] \quad \text{except with probability at most } 2/n^2.$$

Taking a Union Bound over all $n+1$ vertices, we get

$$\mathbf{Pr}[\textit{any} \text{ vertex has degree outside this range}] \leq \sum_{i=1}^{n+1} (2/n^2) = 2(n+1)/n^2 = O(1/n).$$

□

2.2 With high probability

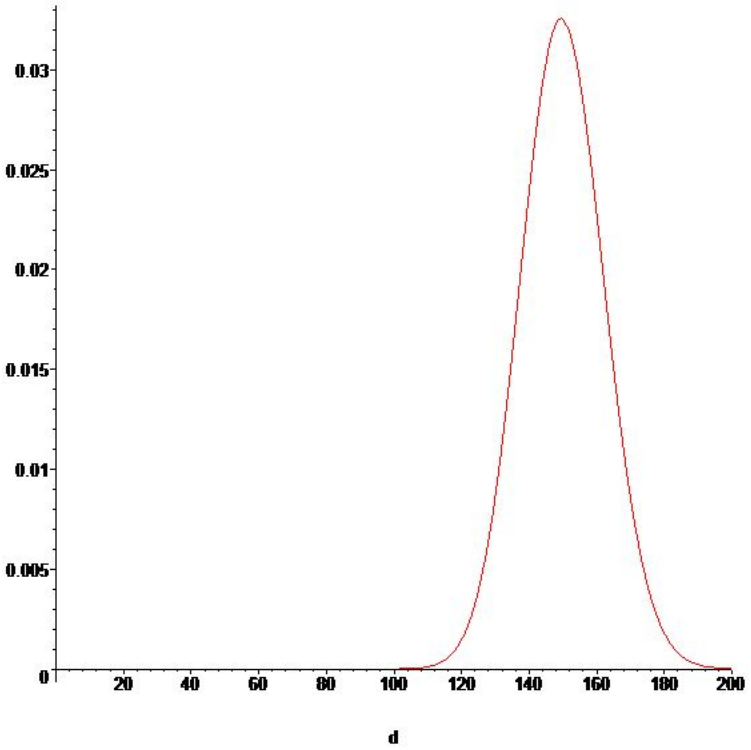
By now you've seen us toss around the phrase "with high probability" a few times — the phrase comes up a lot in the discussion of balls and bins, and random graphs. For any given problem, "with high probability" means "with probability at least $1 - O(1/n^c)$ for some constant $c > 0$", whatever "n" happens to be in your problem (usually this is obvious). For example, in the above Theorem the probability that some vertex failed to have degree in the given range was at most $2/n$ (so here "c" is 1). The expression "with high probability" is often abbreviated "whp".

3 Degree distribution, and other models

We've already seen two (families of) models for random graphs, $\mathcal{G}_{n,p}$ and the very related $\mathcal{G}_{n,m}$. The $\mathcal{G}_{n,1/2}$ model is well-studied in mathematics, but is not always very appropriate for practice. This is because gigantic graphs arising in practice are rarely so dense; if $n = 10^9$, then a $\mathcal{G}_{n,1/2}$ graph will typically have on the order 10^{18} edges, which is astronomical. "Sparsish" random graphs are much more common in practice.

Example: Social networks. A reasonable first model for a "sparsish" random graph might be $\mathcal{G}_{n,p}$ with $p = 150/n$, say. The number of edges in this graph is Binomial($\binom{n}{2}, p$) and thus has expectation $\binom{n}{2}p \approx (n^2/2)(150/n) = 75n$. The degree of a particular vertex in this model has distribution Binomial$(n-1, 150/n)$ and thus, by the Poisson Approximation has distribution very close to Poisson(150).

However, this too might not be very realistic. Imagine modeling the friendship network on Facebook. It might indeed be the case that the average number of friends a person has is 150 ("Dunbar's Number"). But the Poisson(150) PMF looks like this:



That doesn't look so good. For example,

$$Pr[\text{Poisson}(150) = 25] = \frac{e^{-150} \cdot 150^{25}}{25!} \approx 10^{-36}.$$

That means that the expected number of people with exactly 25 friends would be $10^{-36}n$, which is basically 0. But surely there are plenty of people on Facebook with exactly 25 friends! In general, the "degree distributions" you get out of sparse $\mathcal{G}_{n,p}$ models have the property that they are very concentrated around their expectation, which often does not match reality.

3.1 Degree sequences

Given a *particular* graph G with n vertices, its "degree sequence" is just a list of numbers, $d_0, d_1, d_2, \ldots, d_n$, where

$$d_j = \#\text{ of vertices in } G \text{ with degree exactly } j.$$

Of course,

$$d_0 + d_1 + \cdots + d_n = n$$

and

$$\sum_{j=0}^{n} d_j \cdot j = 2m,$$

where m is the number of edges in G.

Consider again modeling the Facebook graph, which has approximately $n = 10^8$ nodes. Facebook certainly knows the degree sequence of its graph, and if they weren't so sucky about publishing data, we would probably know it too. But assume we *start* with some degree sequence $d_0, d_1, d_2, \ldots, d_n$, where $\sum d_j = n$ and $\sum d_j \cdot j$ is even. Is there a reasonable random graph model which *creates* a graph with this exact degree sequence?

Béla Bollobás introduced the following method around 1980:

Random graph with given degree sequence:

1. First, decide what degree each vertex will have: Randomly order the vertices $1, \ldots, n$. Then decide that the first d_0 vertices in the ordering will have degree 0, the next d_1 vertices in the ordering will have degree 1, the next d_2 in the ordering will have degree 2, etc.

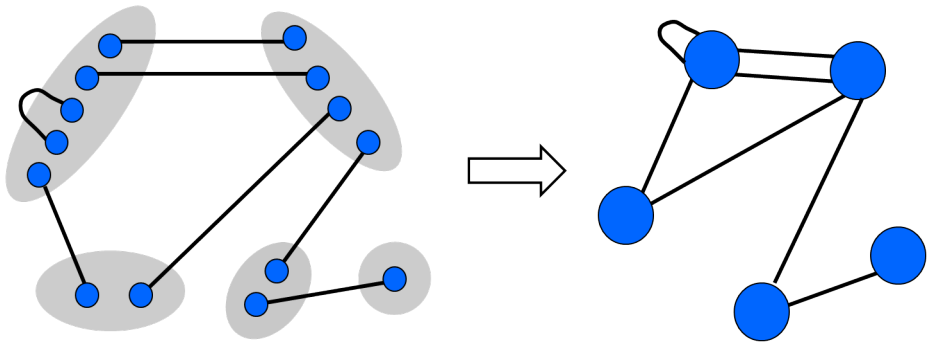
2. Next, for each vertex i, if is supposed to have degree c_i, blow it up into a group of c_i "mini-vertices".

3. There are now $\sum d_j \cdot j = 2m$ mini-vertices. Pick a uniformly random *perfect matching* on these vertices. (This is easy: take any mini-vertex; match it with a uniformly random unmatched mini-vertex; repeat.)

4. Now merge n groups of mini-vertices back into "vertices".

Here is a picture, with $n = 5$ and given degree sequence:

$$d_0 = 0, \quad d_1 = 1, \quad d_2 = 2, \quad d_3 = 0, \quad d_4 = 1, \quad d_5 = 1$$



As you can see, this method might not be 100% satisfactory because it can generate "non-simple" graphs — i.e., graphs with multiple edges and self-loops. However: a) if you only have low (constant) degrees and n is large, you are unlikely to get any multiple edges or self-loops; b) there are ways to sort of fix this problem, although they are significantly more complicated.

3.2 Preferential attachment/Power laws

This degree-sequence model may be good if you want to model a complex gigantic graph, and you know something about the desired degree statistics. But this sort of modeling in some way misses the point. The graph was itself formed organically by some kind of "randomish" process — think about the various ways the Web, or the power grid, or Facebook got formed. Can we model *that* somehow, and if we do, does the model produce graphs with statistics close to those seen in practice?

This is a pretty hard question that people are still working on; ideas are different for different gigantic graphs. The problem may have been worked on most thoroughly for the case of the Web. An influential early model for generating "Web"-like graphs in was given in 1999 by Albert-László Barabási and Réka Albert, who called it "preferential attachment":

Preferential Attachment — the Barabási-Albert ("BA") Model:

1. Start with 0 nodes and 0 edges.
2. Add a new node and stick one end of an edge into it.
3. Stick the other end of this edge into a random node, chosen with probability proportional to its degree.
4. Repeat Steps 2–4 n times in total.

This process generates a graph with n nodes and n edges (multiple edges and self-loops are possible). To clarify Step 3, suppose we have repeated the process t times. We then do the $(t+1)$th iteration of Step 2. At this point there are $t+1$ vertices. In the first t vertices there are t edges, so the sum of the degrees of the first t vertices is $2t$. The $(t+1)$th iteration of Step 2 yields a $(t+1)$th node with degree 1. So the "other end of the $(t+1)$th vertex's edge" goes into:

$$\text{vertex } 1 \leq i \leq t \text{ with probability } \deg(i)/(2t+1),$$

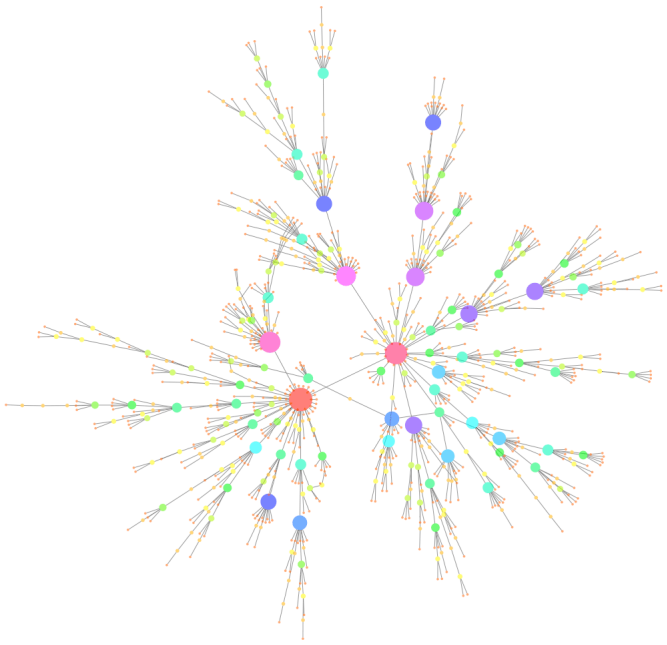
$$\text{vertex } t+1 \text{ itself with probability } 1/(2t+1).$$

E.g., after the first step you always have a single node with a self-loop. After the second step, vertex 2 gets attached to vertex 1 with probability 2/3 and to itself with probability 1/3.

The idea behind this simple model is that when a new web page comes along it adds a link from itself to some random web page, but the more "authoritative" or "popular" web pages are more likely to be linked to. Of course, this is a very simplistic model. For one thing, on the Web links are directed, not undirected. For another, new web pages often include more than one link when they get started! This second objection is incorporated into a variant on the BA Model:

BA Model with average degree $2c$: First, do the BA Model to produce a graph with cn nodes and cn edges. Then merge nodes $1 \dots c$, $c+1 \dots 2c$, $2c+1 \dots 3c$, etc, to produce a graph with n nodes and cn edges.

The BA Model's simplicity can be a virtue too; the model produces fairly "realistic"-looking graphs. Here is a picture with $n = 1000$:



The BA Model is *just* simple enough that proving things about it is possible, albeit hard. Here is one example theorem, proved by Bollobás, Riordan, Spencer, and Tusnády:

Theorem 3. *Assume $c = O(1)$ is a constant. Then with high probability, the fraction of vertices having degree d is $\Theta(d^{-3})$. (This holds for all $d \leq n^{.05}$, say.)*

If a graph has the property that the fraction of vertices with degree d is $\Theta(d^{-\alpha})$ for some constant α, its degree distribution is said to have a **power law with exponent** α. Such graphs are also called **scale-free networks**. In practice, it *seems* like many gigantic graphs are scale-free in this way, although this fact is still disputed. Various sources claim the power law exponent for the Web is anywhere between 1.5 and 3, and similar claims are made for social network power law exponents. Different tweaks on the BA Model can make it give power laws with exponents other than 3.

(Amazingly, a similar preferential attachment graph model, along with an empirical observation of a power law, was made by Scottish statistician Udny Yule in 1925. 1925! Yule was studying evolution, in the context of plants.)

4 Threshold phenomena in Random Graphs

One notable phenomenon that occurs again and again in the study of random graphs is **threshold behavior** — "Tipping Points" as the Canadian journalist Malcolm Gladwell might call them, or "phase transitions" as the physicists call them. This threshold behavior has been studied most

extensively in the Erdős-Rényi model, and that's where it's easiest to analyze too. So that's what we'll talk about now.

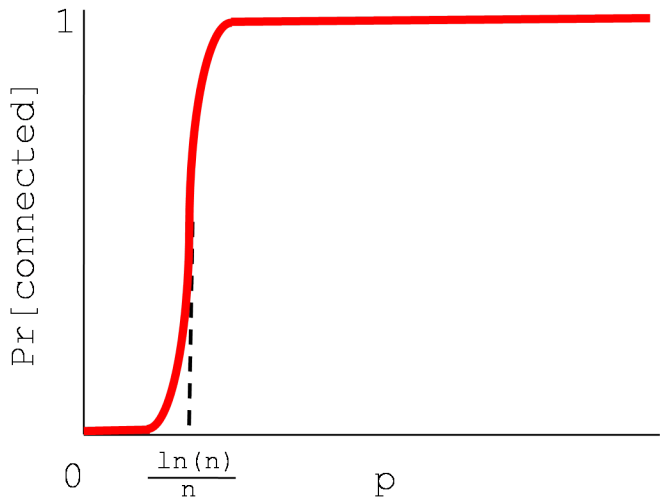
There is an interesting "stochastic process" way of looking at the $\mathcal{G}_{n,p}$ model:

$\mathcal{G}_{n,p}$ as a stochastic process: Each "potential edge" (i,j) independently picks a random real number X_{ij} uniformly in the range $[0,1]$.[3] Now imagine that we "start" $p=0$ and slowly crank it up to $p=1$ over time. Initially all potential edges are "off" (think of them like neon lights). As soon as p gets up to X_{ij}, the edge (i,j) turns "on". In particular, eventually $p=1$ and all edges are on.

Key observation: Fix a particular value $0 \le p_0 \le 1$. Imagine the graph derived by this stochastic process when $p = p_0$. It consists of precisely those edges whose number $X_{i,j}$ is at most p_0. The probability $X_{i,j} \le p_0$ is[4] p_0, and the events $X_{i,j} \le p_0$ are independent. So the distribution of the graph when $p = p_0$ is just $\mathcal{G}_{n,p_0}$.

So the interesting thing about this process is that we can sort of "watch" a graph gain more and more edges, going from the empty graph to the complete graph; and when the "time" is at p, the graph is distributed precisely like $\mathcal{G}_{n,p}$. Actually, the "stochastic process" version of $\mathcal{G}_{n,m}$ is even more natural: in this, you start with the empty graph and for each of m time steps you add in one more uniformly random edge.

Let's now look at various graph **properties**; e.g., the property of being connected. Suppose we plot the probability that a graph $G \sim \mathcal{G}_{n,p}$ is connected, as a function of p. The plot turns out to look something like this (but we haven't drawn things to scale):



Theorem 4. *If $p \le .99\frac{\ln n}{n}$ then a graph $G \sim \mathcal{G}_{n,p}$ is* not connected *with high probability. If $p \ge 1.01\frac{\ln n}{n}$ then a graph $G \sim \mathcal{G}_{n,p}$* is connected *with high probability. In other words, $\frac{\ln n}{n}$ is a "sharp threshold" for connectivity.*

[3] Granted, we have not said what this means yet, but you can probably handle it for the purposes of this discussion. If you really want to be a stickler, imagine that we do $X_{ij} \leftarrow \texttt{RandInt}(10^{100}) \cdot 10^{-100}$.

[4] We think you can agree.

In fact, the threshold is much much sharper than illustrated in the picture :) We have the tools to prove this theorem, but it takes some effort, so we will skip it.

Another example of a threshold phenomenon occurs if you look at the *size of the largest connected component*. We've just seen that if $p \geq 1.01\frac{\ln n}{n}$, the size of the largest component is exactly n with high probability. What about for smaller values of p?

Theorem 5. *If $p \leq .99/n$ then a $\mathcal{G}_{n,p}$ graph has largest component of size $O(\log n)$, with high probability. If $p \geq 1.01/n$ then a $\mathcal{G}_{n,p}$ graph has largest component of size $\Omega(n)$, with high probability.*

Now that's a real tipping point! It's nice to think about this theorem in the context of the stochastic process. As you slowly crank p up from 0, for a little while the graph you get will (with high probability) look like many scattered tiny clumps, each of size at most $O(\log n)$. This holds for p almost as large as $1/n$. Then boom! As soon as p becomes larger than $1/n$ suddenly almost all of these clumps "clump together" and you get one "giant component" consisting of a constant fraction of all vertices!

15-359: Probability and Computing
Fall 2009

Lecture 14: Finite, discrete-time Markov chains

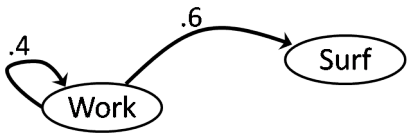
In the next few lectures we will discuss *Markov Chains*. Markov Chains are ubiquitous in computer science, and indeed almost all of science. They arise especially in statistical physics, genetics, statistics, computational geometry, optimization, algorithms, performance analysis, natural language processing, you name it. Today we'll give a toy example and start the basic theory.

1 Basics of finite, discrete-time Markov chains

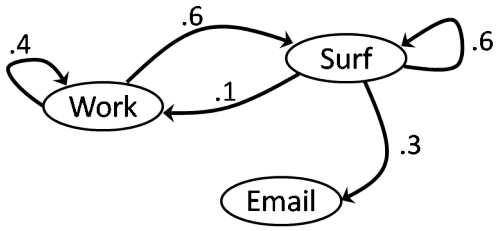
1.1 An example: my day at work

Let's start with a Markov Chain related to my daily life. Every day I wake up and try to get a lot of work done. When I'm working, I look like this:

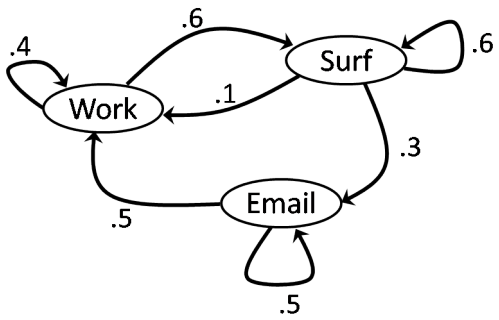
I am easily distracted though. *After each minute* of work, I only keep working with probability .4. With probability .6, I begin surfing the web:



Surfing the web is very addictive. After each minute of surfing the web, I only return to working with probability .1. With probability .6 I continue surfing the web. With probability .3 I feel kind of guilty, so I check my email, which is sort of like working.



After each minute of email-checking, I have probability .5 of coming to an action item which causes me to go back to work. With probability .5 I continue checking my email (I have virtually unending amounts of email).



This last diagram is my daily life. It is a classic example of a *(discrete-time) Markov Chain* with a finite number of states. Markov Chains are named after our old friend and hero of probability, Andrey Andreyevich Markov (senior), who also gave his name to Markov's Inequality. Markov Chains are very frequently drawn as finite-state diagrams like the one above; the key property is that the sum of the probabilities coming out of each state should be 1.

An alternative way to describe a Markov Chain is by a square "transition matrix" K. The rows and columns of K are indexed by the "states" of the Markov Chain. The entry in the (u, v) position, which we write as either K_{uv} or $K[u, v]$ is equal to the probability of taking a "step" from state u to state v. All the entries need to be filled in, so when there is no arrow from state u to state v in the diagram, the entry K_{uv} should be 0. Here is the transition matrix for my working Markov Chain:

$$K = \begin{array}{c} \\ W \\ S \\ E \end{array} \begin{array}{c} \begin{array}{ccc} \curvearrowright & W & S & E \end{array} \\ \begin{pmatrix} .4 & .6 & 0 \\ .1 & .6 & .3 \\ .5 & 0 & .5 \end{pmatrix} \end{array}$$

A legal Markov Chain transition matrix has nonnegative entries, and all the row-sums are 1.

1.2 Definition

A (discrete-time) *finite Markov Chain (MC)* with finite state set S is defined by a finite-state diagram with states S and probabilities on the arrows. We often assume (by relabeling) that the state set is $S = \{1, 2, 3, \ldots, r\}$. Equivalently, one can define a finite MC by any $r \times r$ *transition matrix* K satisfying:

- $K_{uv} \geq 0$ for all $1 \leq u, v \leq r$;
- $\sum_{v=1}^{r} K_{uv} = 1$ for all $1 \leq u \leq r$.

We won't talk about "continuous-time Markov Chains" until later in the course, so we will stop writing "(discrete-time)" all the time for now. Also, in this lecture we will only talk about MC's with finite state sets. (Countably) infinite state sets will be discussed in the next lecture.

We also always associate to a Markov Chain the following experiment (randomized code):

`// the initial state,` X_0`, is some random variable with values in` $\{1, 2, \ldots, r\}$
`for` $t = 1, 2, 3, \ldots$

$$X_t \leftarrow \begin{cases} 1 & \texttt{with probability } K[X_{t-1}, 1], \\ 2 & \texttt{with probability } K[X_{t-1}, 2], \\ \ldots & \\ r & \texttt{with probability } K[X_{t-1}, r]. \end{cases}$$

In other words, given some initial random (or deterministic) state X_0, we generate a sequence of random states X_0, X_1, X_2, X_3, etc., according to the given transition probabilities. We always describe the index t as the *time*.

Now is good time to make a definition which gives a fancy name to a simple concept:

Definition 1. *A collection of random variables $\{X_t : t \in T\}$ is called a* stochastic process *with* time set T. *Often $T = \mathbb{N}$, as in discrete-time Markov Chains. The random variable X_t is called the* state at time t.

1.3 The Markov property

The essential property of Markov Chains, which is both a limitation and a feature, is the following:

Limitation/feature: The state of a Markov Chain at time t is determined only by its state at time $t-1$.

This is the so-called *Markov/Markovian/memoryless property*. At time $t-1$, the process essentially "forgets" how it got into its present state X_{t-1}. All that matters in generating the tth state is what the $(t-1)$th state is.

Question: Does this mean that the random variable X_t is independent of random variables $X_0, X_1, \ldots, X_{t-2}$?

Answer: No! You can prove this rigorously for yourself, but consider my work Markov Chain, for example. You should find it intuitive that $\mathbf{Pr}[X_t = \text{'Surf'} \mid X_{t-2} = \text{'Surf'}]$ is noticeably higher than just $\mathbf{Pr}[X_t = \text{'Surf'}]$. What *is* true the following: Although X_t is dependent on $X_0, X_1, \ldots, X_{t-1}$, *all of the dependency is captured by X_{t-1}*.

Markov Chains are sometimes defined differently, in terms of the Markov property:

Equivalent Definition: *A Markov Chain with finite state set S* is a stochastic process $X_0, X_1, X_2, \ldots$ with the following property:

$$\begin{aligned} \forall t, \quad \forall u_0, u_1, \ldots, u_t \in S, \qquad & \mathbf{Pr}[X_t = u_t \mid X_{t-1} = u_{t-1}, X_{t-2} = u_{t-2}, \ldots, X_1 = u_1, X_0 = u_0] \\ = \quad & \mathbf{Pr}[X_t = u_t \mid X_{t-1} = u_{t-1}] \\ =: \quad & K[u_{t-1}, u_t]. \end{aligned}$$

2 Limiting probabilities

2.1 State at time t: matrix powers

I should probably cut down on my web surfing. In fact, I wonder: if you walk into my office some t minutes into the work day for some large value t, what is the probability I'm surfing?

When we discussed whether X_t was independent of $X_0, \ldots, X_{t-2}$, we said that $\mathbf{Pr}[X_t = \text{'Surf'} \mid X_{t-2} = \text{'Surf'}]$ is noticeably higher than just $\mathbf{Pr}[X_t = \text{'Surf'}]$. Actually, this not quite clear, because $\mathbf{Pr}[X_t = \text{'Surf'}]$ isn't completely defined. It depends on the random variable X_0 giving the starting state! If we care about the probability I'm surfing at some late time t minutes into the day, we're really asking:

Question: What is $\mathbf{Pr}[X_t = \text{'Surf'} \mid X_0 = \text{'Work'}]$? What is $\mathbf{Pr}[X_t = \text{'Surf'} \mid X_0 = \text{'Surf'}]$? What is $\mathbf{Pr}[X_t = \text{'Surf'} \mid X_0 = \text{'Email'}]$?

We might as well try to answer this question in a little more generality; assume a MC with r states and transition matrix K. We'll start slow:

Question: What is $\mathbf{Pr}[X_1 = v \mid X_0 = u]$?

Answer: By definition, it's $K[u, v]$.

Question: What is $\mathbf{Pr}[X_2 = v \mid X_0 = u]$?

Answer: We solve this by the Law of Total Probability, conditioning on the the state X_1:

$$\begin{aligned}
\mathbf{Pr}[X_2 = v \mid X_0 = u] &= \sum_{w=1}^{r} \mathbf{Pr}[X_1 = w \mid X_0 = u]\mathbf{Pr}[X_2 = v \mid X_1 = w, X_0 = u] \\
&= \sum_{w=1}^{r} \mathbf{Pr}[X_1 = w \mid X_0 = u]\mathbf{Pr}[X_2 = v \mid X_1 = w,] \quad \text{(Markov property)} \\
&= \sum_{w=1}^{r} K[u, w]K[w, v].
\end{aligned}$$

We hope this expression leaps out to you as familiar: it is exactly the (u, v)-entry in the matrix product $K \times K$. I.e.,

$$\mathbf{Pr}[X_2 = v \mid X_0 = u] = K^2[u, v].$$

In my daily work Markov Chain, we have:

$$K^2 = \begin{array}{c} \\ W \\ S \\ E \end{array}\!\!\begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .4 & .6 & 0 \\ .1 & .6 & .3 \\ .5 & 0 & .5 \end{pmatrix} \end{array} \times \begin{array}{c} \\ W \\ S \\ E \end{array}\!\!\begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .4 & .6 & 0 \\ .1 & .6 & .3 \\ .5 & 0 & .5 \end{pmatrix} \end{array} = \begin{array}{c} \\ W \\ S \\ E \end{array}\!\!\begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .22 & .6 & .18 \\ .25 & .42 & .33 \\ .45 & .3 & .25 \end{pmatrix}. \end{array}$$

For example, the probability I am surfing at time 2 given that I start out working is .6 (a bit of a coincidence): either I do Work, Work, Surf (probability $.4 \times .6 = .24$) or I do Work, Surf, Surf (probability $.6 \times .6 = .36$), and that sums to $.24 + .36 = .6$. Notice that the row sums are still 1 in K^2, which makes sense (the probability I am in *some* state given that $X_0 = u$ is 1).

You might now be able to guess the answer to the following:

Question: What is $\mathbf{Pr}[X_3 = v \mid X_0 = u]$?

Answer: Again, by the Law of Total Probability, conditioning on the the state X_2:

$$\begin{aligned}
\mathbf{Pr}[X_3 = v \mid X_0 = u] &= \sum_{w=1}^{r} \mathbf{Pr}[X_2 = w \mid X_0 = u]\mathbf{Pr}[X_3 = v \mid X_2 = w, X_0 = u] \\
&= \sum_{w=1}^{r} \mathbf{Pr}[X_2 = w \mid X_0 = u]\mathbf{Pr}[X_3 = v \mid X_2 = w] \\
&= \sum_{w=1}^{r} K^2[u, w]K[w, v], \\
&= K^3[u, v],
\end{aligned}$$

where again we noticed that we were getting the expression for the (u, v) entry in the matrix product $K^2 \times K$. By induction, you can now easily prove:

Theorem 2. $\mathbf{Pr}[X_t = v \mid X_0 = u] = K^t[u, v]$. *More generally,* $\mathbf{Pr}[X_{s+t} = v \mid X_s = u] = K^t[u, v]$.

2.2 Probability vectors

In general, instead of fixing each possible initial state, we usually just allow X_0 to be a random variable itself. For example, perhaps when I begin my day, I start with Work with probability .5, start with Surfing with probability .1, and start with Email with probability .4. It's quite convenient to write these probabilities in a *probability vector*: a row vector where the entries are indexed by states. In my example, the starting probability vector is

$$\pi_0 = \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .5 & .1 & .4 \end{pmatrix} \end{array}$$

(Really, this is just the PMF of X_0, written out.) It's usual to use th letter π (for 'p' for 'probability') for probability vectors. Probability vectors always have nonnegative entries which add up to 1.

So if this is the distribution of my state at time 0, what is the distribution of my state at time t? Again, we can write the probabilities in a probability vector, call it π_t. Then conditioning on X_0 using the Law of Total Probability, we have

$$\pi_t[v] = \mathbf{Pr}[X_t = v] = \sum_{u=1}^{r} \mathbf{Pr}[X_0 = u]\mathbf{Pr}[X_t = v \mid X_0 = u] = \sum_{u=1}^{r} \pi_0[u]K^t[u, v].$$

This formula should also leap out at you:

Theorem 3. *If π_0 is the probability vector representing the initial distribution on states, and π_t is the probability vector representing the distribution on states at time t, then*

$$\pi_t = \pi_0 K^t.$$

More generally,

$$\pi_{s+t} = \pi_s K^t.$$

In my work day example, given my initial distribution π_0, the probability vector representing my distribution at time 1 is

$$\pi_1 = \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .5 & .1 & .4 \end{pmatrix} \end{array} \times \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{array}{c} W \\ S \\ E \end{array} \begin{pmatrix} .4 & .6 & 0 \\ .1 & .6 & .3 \\ .5 & 0 & .5 \end{pmatrix} \end{array} = \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .41 & .36 & .23 \end{pmatrix}. \end{array}$$

2.3 My long-term day

Given Theorem 2, we can figure out the probability I'm surfing at some late minute t given that I started out working; or more generally, all the probabilities $\mathbf{Pr}[X_t = v \mid X_0 = u]$. We just need to compute K^2, K^3, K^4, etc. For this we fire up Matlab and conclude:

$$K^2 = \begin{array}{c} \\ W \\ S \\ E \end{array} \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .22 & .6 & .18 \\ .25 & .42 & .33 \\ .45 & .3 & .25 \end{pmatrix} \end{array}$$

$$K^3 = \begin{array}{c} \\ W \\ S \\ E \end{array} \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .238 & .492 & .270 \\ .307 & .402 & .291 \\ .335 & .450 & .215 \end{pmatrix} \end{array}$$

$$K^{10} \approx \begin{array}{c} \\ W \\ S \\ E \end{array} \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .2940 & .4413 & .2648 \\ .2942 & .4411 & .2648 \\ .2942 & .4413 & .2648 \end{pmatrix} \end{array}$$

That's interesting. After 10 minutes, the rows are almost identical...

$$K^{30} \approx \begin{array}{c} \\ W \\ S \\ E \end{array} \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .29411764705 & .44117647059 & .26470588235 \\ .29411764706 & .44117647058 & .26470588235 \\ .29411764706 & .44117647059 & .26470588235 \end{pmatrix} \end{array}$$

$$K^{60} \approx \begin{array}{c} \\ W \\ S \\ E \end{array} \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .294117647058823 & .441176470588235 & .264705882352941 \\ .294117647068823 & .441176470588235 & .264705882352941 \\ .294117647068823 & .441176470588235 & .264705882352941 \end{pmatrix} \end{array}$$

Wow! It's maybe not too surprising that the matrix powers appear to be converging to a limit, but it surely can't be a coincidence that all the rows in the limit are virtually identically. If you think about it, it makes some sense though:

Interpretation: *In the long run, the starting state doesn't really matter. For large t, we basically have $\mathbf{Pr}[X_t = \text{'Work'}] \approx .294$, $\mathbf{Pr}[X_t = \text{'Surf'}] \approx .441$, $\mathbf{Pr}[X_t = \text{'Work'}] \approx .265$. Hence in a long stretch of time, the average fraction of my minutes that I'm surfing the web is* 44.1%.

3 Stationary distributions

Suppose we believe (and it seems pretty reasonable, based on the evidence!) that after I run my daily Markov Chain for many steps, my distribution on states converges towards some fixed probability vector π. In fact, from the numerical evidence it seems clear that

$$\pi = \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .294 & .441 & .265 \end{pmatrix}. \end{array}$$

Is there any way we could have figured out what π is just by looking at the transition matrix K? After all, it's kind of a drag to raise a matrix to the power of 60 by hand!

Here's one way to think about it. Suppose I start my day in any distribution on states π_0, and I follow the Markov Chain for a large number of minutes, T. At that point, my the distribution on states should basically be π,

$$\pi_T \approx \pi.$$

Well, $T+1$ is also a large number, so my distribution on states at time $T+1$ should also basically be π,

$$\pi_{T+1} \approx \pi.$$

But we know from Theorem 3 that

$$\pi_{T+1} = \pi_T K,$$

and hence we should have

$$\pi = \pi K.$$

In other words, π should be a distribution on states which *doesn't change* when you do a step.

Definition 4. *A probability vector π is called a* stationary distribution *or* steady-state *for a Markov Chain if it satisfies the* stationary equations

$$\pi = \pi K.$$

Notice that this really is a *system of equations*, if we think of the entries of π as variables and the entries of K as known constants.

3.1 An example

Let's consider my daily work MC example:

$$\pi = \pi K \quad \Leftrightarrow \quad \begin{pmatrix}\pi[W] & \pi[S] & \pi[E]\end{pmatrix} = \begin{pmatrix}\pi[W] & \pi[S] & \pi[E]\end{pmatrix}\begin{pmatrix}.4 & .6 & 0\\ .1 & .6 & .3\\ .5 & 0 & .5\end{pmatrix}$$

$$\Leftrightarrow \quad \begin{cases}\pi[W] &= .4\pi[W] + .1\pi[S] + .5\pi[E]\\ \pi[S] &= .6\pi[W] + .6\pi[S] + 0\pi[E]\\ \pi[E] &= 0\pi[W] + .3\pi[S] + .5\pi[E]\end{cases}$$

Let's solve this system! Focusing in on my favorite variable, $\pi[S]$, the second equation gives

$$.4\pi[S] = .6\pi[W] \quad \Rightarrow \quad \pi[W] = \frac{2}{3}\pi[S].$$

The third equation gives

$$.5\pi[E] = .3\pi[S] \quad \Rightarrow \quad \pi[E] = \frac{3}{5}\pi[S].$$

Finally, plugging these results into the first equation gives

$$\frac{2}{3}\pi[S] = \frac{2}{5}\cdot\frac{2}{3}\pi[S] + \frac{1}{10}\pi[S] + \frac{3}{10}\pi[S] = \frac{2}{3}\pi[S].$$

Wait a minute. We don't get anything out of this last equation. Is the system of equations underdetermined?

No! We forgot, there's one more equation: since π is supposed to be a probability vector,

$$\pi[W] + \pi[S] + \pi[E] = 1 \quad \Rightarrow \quad \frac{2}{3}\pi[S] + \pi[S] + \frac{3}{5}\pi[S] = 1 \quad \Rightarrow \quad \frac{34}{15}\pi[S] = 1.$$

Hence

$$\pi[S] = \frac{15}{34}, \pi[W] = \frac{10}{34}, \pi[E] = \frac{9}{34}.$$

And this checks out, since $\frac{10}{34} \approx .294$, $\frac{15}{34} \approx .441$, $\frac{9}{34} \approx .265$.[1]

Reminder: *When solving the stationary equations, also include the equation that the entries of π add up to 1.*

[1] In fact, we could have figured this out with Maple. Try typing `identify(.441176470588235)` into Maple; it will respond with $\frac{15}{34}$.

15-359: Probability and Computing
Fall 2009

Lecture 15: Conclusion of Finite-State Markov Chain theory

1 Stationary Distributions

In the previous lecture we introduced Markov Chains and saw an example illustrating that powers of the transition matrix K^t seemed to converge to a limiting matrix, where all the rows were the same. We also saw that in such a case, we expect that row π to represent a *stationary distribution*, or steady-state, for the Markov Chain, and saw that we could try to solve for π via the *stationary equations*,

$$\pi = \pi K$$

along with the equation saying that π's entries add up to 1,

$$\sum_{u=1}^{r} \pi_u = 1.$$

Let's take a look at stationary distributions in the general case.

1.1 In general

In the general case of a Markov Chain with r states and transition matrix K, the stationary equations

$$\pi = \pi K$$

are r equations over the r unknowns $\pi_1, \ldots, \pi_r$:

$$\pi_v = \sum_{u=1}^{r} K_{uv}\pi_u \qquad \forall\ 1 \le v \le r.$$

As we saw in the previous lecture's example, one of these r equations is *redundant*. To see this, add up all equations: you get

$$\sum_{v=1}^{r} \pi_v = \sum_{v=1}^{r}\sum_{u=1}^{r} K_{uv}\pi_u = \sum_{u=1}^{r}\sum_{v=1}^{r} K_{uv}\pi_u = \sum_{u=1}^{r} \pi_u \sum_{v=1}^{r} K_{uv} = \sum_{u=1}^{r} \pi_u,$$

since all the row-sums $\sum_{v=1}^{r} K_{uv}$ in K are 1. On the other hand, we always include the equation

$$\sum_{u=1}^{r} \pi_u = 1.$$

So this gets us back to r equations in r unknowns. Basic linear algebra tells us that such a system either has a unique solution, or infinitely many solutions (in the undetermined case, where still some of the equations are redundant). So it looks like there is *always* at least one (and possibly infinitely many) stationary distributions. This is actually true, but...

Question: Suppose the solution to the system π has some negative values. Then that would not be a valid probability vector!

Answer: True. But it is a theorem (a somewhat tricky one) that this will not happen. We will see the essence of the proof of this later, but for now, please take the following on faith:

Theorem 1. *Every Markov Chain on a finite set of states has either one stationary distribution or infinitely many.*

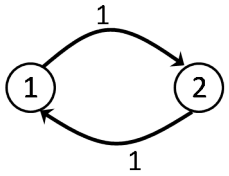
We like it very much when there is a unique stationary distribution, and tend to view the other case as "irregular". We will now discuss the somewhat "annoying" ways in which a Markov Chain can be irregular.

2 Irregularities in Markov Chains

As mentioned, we like when our Markov Chains have unique stationary distributions, and the property that K^t tends to a limit with all the rows are the same. These things mean that the chain is not really "sensitive to the initial conditions". Let's now look at some examples where this goes wrong.

2.1 Periodicity

Here is one annoying Markov Chain:



You can see that this chain just goes back and forth on each time step. In particular, the long-term behavior *is* sensitive to the initial state: If $X_0 \equiv 1$, then you will be in state 1 on the even time steps and state 2 on the odd time steps. The reverse is true if $X_0 \equiv 2$. This is also borne out by the transition matrix:

$$K = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \Rightarrow \quad K^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, K^3 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, K^4 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \text{etc.}$$

In this case, we immediately see that

$$\lim_{t \to \infty} K^t \quad \text{does not exist.}$$

That's a shame. There is an upside though: there *is a unique stationary distribution.* The stationary equations $\pi = \pi K$ yield $\pi_1 = \pi_2$, and if we combine this with $\pi_1 + \pi_2 = 1$ we get:

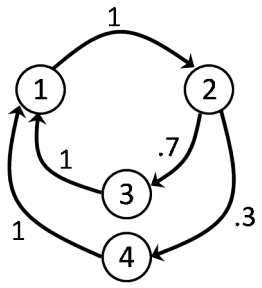
$$\pi = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix} \text{ is the unique stationary distribution.}$$

This makes sense: if you *start* this annoying random chain in the distribution

$$X_0 = \begin{cases} 1 & \text{w.p. } 1/2, \\ 2 & \text{w.p. } 1/2, \end{cases}$$

it's easy to see that distribution is maintained for all time. We will later see that this is related to the fact that in this Markov Chain, the long-term fractional time spent in each state is 1/2.

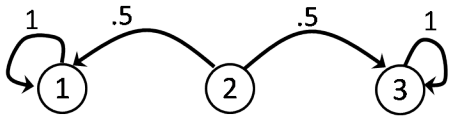
Another example of the same phenomenon occurs in, e.g., this Markov Chain:



Suppose you start at time 0 in state 1. At time 1 you will be in state 2. At time 2 you will be either in state 3 or 4. Either way, at time 3 you will be back in state 1. And then the cycle repeats. In fact, it's easy to see that whatever state you're in at time t, you'll be in the same state at time $t+3k$ for all $k \in \mathbb{N}$. This "periodicity" means that the Markov Chain *is sensitive to the initial state*. You can check that again, the limit of K^t does not exist, although there is a unique stationary distribution. This periodicity annoyance will be discussed further on the homework.

2.2 Reducibility

Here's a Markov Chain illustrating a different kind of irregularity:



It's pretty clear this Markov Chain is also sensitive to the initial state. If you start in state 1, you never leave, and the same with state 3. If you start in state 2, you have a 50-50 chance of being "absorbed" into each of the other two states. This Markov Chain is sort of the opposite of the previous one: the limit of K^t *does* exist (but the rows are not all the same), and there are infinitely many stationary distributions. To see the first statement, simply observe

$$K = \begin{pmatrix} 1 & 0 & 0 \\ .5 & 0 & .5 \\ 0 & 0 & 1 \end{pmatrix} \quad \Rightarrow \quad K^2 = \begin{pmatrix} 1 & 0 & 0 \\ .5 & 0 & .5 \\ 0 & 0 & 1 \end{pmatrix} \quad \Rightarrow \quad K^t = \begin{pmatrix} 1 & 0 & 0 \\ .5 & 0 & .5 \\ 0 & 0 & 1 \end{pmatrix} \forall t.$$

This makes sense: it says that if you start at state 2, then for *every* future time step you have a .5 chance of being in state 1 and a .5 chance of being in state 3 (this is entirely determined by your first step); and, if you start in state 1 or state 3, you'll stay there forever. Clearly, $\lim_{t\to\infty} K^t$ exists and equals K, but the fact that not all rows are the same illustrates the sensitivity to the initial conditions.

Let's try to solve the stationary equations:

$$\pi = \pi K \quad \Leftrightarrow \quad \begin{pmatrix}\pi_1 & \pi_2 & \pi_3\end{pmatrix} = \begin{pmatrix}\pi_1 & \pi_2 & \pi_3\end{pmatrix} \begin{pmatrix}1 & 0 & 0\\ .5 & 0 & .5\\ 0 & 0 & 1\end{pmatrix}$$

$$\Leftrightarrow \quad \begin{cases}\pi_1 &= & \pi_1 + .5\pi_2\\ \pi_2 &= & 0\\ \pi_3 &= & .5\pi_2 + \pi_3\end{cases}$$

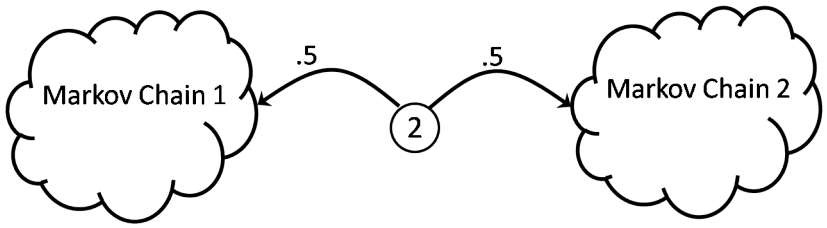
$$\Leftrightarrow \quad \pi_2 = 0$$

Adding in the equation $\pi_1 + \pi_2 + \pi_3 = 1$ only means that $\pi_1 + \pi_3 = 1$. So we have *infinitely many* stationary distributions: anything of the form

$$\pi = \begin{pmatrix}p & 0 & 1-p\end{pmatrix}.$$

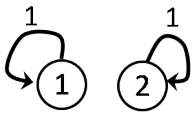
Again, if you think about it, it makes sense: you can't have any probability on state 2 in a steady-state, but you could have any split in the probability mass, p and $1-p$, between states 1 and 3, and this would stay in steady state.

The "problem" with this Markov Chain is essentially that you have two states, 1 and 3, which don't "communicate"; whichever one you get to first, you get stuck in, and you can never get to the other one. More generally, you might have something like this:



Effectively, studying this MC reduces to studying "Markov Chain 1" and "Markov Chain 2" separately. This "reducibility" annoyance will be discussed further on the homework.

Actually, here is the absolute simplest Markov Chain illustrating this issue:



Yeah, it's not even connected! So it's a silly Markov Chain to study, but it's technically allowed. Here we just have

$$K = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}.$$

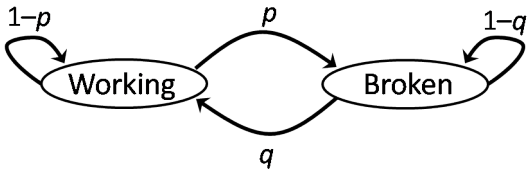
Again, $K^t = K$ for all t, and the limit exists, but the rows are not the same, and any probability vector of the form

$$\pi = \begin{pmatrix}p & 1-p\end{pmatrix}$$

is a stationary distribution.

3 The repair shop Markov Chain

There's a single, simple Markov Chain that illustrates all three kinds of behavior. It is called the *Repair shop Markov Chain*. In this Markov Chain, you have a computer, and it is either working or broken. Each day that it's working, it breaks with probability p, in which case it's sent to the repair shop. Each day that it's in the repair shop, it has a probability q of getting fixed.



$$K = \begin{pmatrix} 1-p & p \\ q & 1-q \end{pmatrix}.$$

Let's first look at the t-step transition matrices; i.e., the powers K^t.

Exercise 2. *For each $t \in \mathbb{N}$,*

$$K^t = \begin{pmatrix} \frac{q}{p+q} + \frac{p}{p+q}(1-p-q)^t & \frac{p}{p+q} - \frac{p}{p+q}(1-p-q)^t \\ \frac{q}{p+q} - \frac{q}{p+q}(1-p-q)^t & \frac{p}{p+q} + \frac{q}{p+q}(1-p-q)^t \end{pmatrix}.$$

(Hint: proof by induction, of course!)

What happens in this case as $t \to \infty$? Let's first consider the "generic" case where $0 < p < 1$ and $0 < q < 1$. In this case, we have

$$-1 < (1-p-q) < 1,$$

with *strict* inequality on both sides. Then we of course know that $(1-p-q)^t \to 0$ as $t \to \infty$. Thus it is evident that

$$K^t \xrightarrow{t\to\infty} W := \begin{pmatrix} \frac{q}{p+q} & \frac{p}{p+q} \\ \frac{q}{p+q} & \frac{p}{p+q} \end{pmatrix}.$$

This is a situation we like! The limiting matrix has its rows the same, and the resulting stationary distribution

$$\pi = \begin{pmatrix} \frac{q}{p+q} & \frac{p}{p+q} \end{pmatrix}$$

makes perfect sense: if p is the "breaking" probability and q is the "fixing" probability, it seems quite plausible that in the long-term limit, the computer will spend $\frac{q}{p+q}$ of its time working and the remaining $\frac{p}{p+q}$ of its time broken.

We can also solve the stationary equations easily:

$$\pi_W = (1-p)\pi_W + q\pi_B, \quad \pi_B = p\pi_W + (1-q)\pi_B \quad \Leftrightarrow \quad p\pi_W = q\pi_B,$$

$$\pi_W + \pi_B = 1.$$

Since neither p nor q is 0, we easily solve these to get $\pi_W = \frac{q}{p+q}$, $\pi_B = \frac{p}{p+q}$, as expected.

What about the "irregular cases"? For example, if $p = q = 1$, we get precisely the "back-and-forth" periodic Markov Chain, where K^t does not converge, but there is a unique stationary

distribution $(\frac{1}{2} \quad \frac{1}{2})$. Or, if $p = q = 0$, we get the silly Markov Chain where you just always stay Working/Broken: $K^t = K$ for all t, but there are infinitely many stationary distributions.

We also have one more somewhat degenerate case, which falls into the "reducible" category: $0 < p < 1$, but $q = 0$ (or the other way around). In this case, whenever your computer breaks, it breaks permanently and is never fixed! Here we have the kind of conclusion we normally like:

$$K^t \xrightarrow{t\to\infty} W := \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \qquad \pi = \begin{pmatrix} 0 & 1 \end{pmatrix}.$$

The powers of K converge to a matrix with the same rows, and there is a unique stationary distribution. Still, it's a bit degenerate — it just says that eventually your computer is broken!

4 The Fundamental Theorem for Finite-State Markov Chains

As we saw from the previous examples, a recurring feature of cases where things go poorly is when the matrices K, K^2, K^3, etc. contain 0's. Somehow, it's problematic when your Markov Chain has the feature that it's "impossible to go from u to v in a number of steps divisible by 3" ("periodicity"), or "impossible to go from u to v in at all" ("reducibility"). It's not necessarily so bad if K has 0's in it; after all, my daily work Markov Chain from last lecture had some 0's in it and it turned out fine. But when K^t has 0's for large t's, it's annoying.

Here's a nice observation:

Proposition 3. *If K^T has no 0's, then neither does K^{T+t} for any $t > 0$.*

In other words, if some transition matrix power is 0-less, so are all higher powers.

Proof. It suffices to prove this for $t = 1$. But just imagine the matrix product

$$K^{T+1} = KK^T.$$

Each entry in K^{T+1} is the dot-product of a row in K with a column in K^T. We know that each row in K adds up to 1, and doing a dot-product with a vector with sum 1 is like doing a "weighted average". We also know, by assumption, that each column in K^T has no 0's. So we're doing a weighted average of a bunch of strictly positive numbers. So each result will be strictly positive; i.e., not 0. □

These observations motivate the following definition:

Definition 4. *We say a Markov Chain with a* finite number of states[1] *is* regular *if there exists a t such that $K^t[u, v] > 0$ for all u, v.*

You will often need to *prove* that a Markov Chain is regular, and this can be a nice puzzle: You need to show there is some fixed number of steps t such that for every pair of states u and v, it is *possible* to go from state u to state v in exactly t steps.

Regular Markov Chains are the kind we like, due to the following:

[1] This is important.

Theorem 5. *(Fundamental Theorem for Finite-State Markov Chains.) Suppose we have a regular Markov Chain with finitely many states. Then* $W = \lim_{t\to\infty} K^t$ *exists and has all its rows equal to some vector* π *with* $\pi[u] > 0$ *for all states* u*. Further,* π *is the* unique *stationary distribution for the chain.*

The first sentence of the Fundamental Theorem is the hard part. You will see one proof of it in the recitation, and one on the homework. The "Further, ..." part of the proof is easy:

Proposition 6. *Suppose* $K^t \to W$ *where* W *has all rows equal to the same vector* π*. Then* π *is the* unique *stationary distribution.*

Proof. Suppose ρ is any stationary distribution. We will show it must equal π. Since ρ is stationary we have $\rho K = \rho$ by definition. Multiplying this equation on the right by K we get $\rho K^2 = \rho K = \rho$. Similarly, one can show $\rho K^3 = \rho$, and indeed $\rho K^t = \rho$ for all t. Since $K^t \to W$, it follows that $\rho W = \rho$.

But think about the vector-matrix product

$$\rho W = \rho \begin{pmatrix} —\pi— \\ —\pi— \\ \cdots \\ —\pi— \end{pmatrix}.$$

The vth entry of the product is just

$$\sum_{u=1}^{r} \rho_u \pi_v = \pi_v \sum_{u=1}^{r} \rho_u = \pi_v,$$

since ρ's entries add up to 1. So $\rho W = \pi$. But we just decided $\rho W = \rho$, so we must have $\rho = \pi$, as claimed. □

5 Mean first passage and first recurrence

Suppose we have a Markov Chain with r states which is regular. Let π be the (unique) stationary distribution. It turns out that π has another nice interpretation. Let's ask:

Question: Suppose the Markov Chain is in state u. What is the expected number of steps until it reaches state u again?

Definition 7. *This quantity is called the expected* first recurrence *time for* u*, and is denoted* M_{uu}*.*

Intuitive Answer: In the long-term limit, we expect to be in state u a π_u fraction of the time. Thus it seems plausible that the expected time between visits to state u should be $1/\pi_u$.

We will now show this is true! To do that, though, we also need to consider the expected time to go from one state u to a different state v.

Definition 8. *The expected time to go from state* u *to state* $v \neq u$ *is called the expected* first passage *time from* u *to* v*, and is denoted* M_{uv}*.*

Actually, you now see that we have defined a full $r \times r$ matrix M which contains the expected first recurrence times on the diagonal and the first passage times off the diagonal.

Theorem 9. *In a regular MC with finitely many states and (unique) stationary distribution π,*

$$M_{uu} = \frac{1}{\pi_u}.$$

Proof. Let u and v be any two states (possibly the same). The idea of the proof is to think about what M_{uv} is, using the Law of Total Probability and Linearity of Expectation. Use the Law of Total Probability, conditioning on the first step from u:

$$M_{uv} = \mathbf{E}[\text{time to go from } u \text{ to } v] = \sum_{w=1}^{r} K_{uw} \cdot \mathbf{E}[\text{time to get from } u \text{ to } v \mid \text{first step } u \to w].$$

There are two cases here for w: if $w = v$, then the expected time to go from u to v conditioned on the first step being v is just 1! Otherwise $w \neq v$. At that point, you have taken 1 step and you are at w. By the Markovian property, it doesn't matter than you came from u; it just matters that you are at w now. Hence:

$$M_{uv} = K_{uv} + \sum_{w \neq v} K_{uw} \cdot (1 + \mathbf{E}[\text{time to get from } w \text{ to } v]).$$

(Technically, we used Linearity of Expectation here to pull the "1+" out of the expectation.)[2] But the last expectation here is just M_{wv}! So for each pair (u, v), we have the equation

$$M_{uv} = K_{uv} + \sum_{w \neq v} K_{uw}(1 + M_{wv}).$$

It's a bit annoying to have a sum over $w \neq v$, so let's add in the missing term $w = v$, and also subtract it:

$$M_{uv} = K_{uv} - K_{uv}(1 + M_{vv}) + \sum_{w=1}^{r} K_{uw}(1 + M_{wv}) = -K_{uv}M_{vv} + \sum_{w=1}^{r} K_{uw}(1 + M_{wv}). \qquad (1)$$

Notice the sum on the right looks a bit like a matrix multiplication is going on. Let's introduce the matrix $r \times r$ matrix of all 1's and call it J. Then by definition,

$$1 + M_{wv} = (J + M)[w, v],$$

and hence

$$\sum_{w=1}^{r} K_{uw}(1 + M_{wv}) = \sum_{w=1}^{r} K[u, w](J + M)[w, v] = (K \cdot (J + M))[u, v].$$

That nicely takes care of the sum term in (1): it's the (u, v) entry of the matrix $K(J + M)$. The left-hand side of (1) is also the (u, v) entry of a matrix, M. Can we get the last term, $-K_{uv}M_{vv}$, to also be the (u, v) entry of matrix?

[2] Super-duper-hyper-technically, for this whole calculation to be valid we need to justify in advance that all these expectations are not infinite. This follows fairly easily from regularity.

To do that, let's introduce one more (!) matrix, D, which is the $r \times r$ matrix which is mostly 0's except that the diagonal entry D_{vv} equals M_{vv}. Then it's easy to see that

$$K_{uv}M_{vv} = (K \cdot D)[u, v].$$

Substituting all of this into (1) gives

$$M[u, v] = -(K \cdot D)[u, v] + (K \cdot (J + M))[u, v].$$

This holds for all entry pairs (u, v); i.e., it's really an equation on matrices:

$$M = -KD + K(J + M) = -KD + KJ + KM.$$

Finally, we get π into the picture by a lovely trick: just multiply both sides of this equation on the left by π:

$$\pi M = -\pi KD + \pi KJ + \pi KM.$$

But π is the stationary distribution, and the stationary equation is $\pi K = \pi$! Hence:

$$\pi M = -\pi D + \pi J + \pi M \quad \Leftrightarrow \quad \pi D = \pi J.$$

Remember now J is the matrix of all 1's. Since π's entries add up to 1 we get that πJ is just the all-1's row vector:

$$\pi D = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}.$$

On the other hand, D is just a diagonal matrix where the D_{uu} entry is M_{uu}. So

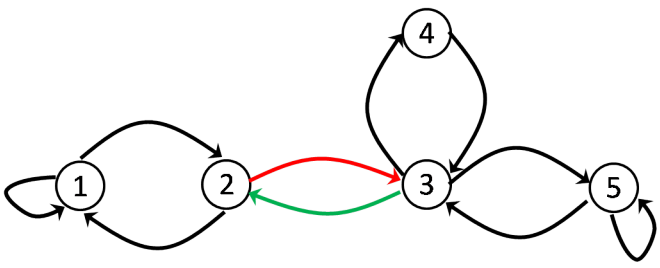
$$\pi D = \begin{pmatrix} \pi_1 M_{11} & \pi_2 M_{22} & \cdots & \pi_r M_{rr} \end{pmatrix}.$$

We conclude that $\pi_u M_{uu} = 1$ for all states u; i.e., $M_{uu} = 1/\pi_u$. □

6 Time reversibility

We know that for a regular Markov Chain, the limit of K^t is a matrix where all rows are the same, π. We also saw that there was a relatively easy way to calculate π (easier than computing the limit K^t): solve the stationary equations. What if we're super-lazy: is there an even easier way? Turns out that sometimes, there is!

Let's take a look at the following Markov Chain. We haven't labeled the arcs with probabilities, but assume they have some nonzero probabilities. The Markov Chain will then be regular (you can try to prove this), and hence have a unique stationary distribution π.



I want you to focus on the red arc from state 2 to state 3. Suppose I ran the chain for 10^6 steps. How many times, in expectation, would you expect to move along the red arc?

Well, fix a t which is somewhat large. The probability of being in state 2 at time t will be close to π_2. And conditioned on being in state 2, the probability you step along the red arc is K_{23}. In other words, for all decently large t we have

$$\mathbf{Pr}[t \to (t+1) \text{ step is the red arc}] \approx \pi_2 K_{23}.$$

Hence by the Linearity of Expectation + Indicators Method, we expect

$$\pi_2 K_{23} \cdot 10^6 \textit{ transitions along the red arc.} \tag{2}$$

Fine, that's some number. Suppose it's $135,524$.

Question: If there are $135,524$ transitions along the red arc, what can you say about the number of transitions along the green arc, from state 3 to state 2?

Answer: If you think about if for a second, you see it *must* be either $135,523$, $135,524$, or $135,525$; i.e., it has to be within 1. The reason is simple. Suppose you take a "red step". Now it's impossible for you to take another "red step" again unless you take (exactly) one green step. You just can't get back to state 2 without going through state 3. Similarly, you can't take multiple "green steps" without taking a "red step" between each one. So the number of red steps and green steps has to differ by either 0 or 1 in any run of the chain!

But wait. The exact same reasoning that led us to (2) would lead us to expect

$$\pi_3 K_{32} \cdot 10^6 \textit{ transitions along the green arc.} \tag{3}$$

But we know that on any run, the number of green and red transitions differs by at most 1. Thus it seems we must have

$$\pi_2 K_{23} \cdot 10^6 \approx \pi_3 K_{32} \cdot 10^6,$$

which seems to force

$$\pi_2 K_{23} = \pi_3 K_{32}. \tag{4}$$

It's actually quite easy to make this argument completely rigorous; indeed, in this Markov Chain, (4) must hold. This equation is saying, "*in steady-state, the rate of transitions from* 2 *to* 3 *equals the rate of transitions from* 3 *to* 2".

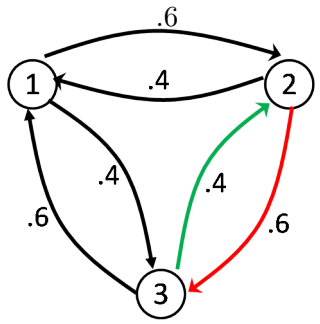
In fact, if you look at the picture of the MC you can see that the same reasoning holds for *every* pair of adjacent states, hence (4) holds for every pair of adjacent states. Indeed, (4) also holds (trivially) for every *other* pair of states (for a state and itself, it's trivial; for disconnected states it's trivial because both sides are 0). I.e.:

$$\pi_u K_{uv} = \pi_v K_{vu} \quad \forall\ u, v. \tag{5}$$

Definition 10. *For Markov Chains having a unique stationary distribution π, we say that the Markov Chain is* time reversible *if the equations* (5) *hold.*

The reason for the terminology is that runs of time reversible Markov Chains are indistinguishable from runs in reverse!

Please note that *not every chain* is time reversible. For example, this one is *not*:



The reasoning about red and green transitions breaks down in this case, because by going around the cycle, you can do many red transitions without doing any green transitions (and vice versa). Please note that the fact that our reasoning broke down does *not prove* the chain is not time reversible. For that, you'd first need to prove that $\pi = \left(\frac{1}{3}\frac{1}{3}\frac{1}{3}\right)$ is the unique stationary distribution, and then observe that, e.g., $\pi_2 K_{23} = .2 \neq .1333\ldots = \pi_3 K_{32}$.

Finally, why the interest in time reversibility? If the time reversibility equations hold for some probability vector π, then π must be a stationary distribution! This is easy to prove:

Theorem 11. *Suppose we have an r-state Markov Chain with transition matrix K. Suppose that π is a probability vector satisfying the time reversibility equations* (5)*. Then π is a stationary distribution.*

Proof. All we need to do is check that the stationary equations hold:

$$\begin{aligned}
(\pi K)[v] &= \sum_{u=1}^{r} \pi[u]K[u,v] \quad \text{(formula for vector-matrix multiplication)} \\
&= \sum_{u=1}^{r} \pi[v]K[v,u] \quad \text{(time reversibility equations (5))} \\
&= \pi[v] \sum_{u=1}^{r} K[v,u] \\
&= \pi[v] \quad \text{(row-sums of a transition matrix equal 1),}
\end{aligned}$$

and hence indeed $\pi K = \pi$. □

Hence as a practical matter:

Recipe: Suppose you are given a finite-state Markov Chain and asked to find a stationary distribution. *First try solving the time reversibility equations* (5) *(along with equation* $\sum_{u=1}^{r} \pi[u] = 1$*.* These equations are very frequently much easier to solve than the stationary equations. However, not all Markov Chains are time reversible. So if there is no solution to the time reversibility equations (i.e., you encounter a contradiction), then you have to fall back on the stationary equations (which are always solvable).

15-359: Probability and Computing
Fall 2009

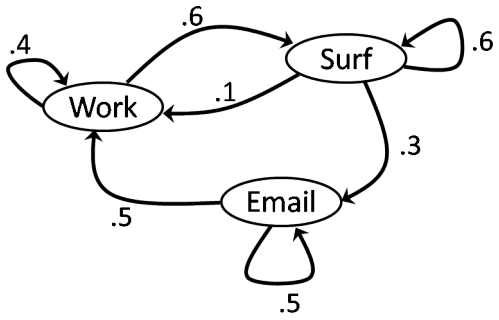
Lecture 16: Markov Chain applications: PageRank and Metropolis

In this lecture we will discuss some applications of Markov Chains in computing. One great application comes from *queueing theory*, an important area of operations research and performance modeling.[1] For now, we will only talk about queueing on the homework. But because the most natural basic scenario in queuing theory involves an *infinite-state* Markov Chain, we will talk a little bit about the theory of infinite-state Markov Chains

1 Finite versus infinite Markov Chains

1.1 Finite MC theory recap

Let's recall what we know about finite-state Markov Chains. First, they look something like this:



Associated to each one is a square *transition matrix*; in the above case,

$$K = \begin{array}{c} \\ W \\ S \\ E \end{array} \begin{array}{c} \begin{array}{ccc} W & S & E \end{array} \\ \begin{pmatrix} .4 & .6 & 0 \\ .1 & .6 & .3 \\ .5 & 0 & .5 \end{pmatrix} \end{array}$$

The tth power of this matrix is called the *t-step transition matrix*:

$$K^t[u,v] = \mathbf{Pr}[\text{going from state } u \text{ to state } v \text{ in exactly } t \text{ steps}].$$

Alternative, a Markov Chain is defined as a sequence of random variables $X_0, X_1, X_2, \ldots$ satisfying the *Markov property*,

$$\begin{aligned} \forall t, \quad \forall u_0, u_1, \ldots, u_t \in S, \qquad & \mathbf{Pr}[X_t = u_t \mid X_{t-1} = u_{t-1}, X_{t-2} = u_{t-2}, \ldots, X_1 = u_1, X_0 = u_0] \\ = \quad & \mathbf{Pr}[X_t = u_t \mid X_{t-1} = u_{t-1}] \\ =: \quad & K[u_{t-1}, u_t]. \end{aligned}$$

[1] The hardest part of queueing theory? Spelling 'queueing'.

The random variable X_t represents the state at time t.

In many cases, the Markov Chain is not sensitive in the long run to the initial distribution on states; i.e., on the distribution of X_0. This occurs when $\lim_{t\to\infty} K^t$ converges,

$$K^t \xrightarrow{t\to\infty} W,$$

to a matrix in which each row is equal to the same (probability) vector π. This π is a *stationary distribution*; in general, a stationary distribution is any probability vector π satisfying the stationary equations

$$\pi = \pi K \qquad (\text{and } \sum_{u=1}^{r} \pi[u] = 1, \text{ of course}).$$

We identified a number of conditions on a Markov Chain which could prevent this limiting case from happening:

- "Periodicity": e.g., you can only go from u to u in a number of steps which is divisible by 7.
- "Reducibility": e.g., it is impossible to get from u to v.

These terms will be defined more precisely on the homework. We also identified a condition which *ensures* the limiting case will happen:

Definition 1. *We say a Markov Chain with a* finite number of states *is* regular *if there exists a t such that $K^t[u, v] > 0$ for all u, v.*

Theorem 2. *(Fundamental Theorem for Finite-State Markov Chains.) Suppose we have a regular Markov Chain with finitely many states. Then $W = \lim_{t\to\infty} K^t$ exists and has all its rows equal to some vector π with $\pi[u] > 0$ for all states u. Further, π is the* unique *stationary distribution for the chain.*

Finally, we also saw other potential features of the stationary distribution:

Theorem 3. *Suppose we have a regular finite-state MC with (unique) stationary distribution π. The "mean first recurrence time for u", i.e., the expected number of steps to go from state u back to state u, is*

$$M_{uu} = \frac{1}{\pi_u}.$$

Theorem 4. *Suppose π is a probability vector satisfying the "time reversibility equations",*

$$\pi_u K_{uv} = \pi_v K_{vu} \quad \forall\ u, v.$$

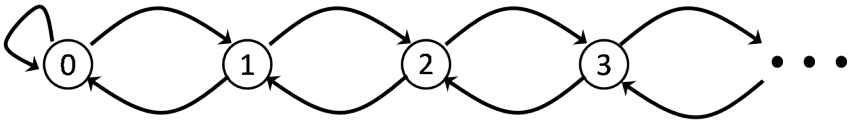
Then π is a stationary distribution.

1.2 Infinite-state Markov Chains

What about Markov Chains with a countably infinite number of states? The theory of these, unfortunately, gets a little complicated.[2] Still, it will be very useful to allow this case, just as we find it very useful to allow countably infinite sample spaces in basic probability theory.

At first it doesn't look so bad. An infinite-state Markov Chain might look something like this:

[2]Though still vastly simpler than the case of uncountably infinite states, something we will not discuss in this class.



(I haven't specified the probabilities on the arcs here, but the sum of the probabilities going out of each node should be 1, as usual.) The transition "matrix" K is a little strange in this case because it has infinitely many rows and columns — that might be a tiny bit worrisome, but it doesn't immediately cause any problems. Indeed, you can still calculate the powers K^t in the usual way, and they have the usual interpretation.

An infinite-state Markov Chain can have the usual annoying properties. For example, it can be "reducible" (for example, if the arrow from state 0 to state 1 above has probability 0) or "periodic" (for example, if the self-loop on state 0 above has probability 0, then you can only go from 0 to 0 in an even number of steps). However, there are additional ways in which an infinite-state Markov Chain can go haywire. For example, suppose that in the above MC, all the right-ward transitions have probability .9 and all left-ward transitions have probability .1. In this case, when you run the Markov Chain you'll tend to wander further and further off to the right. For any fixed state u, in the limit of the run you will have 0 probability of being at u, because you'll eventually wander off to the right of u. But you having probability 0 for every state is not a valid limiting distribution!

In short, *stationary distributions do not always exist.* We need a more complicated condition to get nice limiting properties:

Definition 5. *An infinite-state Markov Chain is called* regular *if it is "aperiodic", "irreducible", and you can prove that the mean recurrence time M_{uu} for each state u is* finite.

For infinite-state Markov Chains which are regular, the Fundamental Theorem holds, as does Theorem 3.

However, I promise not to harass you about whether infinite-state Markov Chains are regular. *In this course, I will only give you regular infinite MC's, and you can assume the conclusion of the Fundamental Theorem.*

2 Google and PageRank

2.1 On search engines

The year was 1997. Bill Clinton was in the White House. A computer (Deep Blue) beat the chess world champion (Garry Kasparov) for the first time ever. *Tubthumping* by Chumbawamba was topping the charts. And the Internet kind of sucked.

More precisely, finding anything on the Internet was pretty painful. You probably don't remember a time before Google, but trust me, it was bad. You had your Infoseek, your Excite, your Lycos[3], your AltaVista. There was also Yahoo!, which at the time wasn't really a search engine; rather, it was a huge, human-categorized directory of a fraction of all web pages (of which there were maybe 50 million back then).

[3]Created by Michael Mauldin of CMU, it was still the #1 most visited site on the web in 1999.

AltaVista was probably the biggest search engine (it answered about 20 million queries per day back then) and the best, but it was still pretty bad. Let me put it to you this way: in November 1997, only one of the top four commercial search engines actually *found itself* when you searched for it. No joke.

Question: What was the problem?

Answer: The problem was that search engines worked pretty much exclusively by matching words. If you searched for 'Harvard', would you get `www.harvard.edu`? No. Instead you would get the web pages that contained the word 'Harvard' the most times. Perhaps `www.harvard.edu` contained the word 'Harvard' 5 times, whereas some other random pages might have it 50 times for some bizarre reason. So `www.harvard.edu` would be buried in the results. As another example, the top search result for 'Bill Clinton' back then was the "Bill Clinton Joke of the Day" website. More seriously, spammers and advertisers could easily exploit this system, by making pages containing key words and phrases hundreds of times.

Question: How could one fix this?

Answer: An obvious improvement is to instead collect up all of the pages which have a decent textual match, and then *rank* all of these pages by some measure of "quality" or "authority". Yahoo! was kind of doing this by hand back then, but that approach is obviously not scalable.

Enter two groups: 1. Jon Kleinberg, a computer science professor at Cornell, who had spent some time thinking about the problem while visiting IBM Research. 2. Larry Page and Sergey Brin, two computer science Ph.D. students at Stanford with an interest in information retrieval.

These two groups simultaneously and independently came up with (roughly) the same brilliant idea for solving the problem. (In fact, they met and compared notes in the summer of 1997.) Larry Page and Sergey Brin took the idea and founded Google, making billions. Kleinberg took the idea, wrote a number of academic papers, and won the MacArthur Genius Prize and the Nevanlinna Prize (i.e., the Fields Medal for Computer Science). So you see, it was a pretty good idea.[4] We will discuss Google's variant, which is called *PageRank*.

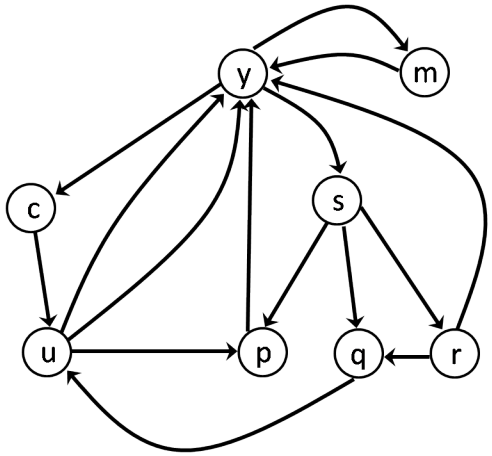
Question: It's called PageRank because it's about ranking web pages, right?

Answer: No. It's because Larry Page named it after himself!

2.2 PageRank

How can we try to measure the quality or authority of a web page in an automatic way? Simple textual analysis had hit a limit, and it was a bad one. The basic idea of Kleinberg and Brin–Page is to use *hyperlink analysis*; i.e., take into account the directed graph structure of the World Wide Web. Maybe a (piece of) the Web looks like this:

[4] As Kleinberg wrote in his original paper, the idea was not even completely new. Social scientists studying the graph of research papers/citations had been thinking of the same issues, and one 1976 paper on the topic by Pinski and Narin had roughly the same idea. Larry Page was also influenced by the Pinski–Narin paper.



How can these links tell you something about the importance of a page?

Idea 1: "Citations". Maybe, as in academic publishing, it's a good idea to think of each link to a page y as a "citation", or a "vote of quality". Perhaps we should rank the web pages by their *in-degree*.[5]

Problems: One immediate problem with this is spamming. If you want to make your web page very highly ranked, just create 100 bogus web pages and have them all link to yours. Somehow, the method needs to detect the fact that the 100 bogus pages's votes shouldn't count (much).

A less obvious equally important problem is this: Suppose a web site on a specialty topic has in-degree only 1, but the web page linking to it is a high-level page on Yahoo!'s directory. Yahoo! had a pretty authoritative directory at the time, and that single link should somehow be worth more than 5 links from more obscure web sites.

More along the lines of the first problem is that a web page might be a little to non-discriminating in its taste. For example, maybe you can't obtain 100 web pages, but you can easily add a link to each of your 100 friends' websites. It seems like a website being able to give a whole "point" to each of its friends so easily is not desirable.

Idea 2: Fractional votes. One way to at least solve the last problem here is to switch to "fractional" citations, or votes. Specifically, if a web page has d outgoing links, perhaps each page it links to should only get $1/d$ of a citation. Take the above example, and suppose we are concerned with the importance of page y. Idea 2 suggests we should look at all the pages that link to y, and let each of them give it a fractional vote. In the above picture, r, m, u, and p would contribute 1/2, 1, 2/3, and 1, respectively. This idea of fractional votes is a good one, but it does not solve the other two problems mentioned.

Idea 3: A recursive definition. Here's a crucial idea that seems to take care of most problems. Let's say we want to decide on the "importance" of web page y. Idea 2 suggests that if web page x has d outgoing links, one of which goes to y, then this should contribute $1/d$ to the importance of x. But we ought to take into account the importance of x! I.e., we should weight x's contribution by *its* importance.

[5]Larry Page called the links coming *into* a web page its "back links". And indeed, he originally called the Google project "BackRub". What a terrible name!

This is a recursive definition, which seems like it could be a problem! But with some suggestive notation, we'll know just what to do. Suppose we try to write π_v for the importance of web page v. For each other page u, it has some fraction of its outlinks which point to v. Let's call this K_{uv}. Then our recursive definition requires

$$\pi_v = \sum_u \pi_u K_{uv}.$$

Look familiar?!

2.3 The random surfer model

Obviously, the diagram of the Web and the equation above are very suggestive. The interpretation is clear: We have a (finite-state) Markov Chain, where the states are web pages and the transitions are links. At each time step, we transition according to a random link on the page. This became dubbed the "random surfer model". The importance, or "PageRank" of a page is nothing more than its probability mass under the *stationary distribution*; i.e., the long-term fraction of time a random surfer is at the page.

There are still a few problems with the model!

Problem 1: Dangling nodes. Actually, the Web *cannot* be thought of as a Markov Chain, because some pages have *no* outgoing links. Such pages are called "dangling nodes", or "dead ends". They are an important issue in practice, because they actually also occur when a page has outgoing links which *haven't been crawled yet*. The way this issue is handled is the following: for each dangling node, *we pretend that it has a link to every other web page*. Alternatively, you can imagine that when the random surfer gets "stuck", s/he goes to the URL bar and hops to a random page on the Web. Notice that this fix hardly changes anything; if there are 100 billion pages on the web, the dead end now contributes only $1/10^{11}$ importance to each of them.

Problem 2: Rank sinks. Another problem the random surfer might encounter is a *rank sink*. This is a group of pages which only have links to each other; once you go in, there is no way to escape. This is actually a major pitfall in practice: if a spammer plunks down a big "clique" of pages and gets at least one in-link from the rest of the web, it can generate major importance. The Brin–Page solution to this is:

Scaling/damping. Google has a parameter "α" called the *scaling* parameter (originally called the *damping* parameter). In the original paper, Brin and Page said they used $\alpha = .85$. Roughly, the importance a page derives from its in-links is damped by the factor α. More precisely, we change the random surfer model as follows:

Random surfer model:
On each step, with probability α follow a random link on the current page,
and with probability $1 - \alpha$ go to a uniformly random page on the web.
(If you are on a dead-end, *always* go to a uniformly random page.)

In other words, at each step, the surfer gets bored with probability $1 - \alpha$ and hops to a totally random page.

In addition to fixing practical problems, the scaling technique is extremely helpful theoretically. Notice that the effect of scaling is to replace transition probability K_{uv} with $\alpha K_{uv} + (1-\alpha)/r$, where r is the total number of pages (states). In particular, every entry of K becomes positive; i.e., K becomes regular! Thus the Fundamental Theorem applies: there is a unique stationary distribution π, giving the ranks of web pages; and, the random surfer's distribution on pages converges to π in the limit.

2.4 In practice

That's it! That's the whole PageRank idea, which got Google started and made them billions. Unsurprisingly, there were subsequently hundreds of tweaks to it, most of which became trade secrets we don't know about. But the originally method proved to give outstandingly good ranking of search results, and to be surprisingly resistant to spam attacks.

One obvious thing we've overlooked:

Question: How does Google *compute* the PageRank? Assume they've crawled the whole web and know the entire link structure. Do they... solve the stationary equations?

Answer: No. The standard way of solving a system of equations in r variables takes $O(r^3)$ time. If $r = 10^{11}$ as is it now, this is 10^{33} time, which is completely infeasible. Instead, they simply simulate the random surfer, starting from a uniformly random state. I.e., they start out π as uniform, $\pi_u = 1/r$ for each u, and then compute πK, πK^2, πK^3, etc. These are matrix-vector multiplications, which you would think take $O(r^2)$ time. But in practice, K is a *sparse* matrix; each web page only has a few links, so the number of nonzero entries in K is more like $O(r)$, not $O(r^2)$.[6] So you can compute each new vector πK^t in linear time.

But how large does t need to be before this πK^t converges to close to the stationary distribution? There is a great deal of theoretical work on this issue, but in practice, the answer seems to be that the convergence is extremely fast, and only 50 to 100 powers are necessary.

Still, the whole process might take 10^{14} steps, then. The original Brin–Page paper reported that computing the PageRank for their 20 million page data set took about 3 hours on a fast computer. Of course, there are way more pages on the web now, and even today it is reputed that Google only (re)computes PageRank every week or two weeks.

3 The Metropolis Algorithm

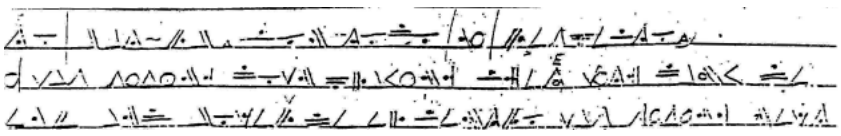
Here's a seemingly backwards question: Suppose you have a set of states S, and you are *given* a desired stationary distribution π for them. Can you *construct* a Markov Chain on S which has the desired stationary distribution?

The "Metropolis Algorithm" is designed to do just this. It was not named after Superman's home town, but rather after physicist Nicholas C. Metropolis, who coauthored a paper about it in 1953.

[6] We mean here, *before* you do the scaling/damping. You can account for the effect of scaling separately, easily.

3.1 Cryptanalysis of simple substitution ciphers

Let's discuss a real-world example, reported by eminent Stanford probabilist Persi Diaconis. One day, a psychologist from the California state prison system came to Diaconis's office. The psychologist had some coded messages the prisoners had written and asked for Diaconis's help in deciphering them. Each message was a long sequence $s_1, \ldots, s_n$ of funny symbols, like so:



Diaconis noted that there were 28 distinct symbols[7], and correctly guessed that the messages were "simple substitution ciphers". This means that the message was encrypted with some one-to-one map

$$u : \{\text{code symbols}\} \to \{a, b, c, d, \ldots, z, ., \sqcup\},$$

where $\sqcup$ means space. How did Diaconis and his students decode the messages?

Idea 1: Letter frequencies. (Not the actual solution.) Get some huge corpus of English text; Diaconis's student downloaded *War and Peace* from Project Gutenberg.[8] Compute the frequency of each letter: $p(a)$ is the fraction of a's in the text, $p(b)$ the fraction of b's, etc., up to $p(\sqcup)$, the fraction of $\sqcup$'s (spaces). Based on this, you might try to pick out the "likeliest" encryption u, which is the one maximizing

$$\prod_{i=1}^{n} p(u(s_i)).$$

It's pretty obvious that you maximize this by having u map the most frequent symbol to the highest frequency letter (probably $\sqcup$), the second most frequent symbol to the second-highest frequency letter (probably e), etc.

The problem with this solution is that it just won't work. You'll surely get gibberish.

Idea 2: Bigram frequencies. (This is the actual solution Diaconis and his students used.) Using the English corpus, compute the frequency of each *bigram*; i.e., pair of consecutive letters. I.e., $p_2(c, h)$ should be the fraction of consecutive letter pairs in the corpus which are c followed by h. There are 28^2 numbers $p_2(\ell_1, \ell_2)$ to compute here. Based on this, try to pick out the "likeliest" encryption u, which is the one maximizing

$$z(u) = \prod_{i=1}^{n-1} p_2(u(s_i), u(s_{i+1})).$$

This actually works amazingly well, as it turns out. There are a couple of issues, though:

Issue 1: Now how do you compute the likeliest u; i.e., the mapping with highest $z(u)$? If you think about it for a while, you can solve it using dynamic programming in $O(n^2)$ time. That's slightly tricky, though, and if you want to do *trigrams* it'll take $O(n^3)$ time...

[7] Okay, actually, there were actually about 40, but let me simplify things slightly here.

[8] If you're going to read *War and Peace*, by the way, don't download it form Project Gutenberg; you'll get the horrible translation by Maude & Maude.

Issue 2: It might be that the *likeliest* u is not correct, but some *pretty likely* u is correct. So we might not want to find the likeliest u per se; instead, we might want to *sample from the pretty likely u's*. I don't believe this can be done with dynamic programming.

Ideal: What would be great, then, would be to find an algorithm which *outputs u with probability proportional to $z(u)$*. If we had that, we could run it a bunch of times and eyeball the answers.

Luckily, the *Metropolis Algorithm* can do exactly this task. Diaconis and his students coded it up, and cracked the messages. E.g.,

```
to batrb.  con todo mi respeto.  i was sitting down playing chess with danny de
emf and boxer de el centro was sitting next to us.  boxer was making loud and loud
voices so i tell him por favor can you kick back homie cause im playing chess a minute
later the vato starts back up again...
```

3.2 Optimization and sampling with Metropolis

Let's generalize slightly. The task we have is the following: We have a *huge* set S of possible solutions. In the cryptanalysis scenario, there are 28! possible solutions, u. Each solution $u \in S$ has some positive *value*, which we denote $z(u)$. In the cryptanalysis scenario, this was the "likelihood" defined in terms of bigram frequencies. It should be easy to compute $z(u)$ given u. One natural goal is to try to find the u maximizing $z(u)$. But as described, we'd rather focus on a slightly different goal:

Goal: *Sample* from the solutions u, with probabilities *proportional* to the values. I.e., find a way of drawing a random sample according to the probability vector π defined by

$$\pi[u] = z(u)/Z, \quad \text{where } Z = \sum_{u \in S} z(u);$$

i.e., output u with probability $\pi[u]$.

Observation 1: The vector π is hugely long; it has 28! entries in the cryptanalysis scenario. So you're never going to be able to store it. You still might somehow be able to sample from it, though.

Observation 2: For a particular solution u, *you have no hope of computing* $\pi[u]$. The reason is that computing the normalization factor Z is almost certainly intractable. I mean, if there are 28! different solutions u, it would take forever to add up all their values.

Observation 3: On the other hand, given two solutions u and v, it's easy to compute the *ratio* $\pi[u]/\pi[v]$. That's because this is just $z(u)/z(v)$, and you can easily compute $z(u)$ and $z(v)$.

So we have very little to go on here; there is some huge mystery probability vector π, we want to sample a u with probability $\pi[u]$, but we can only compute *ratios* like $\pi[u]/\pi[v]$. The idea behind the Metropolis Algorithm is to set up (implicitly) a Markov Chain with state set S, and somehow ensure that its stationary distribution is π. If we can do this, we can (approximately) sample from π just by running the Markov Chain for a decent number of steps. (In the cryptanalysis problem, Diaconis and co. cracked the ciphers after running the chain for just a few thousand steps.)

We said "implicitly" above because the state set S is so huge, we won't actually be able to store it. Still, we need some way of getting around in S.

Assumption: There is an implicit *navigation graph*[9] G whose vertex set is S. The graph G should be undirected and connected, and all its vertices should have the same degree. (We will relax these assumptions in the homework.) Finally, give a vertex/state $u \in G$, it should be easy to figure out the neighbors of u.

In the cryptanalysis example, Diaconis and students used the following simple navigation graph: For each state (i.e., encoding) u, the neighbors of u are all encodings you get by switching a pair of symbols. I.e., each encoding u has $\binom{28}{2}$ neighbor-encodings v.

Given a navigation graph, the Metropolis Algorithm is nothing more than the following Markov Chain:

The Metropolis Algorithm:

Start in some arbitrary state U_0.

`for` $t = 1, 2, 3, \ldots$

Pick a random neighbor v of U_{t-1} in the navigation graph.

Compute the "acceptance ratio" $A = \pi[v]/\pi[U_{t-1}] = z(v)/z(U_{t-1})$.

If $A > 1$, set $U_t \leftarrow v$.

If $A \leq 1$, set $U_t \leftarrow v$ with probability A. Otherwise, set $U_t \leftarrow U_{t-1}$.

Remark 1: In the algorithm, if the neighbor you choose has a higher value, you go to it. But even if it has a lower value, you have a positive probability of going to it (since all values $z(u)$ are positive). Since G is connected, it follows that every state in this chain can reach every other state. If you're worried about periodicity, you can slap a self-loop onto every vertex in the navigation graph. In this way, the resulting Markov Chain will be provably regular.

Remark 2: Notice that we only needed to be able to compute *ratios* of values. This computation is often extremely easy, especially if U_{t-1} and v are somehow "close" or "similar". In the cryptanalysis scenario, e.g., you are only changing the mapping on two symbols, so the ratio of the values only depends on the counts of bigrams in the message that use those two symbols.

Remark 3: Often you can be smart about how you choose the initial state U_0. In the cryptanalysis scenario, e.g., it makes sense to start with the mapping u which is likeliest from the single-letter-frequency point of view.

So we have a nice, regular Markov Chain; all we want to do now is make sure its (unique) stationary distribution is π. We could try to check that π satisfies the stationary equations, but we can check something easier:

Theorem 6. *The probability vector π solves the time reversibility equations.*

As we saw last lecture, this proves that π is the unique stationary distribution.

Proof. Suppose u and v are different states. If $\pi[v] \geq \pi[u]$, then $K[u, v] = 1$, but then also $K[v, u] = \pi[u]/\pi[v]$. Hence $\pi[v]K[v, u] = \pi[u] = \pi[u]K[u, v]$, as required. The case that $\pi[v] \leq \pi[u]$ is very similar. □

[9] Note: nonstandard terminology.

15-359: Probability and Computing
Fall 2009

Lecture 17: Intro to Continuous Random Variables

Today's lecture is very exciting: we *finally* get to talk about picking a random real number between 0 and 1; more generally, about *continuous* random variables.

1 Continuous random variables: a brave new world

Almost every programming language has a built-in (pseudo)random number generator that *supposedly* returns a random real number between 0 and 1. We will now have one too, and will allow experiments (randomized code) containing lines such as

$$X \leftarrow \text{Unif}[0,1].$$

This random number X is supposed to be "uniformly" distributed on the interval $[0,1]$.[1] Intuitively, what this means is that

$$\mathbf{Pr}[X \leq .3] = .3, \quad \mathbf{Pr}[X \geq .9] = .1, \quad \mathbf{Pr}[a \leq X \leq b] = b - a, \tag{1}$$

etc.

Unfortunately, once you have random variables with uncountably many values, it's quite technical if you want to give formal, non-contradictory definitions. Fortunately, for 99% of cases, things are not so bad; it's only some extremely wild and wacky obscurities that make trouble. **For this course, I promise not to harass you about any super-bizarre mathematical arcana.** In exchange for this, you'll have to let me cheat at the math ever-so-slightly on some occasions.

Still, even to handle the 99% normal cases, things get a little counterintuitive (for example, events with probability 0 *can* happen!) and a little mathematically more advanced (hello, calculus!). For example, we can't really draw probability trees anymore: even a single call to Unif$[0,1]$ would produce a tree with uncountably many branches.

Why bother with continuous random variables, then? Well for one, it's quite natural to want to have them: it seems reasonable and desirable to be able to say things like, "The time it takes for a packet to be transmitted is a uniformly random number of milliseconds between 0 and 1." (On the other hand, a picky quantum physicist might say, "Actually, although time seems continuous, it actually occurs in discrete, Planck Time steps.") But the real reason is this: despite their difficulties, continuous random variables ultimately actually *simplify* reasoning about discrete random variables! Let's see this observation in action...

[1] Please recall "interval" notation. $[a,b]$ means the set $\{x \in \mathbb{R} : a \leq x \leq b\}$. Also $(a,b) = \{x \in \mathbb{R} : a < x < b\}$, $(a,b] = \{x \in \mathbb{R} : a < x \leq b\}$, $[a,\infty) = \{x \in \mathbb{R} : a \leq x\}$, etc.

2 Uniform random variables

Let's think about "uniform random variables", and build up slowly to what the right definitions should be.

2.1 Discrete approximation and events

As mentioned, almost every programming language claims to implement the random number generator Unif$[0,1]$. Of course it doesn't; setting aside the question of pseudorandomness, computers simply do not store or work with real numbers of unbounded precision. So really, every such program language is doing something like this:

$$X \leftarrow \text{RandInt}(N) \cdot \frac{1}{N}$$

for some unbelievably huge number N; e.g., $N = 2^{64}$.

It seems like for almost all intents and purposes, such an X should act like the desired "uniform on $[0,1]$ random variable". And since we know how to reason about with discrete random variables, why not just work with these discrete approximations? you might say.

Let's first check that the events we looked at in (1) are as expected.

$$\begin{aligned} \mathbf{Pr}[X \le .3] &= \mathbf{Pr}\left[X \in \{\tfrac{1}{N}, \tfrac{2}{N}, \tfrac{3}{N}, \ldots, \tfrac{\lfloor .3N \rfloor}{N}\}\right] \\ &= \mathbf{Pr}\left[\text{RandInt}(N) \in \{1, 2, 3, \ldots, \lfloor .3N \rfloor\}\right] \\ &= \frac{\lfloor .3N \rfloor}{N}. \end{aligned}$$

Hmm. That 'floor' is slightly annoying, but we certainly know that

$$.3N - 1 \le \lfloor .3N \rfloor \le .3N,$$

and so

$$.3 - \tfrac{1}{N} \le \mathbf{Pr}[X \le .3] \le .3.$$

That's just about right! Since $1/N$ is unimaginably small if $N = 2^{64}$, say, we can feel good about saying $\mathbf{Pr}[X \le .3] \approx .3$. We will more casually say

$$\mathbf{Pr}[X \le .3] = .3 \pm O(\tfrac{1}{N}).$$

As a little exercise, I encourage you to check that

$$.1 \le \mathbf{Pr}[X \ge .9] \le .1 + \tfrac{1}{N}; \qquad \text{i.e., } \mathbf{Pr}[X \ge .9] = .1 \pm O(\tfrac{1}{N}).$$

When you analyze $\mathbf{Pr}[a \le X \le b]$, you will get both an annoying ceiling and an annoying floor, but still:

$$b - a - \tfrac{1}{N} \le \mathbf{Pr}[a \le X \le b] \le b - a + \tfrac{1}{N}; \qquad \text{i.e., } \mathbf{Pr}[a \le X \le b] = b - a \pm O(\tfrac{1}{N}).$$

So everything looks approximately fine, although these floors and ceilings and $O(1/N)$'s are a bit annoying.

2.2 Probability 0?

On the other hand, suppose u is a real number and we ask about $\mathbf{Pr}[X = u]$. There are two cases: i) u is not of the form w/N for an integer $0 < w \leq N$; in this case, the probability is 0. ii) u *is* of this form; in this case the probability is $1/N$. But this is exactly the quantity we've been neglecting! This seems to suggest that for $U \sim \text{Unif}[0, 1]$, we ought to have

$$\mathbf{Pr}[U = u] = 0 \quad \forall u.$$

This seems contradictory! How can a random variable take on each number with probability 0? If you make a draw from Unif$[0, 1]$ you'll surely get *some* number u. How could that happen if $\mathbf{Pr}[U = u] = 0$?!

Well, I'm sorry to say that's the way it will be. Strange but true. It's actually not *that* weird if you think about it. For our random variable X, if $N = 2^{64}$ then we have $\mathbf{Pr}[X = u] \leq 2^{-64}$ for all u. And this number is truly basically 0 in practice. The point is that if you *fix* a number u and *then* ask, what's the chance a draw from $\text{RandInt}(N) \cdot \frac{1}{N}$ is u, it really is essentially 0. You'd never see it happen in your lifetime, or likely the universe's lifetime. Nevertheless, every time you do a draw, you do get *some* number!

To understand this a bit better, it will help to look at expectations related to the random variable X.

2.3 Discrete approximation and expectations

Let's continue to think about

$$X \leftarrow \text{RandInt}(N) \cdot \frac{1}{N},$$

and associated expectations. To begin, it seems clear that we ought to have $\mathbf{E}[X] \approx 1/2$, and indeed

$$\begin{aligned}
\mathbf{E}[X] &= \sum_{w=1}^{N} \frac{w}{N} \cdot \mathbf{Pr}\left[X = \frac{w}{N}\right] \\
&= \sum_{w=1}^{N} \frac{w}{N} \cdot \frac{1}{N} \\
&= \frac{1}{N^2} \sum_{w=1}^{N} w^2 \\
&= \frac{N(N+1)}{2N^2} = \frac{1}{2} + \frac{1}{2N} = \frac{1}{2} \pm O(\tfrac{1}{N}).
\end{aligned}$$

So far so good, although doing that sum required a trick. What about variance? For that we'll need:

$$\begin{aligned}
\mathbf{E}[X^2] &= \sum_{w=1}^{N} \left(\frac{w}{N}\right)^2 \cdot \frac{1}{N} \\
&= \frac{1}{N^3} \sum_{w=1}^{N} w^2.
\end{aligned}$$

This is slightly tricky. If you know a lot of elementary math tricks, you might know that $\sum_{w=1}^{N} w^2 = N(N+1)(2N+1)/6$. Hence

$$\mathbf{E}[X^2] = \frac{1}{N^3} \cdot \left(\frac{N^3}{3} + \frac{N^2}{2} + \frac{N}{6} \right) = \frac{1}{3} + O(\tfrac{1}{N}),$$

and so

$$\mathbf{Var}[X] \approx \frac{1}{3} - \left(\frac{1}{2}\right)^2 = \frac{1}{12}.$$

Fine, I guess, but this process is starting to look tricky.

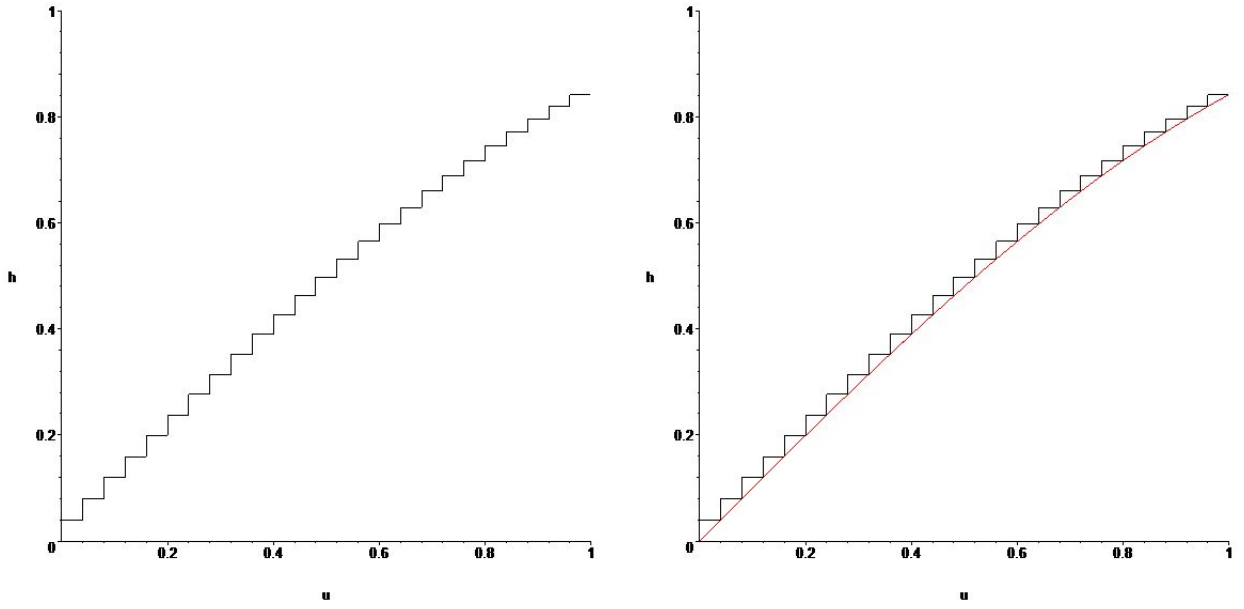
What if we wanted to know $\mathbf{E}[\sin(X)]$? Not sure why you *would* want to know this, but still: for discrete random variables we *theoretically* know how to compute this.

$$\mathbf{E}[\sin(X)] = \sum_{w=1}^{n} \sin\left(\frac{w}{N}\right) \cdot \frac{1}{N}.$$

Hmm.

2.4 Calculus to the rescue

I don't know how to compute that last sum exactly. But whenever you're adding up a huge number of tiny quantities, you should think calculus. Let's "plot it" in the case $N = 25$; in the figure on the left, for each of the 25 horizontal segments, we need to add up $\frac{1}{25}$ times its height.



I.e., we need to compute the area under the "curve" on the left. The figure on the right also includes the plot of the curve $\sin(u)$ between $u = 0$ and $u = 1$. You can see that the areas should be almost the same. Hence:

$$\mathbf{E}[\sin(X)] \approx \int_0^1 \sin(u)\, du.$$

And this is an easy thing to calculate! It's just

$$(-\cos(u))\Big|_0^1 = (-\cos(1)) - (-\cos(0)) \approx .46.$$

(In fact, we can easily bound the *error* between the integral and the true sum. From the plot, you can see that the error is equal to the sum of areas of some slightly curved right triangles. If you push these triangles all the way to the left in the plot, you see that they all fit disjointly into a single column, which has width $\frac{1}{N}$ and height $\sin 1 \approx .84$. Hence the total error is at most $\frac{.84}{N} = O(\frac{1}{N})$.)

This idea also simplifies the previous calculations we did:

$$\mathbf{E}[X] \approx \int_0^1 u \, du = \frac{1}{2}u^2\Big|_0^1 = \frac{1}{2} \cdot 1^2 - \frac{1}{2} \cdot 0^2 = \frac{1}{2};$$

$$\mathbf{E}[X^2] \approx \int_0^1 u^2 \, du = \frac{1}{3}u^3\Big|_0^1 = \frac{1}{3}.$$

Also, if we let $I(u)$ be the function which is 1 when $a \leq u \leq b$ and is 0 otherwise, then

$$\mathbf{E}[I(X)] \approx \int_0^1 I(u) \, du = \int_a^b du = b - a,$$

which agrees with the idea that this quantity should also equal $\mathbf{Pr}[a \leq X \leq b]$.

2.5 Tentative definition

As you can see, even though "in practice" our computer might do something like

$$X \leftarrow \text{RandInt}(N) \cdot \frac{1}{N},$$

it's actually *more convenient and simpler* to make and use a definition for $\text{Unif}[0,1]$. We haven't quite come to the definition yet, but it seems like we'll want that if $U \sim \text{Unif}[0,1]$, then for any function $h : [0,1] \to \mathbb{R}$,

$$\mathbf{E}[h(U)] = \int_0^1 h(u) \, du.$$

3 Exponential random variables

Our ideas for the "$\text{Unif}[0,1]$ random variable" were based on making a "continuous" version of RandInt. We will now try to come up with a continuous version of Geometric random variables. Once we do this, the general nature of a continuous random variable will become clear, and we'll be able to make formal definitions.[2]

As before, we will take a Geometric random variable with an "extreme" parameter, but then rescale it so that its mean is not extreme. Specifically, let $\lambda > 0$ be a nice medium number such as 1 or .5 or 10, and again take N to be some incredibly huge number. Let's define

$$Y \sim \text{Geometric}(\tfrac{\lambda}{N}) \cdot \tfrac{1}{N}.$$

This actually corresponds to several real-world phenomena. For example, suppose we are simulating radioactive decay. We have a single atom of Nobelium-251. Every $\frac{\lambda}{N}$ of a second, it has a $\frac{1}{N}$ chance of "decaying". Thus the random variable Y gives the number of seconds till it decays. We of course have

$$\mathbf{E}[Y] = \frac{1}{\frac{\lambda}{N}} \cdot \frac{1}{N} = \frac{1}{\lambda}.$$

[2]By the way, if you try to make a continuous version of our other favorite discrete random variables, Binomials, you'll end up with the definition of *Gaussian* random variables. We'll do exactly this in a later lecture!

3.1 Probabilities

As with our discrete approximation to uniform X, we have $\mathbf{Pr}[Y = u] \approx 0$ for every number u. E.g.,

$$\mathbf{Pr}[Y = \tfrac{w}{N}] = 0 \text{ if } w \notin \mathbb{Z}^+, \quad \mathbf{Pr}[Y = \tfrac{1}{N}] = \tfrac{\lambda}{N} \approx 0, \quad \mathbf{Pr}[Y = \tfrac{5}{N}] = \tfrac{\lambda}{N}(1 - \tfrac{\lambda}{N})^4 \leq \tfrac{\lambda}{N} \approx 0,$$

etc. On the other hand, the probability that Y is in various *intervals* is quite reasonable:

$$\begin{aligned} \mathbf{Pr}[Y \leq 1] &= \mathbf{Pr}[\text{Geometric}(\tfrac{\lambda}{N}) \leq N] \\ &= 1 - \mathbf{Pr}[\text{Geometric}(\tfrac{\lambda}{N}) > N] \\ &= 1 - (1 - \tfrac{\lambda}{N})^N \\ &\approx 1 - \exp(-\tfrac{\lambda}{N})^N \quad = \quad 1 - e^{-\lambda}. \end{aligned}$$

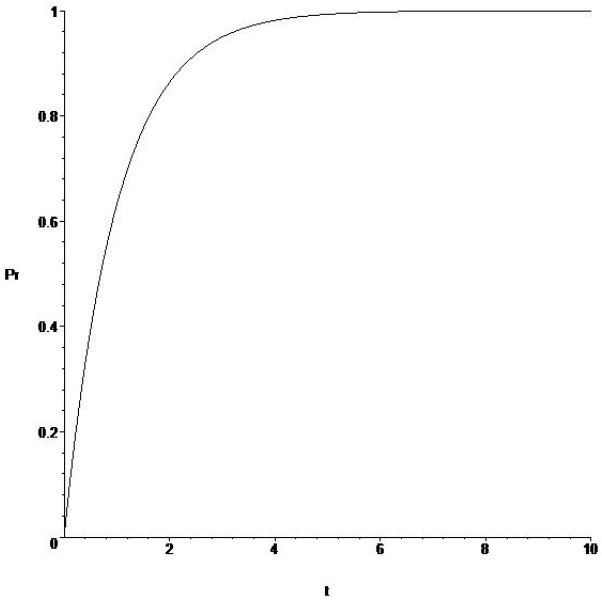
(The last step here used the Most Useful Approximation Ever. The second-to-last step was just because if you are flipping a coin until you get a heads, and the heads-probability is $\frac{\lambda}{N}$, then the probability it takes you more than N flips is the same as the probability that the first N flips are all tails.) E.g., if $\lambda = 1$, so the nobelium atom has an average lifetime of 1 second, the probability it decays in at most 1 second is (about) $1 - 1/e$.

Similarly, we could calculate:

$$\begin{aligned} \mathbf{Pr}[Y \leq 5] &= \mathbf{Pr}[\text{Geometric}(\tfrac{\lambda}{N}) \leq 5N] \\ &= 1 - \mathbf{Pr}[\text{Geometric}(\tfrac{\lambda}{N}) > 5N] \\ &= 1 - (1 - \tfrac{\lambda}{N})^{5N} \\ &\approx 1 - \exp(-\tfrac{\lambda}{N})^{5N} \quad = \quad 1 - e^{-5\lambda}. \end{aligned}$$

In general, we have

$$\mathbf{Pr}[Y \leq t] \approx 1 - e^{-\lambda t}.$$



A plot when $\lambda = 1$.

As we'll see soon, this is actually a good way to *define* a new continuous random variable. Because of the $e^{-\lambda t}$, it is called an *exponential* random variable with parameter λ. The defining property of $Y \sim \text{Exponential}(\lambda)$ is the equation

$$\mathbf{Pr}[Y \leq t] = 1 - e^{-\lambda t}.$$

3.2 Expectations

Before we get to that, let's again try to compute some expectations. We saw that $\mathbf{E}[Y] = \frac{1}{\lambda}$. Since we know the variance of Geometric, we know that our discrete random variable Y has

$$\mathbf{Var}[Y] = \frac{1 - \frac{\lambda}{N}}{\left(\frac{\lambda}{N}\right)^2} \cdot \frac{1}{N^2} = \frac{1}{\lambda^2} - \frac{1}{\lambda N} = \frac{1}{\lambda^2} \pm O(\tfrac{1}{N}).$$

(Hence we expect that an Exponential(λ) random variable, when we define it, will have variance $1/\lambda^2$.) But you can surely see that we'll have to resort to calculus to handle more complicated expectations. For example:

$$\begin{aligned}
\mathbf{E}[\sin(Y)] &= \sum_{w=1}^{\infty} \sin\left(\frac{w}{N}\right) \cdot \frac{\lambda}{N} \left(1 - \frac{\lambda}{N}\right)^{w-1} \\
&= \sum_{u=\frac{1}{N},\frac{2}{N},\frac{3}{N},\ldots} \sin(u) \cdot \lambda \left(1 - \frac{\lambda}{N}\right)^{Nu-1} \cdot \frac{1}{N} \\
&\approx \sum_{u=\frac{1}{N},\frac{2}{N},\frac{3}{N},\ldots} \sin(u) \cdot \lambda e^{-\lambda u} \cdot \frac{1}{N}
\end{aligned}$$

(Again, the approximation is the Most Useful one Ever.) And we can approximate this by an integral:

$$\mathbf{E}[\sin(Y)] = \int_0^{\infty} \sin(u) \cdot \lambda e^{-\lambda u}\, du.$$

This is a tidy formula; you can have your computer evaluate it.[3]

4 Probability density functions

4.1 Our expectation formulas

It seems that for our potential "continuous" random variables $X \sim \text{Unif}[0,1]$ and $Y \sim \text{Exponential}(\lambda)$ we had different formulas for expectations:

$$\mathbf{E}[h(X)] = \int_0^1 h(u)\, du,$$
$$\mathbf{E}[h(Y)] = \int_0^{\infty} h(u) \cdot \lambda e^{-\lambda u}\, du.$$

[3] Or, if you are a real calculus whiz, you can show by hand that it is precisely $\lambda/(1+\lambda^2)$.

What's going on? If you look at how we derived these expressions, you see that we essentially we had a tiny quantity $\frac{1}{N}$ — which we might call "du" — and we computed

$$\mathbf{Pr}[u \le X \le u + du] \approx du,$$
$$\mathbf{Pr}[u \le Y \le u + du] \approx \frac{\lambda}{N}\left(1 - \frac{\lambda}{N}\right)^{Nu-1} \approx \lambda e^{-\lambda u}\, du.$$

E.g., the probability of Y being in a tiny interval near u is about $\lambda e^{-\lambda u}$ times the width of the interval. The probability of X being in a tiny interval near u is about 1 times the width of the interval.

If we add these probabilities up over all intervals, we would expect to get 1. This is also borne out by the expectation formulas:

$$1 = \mathbf{E}[1] = \int_0^1 1\, du = 1,$$
$$1 = \mathbf{E}[1] = \int_0^\infty 1 \cdot \lambda e^{-\lambda u}\, du = (-e^{-\lambda u})\Big|_0^\infty = 0 - (-1) = 1.$$

4.2 Probability density functions

In general, we can imagine having a continuous random variable Z where the probability of Z being in the interval $[u, u + du]$ is some $f_Z(u)du$, so long as these quantities are 0 and "add up to" (i.e., integrate to) 1. This leads to:

Official definition: *A function $f : \mathbb{R} \to \mathbb{R}$ is called a* probability density function (PDF) *if:*

- $f(u) \ge 0$ *for all* $u \in \mathbb{R}$,
- $\int_{-\infty}^{\infty} f(u)\, du = 1$.

We associate to any valid PDF a continuous random variable*, Z. We sometimes write f_Z for the PDF of Z.*

This is just like the method of "introducing a discrete random variable by stating its PMF".

Unofficial interpretation: *If f_Z is the PDF of the continuous random variable Z, then* $\mathbf{Pr}[u \le Z \le u + du] \approx f_Z(u)du$ *for "tiny" $du > 0$.*

NOTE: *Please note that unlike PMFs, a PDF f does* not *in general satisfy $f(u) \le 1$.*

Definition 1. *If Z is a continuous random variable with PDF f_Z, and $h : \mathbb{R} \to \mathbb{R}$ is a function, then we define*

$$\mathbf{E}[h(Z)] = \int_{-\infty}^{\infty} h(u) f_Z(u)\, du,$$

the expectation of $h(Z)$.[4]

[4]Provided the integral $\int_{-\infty}^{\infty} |h(u)| f_Z(u)\, du$ is well-defined.

One case of this definition is when $h : \mathbb{R} \to \mathbb{R}$ is the "indicator function" of the interval $[a, b]$:

$$h(u) = \begin{cases} 1 & \text{if } a \le u \le b, \\ 0 & \text{otherwise.} \end{cases}$$

By the principle that the expectation of an indicator equals the probability of the event, we have the following:

Definition/Fact: $\mathbf{Pr}[Z \in [a, b]] = \int_a^b f_Z(u)\, du.$

Notice that it doesn't matter in the integral whether the endpoints are "included" or not. Hence

$$\mathbf{Pr}[Z \in [a, b]] = \mathbf{Pr}[Z \in (a, b)],$$

which is consistent with the notion that $\mathbf{Pr}[Z = a] = \mathbf{Pr}[Z = b] = 0$. This holds for every continuous random variable.

4.3 Examples

Definition 2. *We say X is a uniform random variable on the interval $[0, 1]$, written $X \sim \mathrm{Unif}[0, 1]$, if its PDF is*

$$f_X(u) = \begin{cases} 1 & \textit{if } 0 \le u \le 1, \\ 0 & \textit{otherwise.} \end{cases}$$

Notice that it doesn't really matter if we write $0 \le u \le 1$ or $0 < u < 1$.

More generally, suppose we want a random variable W which is "uniform on the interval $[a, b]$". If you think about it for a moment, you decide that for $a < u < b$ this should mean

$$\mathbf{Pr}[u \le W \le u + du] \approx \frac{1}{b - a} du.$$

Alternately, it makes sense that the PDF $f_Z(u)$ should be the same ("uniform") on the interval $[a, b]$, and 0 outside it. Then for the PDF property $\int f_Z(u)\, du = 1$ we need $f_Z(u) = \frac{1}{b-a}$.

Definition 3. *We say W is a uniform random variable on the interval $[a, b]$, written $X \sim \mathrm{Unif}[a, b]$, if its PDF is*

$$f_W(u) = \begin{cases} \frac{1}{b-a} & \textit{if } 0 \le u \le 1, \\ 0 & \textit{otherwise.} \end{cases}$$

Let's do a basic calculation with this W. Using the function $h(u) = u$,

$$\mathbf{E}[W] = \mathbf{E}[h(W)] = \int_{-\infty}^{\infty} u f_W(u)\, du = \int_a^b u \frac{1}{b-a}\, du = \frac{1}{b-a} \cdot \tfrac{1}{2} u^2 \Big|_a^b = \frac{b^2 - a^2}{2(b-a)} = \frac{b+a}{2},$$

just as we would expect.

Let's now do Exponentials:

Definition 4. *We say Y is an exponential random variable with parameter $\lambda > 0$, written $X \sim$ Exponential(δ), if its PDF is*

$$f_Y(u) = \begin{cases} \lambda e^{-\lambda u} & \text{if } u \geq 0, \\ 0 & \text{if } u < 0. \end{cases}$$

Let's calculate the mean and variance to make sure they are what we expect:

Proposition 5. *If $Y \sim$ Exponential(λ) then $\mathbf{E}[Y] = \frac{1}{\lambda}$ and $\mathbf{Var}[Y] = \frac{1}{\lambda^2}$ (hence $\mathbf{stddev}[Y] = \frac{1}{\lambda}$).*

Please remember these formulas!

Proof.

$$\mathbf{E}[Y] = \int_0^\infty u \cdot \lambda e^{-\lambda u}\, du.$$

You need to use integration by parts to do this integral:

$$\int_0^\infty u \cdot \lambda e^{-\lambda u}\, du = -\int_0^\infty u\, d(e^{-\lambda u}) = -ue^{-\lambda u}\Big|_0^\infty + \int_0^\infty e^{-\lambda u}\, du = 0 - \frac{1}{\lambda} e^{-\lambda u}\Big|_0^\infty = \frac{1}{\lambda}.$$

As for the variance: on the homework! □

5 Cumulative density functions

5.1 Current shortcomings

We now have an acceptable theory of "continuous random variables". There are a few ways in which it could be better, though. One issue so far is that it is "unrigorous" and somewhat difficult to reason about PDFs by reasoning about "infinitesimal" quantities du. Here is an example of this:

Question: Suppose $X \sim \text{Unif}[0,1]$, and let $Z = X^2$. What is the PDF of Z?

Non-rigorous answer: We believe that for $0 < u < 1$,

$$\mathbf{Pr}[u \leq Z \leq u + du] = \mathbf{Pr}[u \leq X^2 \leq u + du] = \mathbf{Pr}[\sqrt{u} \leq X \leq \sqrt{u + du}].$$

But what is $\sqrt{u + du}$ when u is a normal number and du is "tiny"? You can reason as follows:

$$\sqrt{u + du} = \sqrt{u}(1 + \tfrac{du}{u})^{1/2} = \sqrt{u}(1 + (1/2)\tfrac{du}{u} - \tfrac{(1/2)^2}{2}\tfrac{du^2}{u^2} + \cdots) \approx \sqrt{u}(1 + \tfrac{du}{2u}) = \sqrt{u} + \tfrac{1}{2\sqrt{u}} du,$$

where in the middle we used the Generalized Binomial Theorem, or Taylor's Theorem. Thus

$$\mathbf{Pr}[u \leq Z \leq u + du] \approx \mathbf{Pr}[\sqrt{u} \leq X \leq \sqrt{u} + \tfrac{1}{2\sqrt{u}} du] \approx \tfrac{1}{2\sqrt{u}}.$$

So we expect

$$f_Z(u) = \frac{1}{2\sqrt{u}}$$

for $u \in [0,1]$. But this is not really rigorous math.

Another disappointing aspect of the picture so far is that we seem to need to treat continuous and discrete random variables totally separately: PMFs and PDFs are totally different beasts.

5.2 Cumulative distribution functions

Both of these shortcomings can be overcome by introducing a simple new notion:

Definition 6. *The* cumulative distribution function (CDF) *of a random variable X is the function $F_X : \mathbb{R} \to \mathbb{R}$ defined by*

$$F_X(t) = \mathbf{Pr}[X \leq t].$$

Alternatively, a CDF is a function F with the following properties:

- *$F_X(t)$ is a monotonically nondecreasing function of t.*
- *$F_X(t) \to 1$ as $t \to \infty$ and $F_X(t) \to 0$ as $t \to -\infty$.*

Hence the plot of a CDF will look something like the plot at the bottom of page 6 (except that you should also think of this plot extending leftward to $-\infty$, having value 0 for $t \leq 0$).

Remark 7. *If X is a discrete random variable then F_X is a "step function" (piecewise constant). On the other hand, if F_X is a continuous function, we say that X is a continuous random variable. (It is possible that neither is true.)*

For a continuous random variable X, we have the following simple relationship between the CDF and PDF:

Proposition 8. *If X is a continuous random variable*[5] *with CDF F_X, then its PDF f_X is the derivative: $F'_X(t) = f_X(t)$.*

Proof. This is the Fundamental Theorem of Calculus:

$$F_X(t) = \mathbf{Pr}[X \leq t] = \int_{-\infty}^{t} f_X(u)\, du,$$

so $\frac{d}{dt} F_X(t) = f_X(t)$. □

5.3 Examples

If $X \sim \text{Unif}[0,1]$ then we know $\mathbf{Pr}[X \leq t] = t$ when $t \in [0,1]$. (Check: $\frac{d}{dt} t = 1$, the PDF of X on $[0,1]$.) More precisely:

Proposition 9. *The CDF of $X \sim \text{Unif}[0,1]$ is*

$$F_X(t) = \begin{cases} 0 & \text{if } t \leq 0, \\ t & \text{if } 0 \leq t \leq 1, \\ 1 & \text{if } t \geq 1. \end{cases}$$

As for exponential random variables, we know to expect the following from our original analysis of the discrete analogue:

Proposition 10. *The CDF of $Y \sim \text{Exponential}(\lambda)$ is*

$$F_Y(t) = \begin{cases} 0 & \text{if } t \leq 0, \\ 1 - e^{-\lambda t} & \text{if } t \geq 0. \end{cases}$$

[5] *Math nerd technicality:* let's say that X should have a piecewise continuously-differentiable CDF.

Proof. For $t \geq 0$,

$$F_Y(t) = \mathbf{Pr}[Y \leq t] = \int_{-\infty}^{t} f_Y(u)\,du = \int_0^t \lambda e^{-\lambda u}\,du = -e^{-\lambda u}\Big|_0^t = -e^{\lambda t} - (-1) = 1 - e^{-\lambda t}.$$

□

Check: $F_Y'(t) = \lambda e^{-\lambda t} = f_Y(t)$ for $t \geq 0$.

5.4 Derived distributions

You can use CDFs to (properly, rigorously) determine the PDFs of "derived distributions". Let's do the previous example properly.

Question: Suppose $X \sim \text{Unif}[0,1]$, and let $Z = X^2$. What is the PDF of Z?

Answer: We compute the CDF of Z. Clearly $F_Z(t) = 0$ for $t < 0$ and $F_Z(t) = 1$ for $t > 1$. For $0 \leq t \leq 1$,

$$F_Z(t) = \mathbf{Pr}[Z \leq t] = \mathbf{Pr}[X^2 \leq t] = \mathbf{Pr}[X \leq \sqrt{t}] = F_X(\sqrt{t}) = \sqrt{t}.$$

The derivative of the constant 0 is 0, as is the derivative of the constant 1. For $0 \leq t \leq 1$ we have

$$f_Z(t) = \frac{d}{dt}\sqrt{t} = \frac{1}{2\sqrt{t}}.$$

Hence the PDF of Z is given by

$$f_Z(t) = \begin{cases} 0 & \text{if } t \leq 0, \\ \frac{1}{2\sqrt{t}} & \text{if } 0 \leq t \leq 1, \\ 0 & \text{if } t \geq 1. \end{cases}$$

15-359: Probability and Computing
Fall 2009

Lecture 18: Joint continuous random variables

1 One continuous random variable: a review

Let's review the basic definitions for continuous random variables we saw last time. Generally, continuous random variables are introduced by their PDF (probability density function):

Definition 1. *A continuous random variable X is defined by its* probability density function *(PDF) $f_X : \mathbb{R} \to \mathbb{R}$, which has the properties:*

- $f_X(u) \geq 0$ *for all* $u \in \mathbb{R}$,
- $\int_{-\infty}^{\infty} f_X(u)\,du = 1$.

The unofficial interpretation is

$$\text{“}\mathbf{Pr}[u \leq X \leq u + du] \approx f_X(u)\,du\text{”}.$$

Our basic examples are the Uniform distribution on the interval $[a, b]$ and the Exponential distribution with parameter $\lambda > 0$:

$$X \sim \text{Unif}[a,b], \qquad f_X(u) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq u \leq b, \\ 0 & \text{else.} \end{cases}$$

$$Y \sim \text{Exponential}(\lambda), \qquad f_Y(u) = \begin{cases} \lambda \exp(-\lambda u) & \text{if } u \geq 0, \\ 0 & \text{else.} \end{cases}$$

We define expectations as usual:

Definition 2. *Given a continuous random variable X and a function $h : \mathbb{R} \to \mathbb{R}$,*

$$\mathbf{E}[h(X)] = \int_{-\infty}^{\infty} h(u) f_X(u)\,du.$$

We can also define events straightforwardly:

Definition 3. *In an experiment with one continuous random variable X, an* event *is a subset $A \subseteq \mathbb{R}$, often an interval. We have*

$$\mathbf{Pr}[A] = \mathbf{Pr}[X \in A] = \int_A f_X(u)\,du.$$

For example, if $Y \sim \text{Exponential}(1)$, then

$$\mathbf{Pr}[1 \le Y \le 2] = \int_1^2 \exp(-u)\,du = -\exp(-u)\Big|_1^2 = \frac{1}{e} - \frac{1}{e^2}.$$

Definition 4. *The* cumulative density function *(CDF) of X is*

$$F_X(t) = \mathbf{Pr}[X \le t].$$

Its derivative is the PDF:

$$\frac{d}{dt}F_X(t) = f_X(t).$$

It has the property that it increases from $F_X(t) = 0$ to $F_X(t) = 1$ as t goes from $-\infty$ to ∞.

For our canonical examples,

$$X \sim \text{Unif}[a,b], \qquad F_X(t) = \begin{cases} 0 & \text{if } t < a, \\ \frac{t-a}{b-a} & \text{if } a \le t \le b, \\ 1 & \text{if } t > b. \end{cases}$$

$$Y \sim \text{Exponential}(\lambda), \qquad F_Y(t) = \begin{cases} 0 & \text{if } t < 0, \\ 1 - \exp(-\lambda t) & \text{if } t \ge 0. \end{cases}$$

Finally, it's easy to describe conditioning a random variable on an event:

Definition 5. *Given a continuous random variable X and an event $A \subseteq \mathbb{R}$ with $\mathbf{Pr}[A] \neq 0$, the random variable $X \mid A$, "X conditioned on A", has PDF*

$$f_{X|A}(u) = \begin{cases} \frac{f_X(u)}{\mathbf{Pr}[A]} & \text{if } u \in A, \\ 0 & \text{else.} \end{cases}$$

We define conditional expectations,

$$\mathbf{E}[h(X) \mid A] = \int_{-\infty}^{\infty} h(u) f_{X|A}(u)\,du$$

as usual.

2 Two continuous random variables

Let's now discuss experiments where there is more than one continuous random variable. We'll mainly discuss the case of two continuous random variables, the generalizations to three or more being the obvious ones.

2.1 Joint PDFs

The simplest way to define an experiment with two continuous random variables is to simply *state their joint PDF*:

Definition 6. *A pair of continuous random variables X and Y is defined by its* joint ("bivariate") probability density function $f_{XY} : \mathbb{R}^2 \to \mathbb{R}$*, which has the properties:*

- $f_{XY}(u, v) \geq 0$ *for all* $u, v \in \mathbb{R}$*,*
- $\iint f_{XY}(u, v)\, du\, dv = 1$*, where the double integral is over the whole plane* $\mathbb{R}^2$*.*

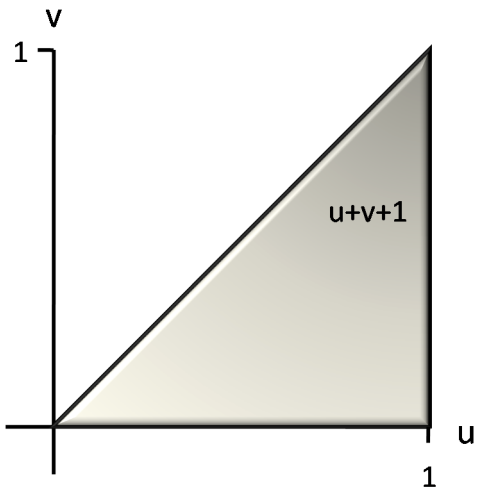
As you might guess, the unofficial interpretation is

$$\text{"}\mathbf{Pr}[u \leq X \leq u + du \;\&\; v \leq Y \leq v + dv] \approx f_{XY}(u, v)\, du\, dv\text{"}.$$

It is **very** helpful to draw pictures when dealing with pairs of random variables. Let's have a nontrivial running example:

Running Example: Let $f_{XY}(u, v) = \begin{cases} u + v + 1 & \text{if } 0 \leq u \leq 1 \text{ and } 0 \leq v \leq u, \\ 0 & \text{else.} \end{cases}$

Here the joint PDF is nonzero only on the triangle with corners $(0, 0)$, $(1, 0)$, and $(1, 1)$, and its value slopes upwards in the direction of $(1, 1)$:



We should check that this is a valid PDF. It's definitely nonnegative everywhere, so that condition is fine. To check that it integrates to 1, you need your basic calculus skills:

$$\iint f_{XY}(u, v)\, du\, dv = \int_0^1 \int_0^u (u + v + 1)\, dv\, du.$$

The inner integral is

$$\int_0^u (u + v + 1)\, dv = \frac{1}{2}v^2 + (u + 1)v \Big|_0^u = \frac{1}{2}u^2 + (u + 1)u = \frac{3}{2}u^2 + u.$$

Thus the whole integral is

$$\int_0^1 \left(\frac{3}{2}u^2 + u\right) du = \frac{1}{2}u^3 + \frac{1}{2}u^2 \Big|_0^1 = \frac{1}{2} + \frac{1}{2} = 1,$$

as needed.

2.2 Expectations, events, and CDFs

Expectations, events, and CDFs for joint random variables are defined in a straightforward way.

Definition 7. *Given joint continuous random variables X and Y, and a function $h : \mathbb{R}^2 \to \mathbb{R}$,*

$$\mathbf{E}[h(X,Y)] = \iint_{\mathbb{R}^2} h(u,v) f_{XY}(u,v)\, du\, dv.$$

In our running example, if we wanted to compute the expected value of, say, Y/X, we would need to evaluate the integral

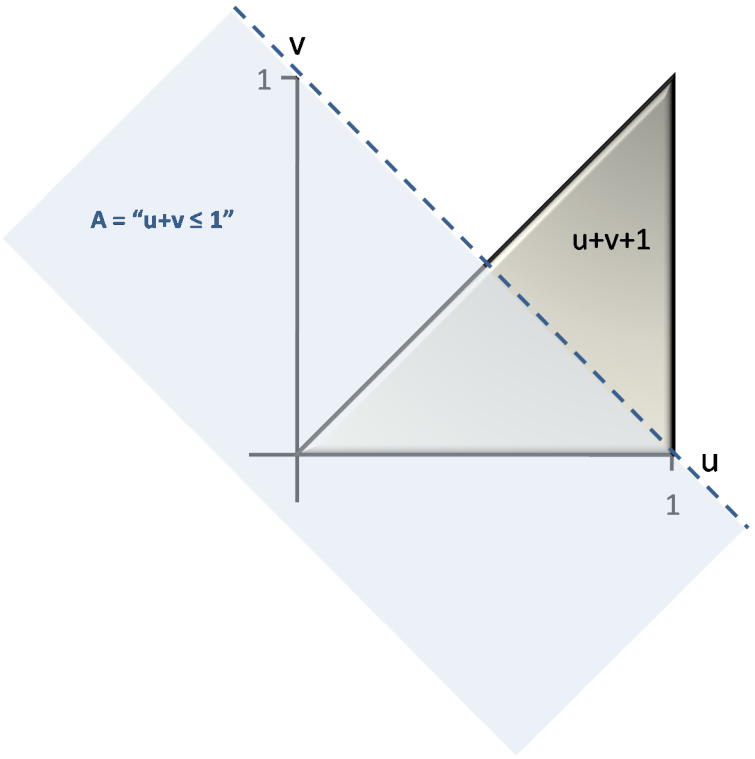
$$\int_0^1 \int_0^u \frac{v}{u} \cdot (u+v+1)\, dv\, du.$$

Exercise: Show that this integral equals $\frac{19}{36}$.

Definition 8. *In an experiment with two continuous random variables X and Y, an event A is a subset of the plane, $\mathbb{R}^2$. Its probability is*

$$\mathbf{Pr}[A] = \mathbf{Pr}[(X,Y) \in A] = \iint_A f_{XY}(u,v)\, du\, dv.$$

In our running example, let's consider the event $A = \{(u,v) : u+v \leq 1\}$, which we would more casually call "$X+Y \leq 1$". To find its probability, we would compute — with the help of the following diagram



— the integral

$$\begin{aligned}
\iint_{u+v\le 1} f_{XY}(u,v)\,du\,dv &= \int_{v=0}^{1/2}\int_{u=v}^{1-v}(u+v+1)\,du\,dv \\
&= \int_0^{1/2}\left(\frac{1}{2}u^2+(v+1)u\Big|_v^{1-v}\right)dv \\
&= \int_0^{1/2}\left(\frac{1}{2}(1-v)^2-\frac{1}{2}v^2+(v+1)(1-v)-(v+1)v\right)dv \\
&= \int_0^{1/2}\left(\frac{3}{2}-2v-2v^2\right)dv \\
&= \frac{3}{2}v-v^2-\frac{2}{3}v^2\Big|_0^{1/2} \quad = \quad \frac{5}{12}.
\end{aligned}$$

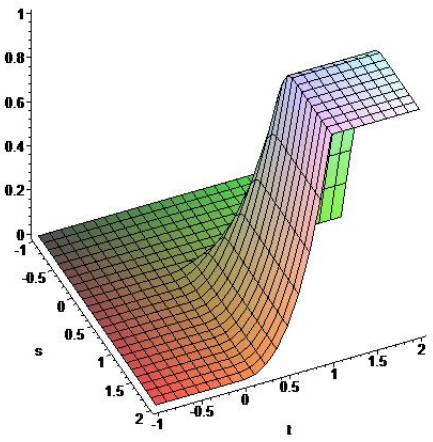
Definition 9. *The joint CDF of continuous random variables X and Y is $F_{XY}:\mathbb{R}^2\to\mathbb{R}$ defined by*

$$F_{XY}(s,t)=\mathbf{Pr}[X\le s\ \&\ Y\le t].$$

It satisfies the following relation with the joint PDF:

$$\frac{\partial^2}{\partial s\partial t}F_{XY}(s,t)=f_{XY}(s,t).$$

We could calculate this for our running example, although it is a bit of a chore. Having done so, the plot of $F_{XY}(s,t)$ would look like this:



3 Marginals and independence

Remember that if you have two discrete random variables X and Y, with joint PMF $p_{XY}(u,v)$, you can get determine the PMF $p_X(u)$ for X by summing over ("columns in the table") v, and similarly you can get the PMF $p_Y(v)$ for Y by summing over ("rows in the table") u? We can do a very similar thing for continuous random variables; these are called the *marginals*.

Definition 10. *Let X and Y be continuous random variables with joint PDF $f_{XY}(u,v)$. The PDF for X alone, called the* marginal density *for X, is*

$$f_X(u) = \int_{-\infty}^{\infty} f_{XY}(u,v)\,dv.$$

Similarly, the marginal density for Y is

$$f_Y(v) = \int_{-\infty}^{\infty} f_{XY}(u,v)\,du.$$

Remark 11. *It's easy to check that this really is a proper PDF. For example, $f_X(u)$ is clearly nonnegative for all u, since it is the integral of a nonnegative function. And $\int f_X(u)\,du = 1$, because this is just the integral of the joint PDF f_{XY} over the whole plane $\mathbb{R}^2$.*

In our running example, the marginal density for X is

$$f_X(u) = \int_{-\infty}^{\infty} f_{XY}(u,v)\,dv = \int_0^u (u+v+1)\,dv = \frac{3}{2}u^2 + u,$$

for $u \in [0,1]$ (and 0 outside this interval). You can check that this is a valid single PDF.

This definition extends to multiple PDFs in the obvious way:

Definition 12. *If X, Y, and Z are continuous random variables with joint PDF $f_{XYZ}(u,v,w)$, then the joint (marginal) PDF of X and Y is*

$$f_{XY}(u,v) = \int_{-\infty}^{\infty} f_{XYZ}(u,v,w)\,dw,$$

the marginal PDF of Y is

$$f_Y(v) = \iint f_{XYZ}(u,v,w)\,du\,dw,$$

etc.

We define independence (e.g., for 3 random variables) as follows:

Definition 13. *Continuous random variables X, Y, and Z are* independent *if and only if*

$$f_{XYZ}(u,v,w) = f_X(u) f_Y(v) f_Z(w).$$

3.1 Familiar facts regarding expectations

Linearity of expectation holds as usual for continuous random variables. E.g., we can check the simple case of the sum of two random variables:

$$\begin{aligned}
\mathbf{E}[X+Y] &= \iint (u+v) f_{XY}(u,v)\,du\,dv \\
&= \iint u f_{XY}(u,v)\,dv\,du + \iint v f_{XY}(u,v)\,du\,dv \\
&= \int u \left(\int f_{XY}(u,v)\,dv \right) du + \int v \left(\int f_{XY}(u,v)\,du \right) dv \\
&= \int u f_X(u)\,du + \int v f_Y(v)\,dv \quad = \quad \mathbf{E}[X] + \mathbf{E}[Y].
\end{aligned}$$

Expectation of a product equals product of expectations for *independent* random variables continues to hold for continuous random variables. We leave it as an exercise for you to show that

$$\mathbf{E}[XY] = \mathbf{E}[X]\mathbf{E}[Y] \qquad \text{when } X \text{ and } Y \text{ are independent.}$$

4 Conditional distributions

Given two random variables X and Y, we can define the conditional distribution given an event in the natural way:

Definition 14. *Given continuous random variables X and Y and an event $A \subseteq \mathbb{R}^2$ with $\mathbf{Pr}[A] \neq 0$, the conditional PDF of X and Y given A is*

$$f_{XY|A}(u) = \begin{cases} \frac{f_{XY}(u,v)}{\mathbf{Pr}[A]} & \text{if } (u,v) \in A, \\ 0 & \text{else.} \end{cases}$$

It's also natural to want to condition on an event like "$Y = v$". Using the above definition is problematic, because of course $\mathbf{Pr}[Y = v]$ is always 0 if Y is a continuous random variable. Nevertheless, we can still make an appropriate definition:[1]

Definition 15. *Given continuous random variables X and Y, the* conditional PDF of X given $Y = v$ *is*

$$f_{X|Y=v}(u) = \frac{f_{XY}(u,v)}{f_Y(v)},$$

provided that the marginal density $f_Y(v) \neq 0$.

Again, it's easy to check this is a valid PDF; if you integrate it over u, you'll get $\frac{f_Y(v)}{f_Y(v)} = 1$. The following fact is also immediate from the definitions:

Fact 16. $f_{XY}(u,v) = f_Y(v) f_{X|Y=v} = f_X(u) f_{Y|X=u}$.

4.1 An example

It is only now with all these definitions in place that we can properly analyze what looks like a basic example. Suppose:

$X \leftarrow \text{Unif}[0, 10]$
$Y \leftarrow \text{Unif}[0, X]$.

Question: What is the joint PDF of X and Y? What is the marginal density of Y?

Answer: We have

$$f_X(u) = \begin{cases} .1 & \text{if } 0 \leq u \leq 10, \\ 0 & \text{else.} \end{cases}$$

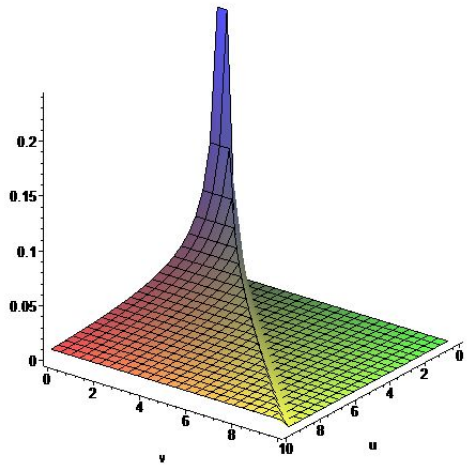
We then have (for $u \neq 0$)

$$f_{Y|X=u} = \begin{cases} \frac{1}{u} & \text{if } 0 \leq v \leq u, \\ 0 & \text{else.} \end{cases}$$

[1] Things are getting mathematically a little shady at this point; please believe me that it's possible to make all of our definitions 100% mathematically rigorous.

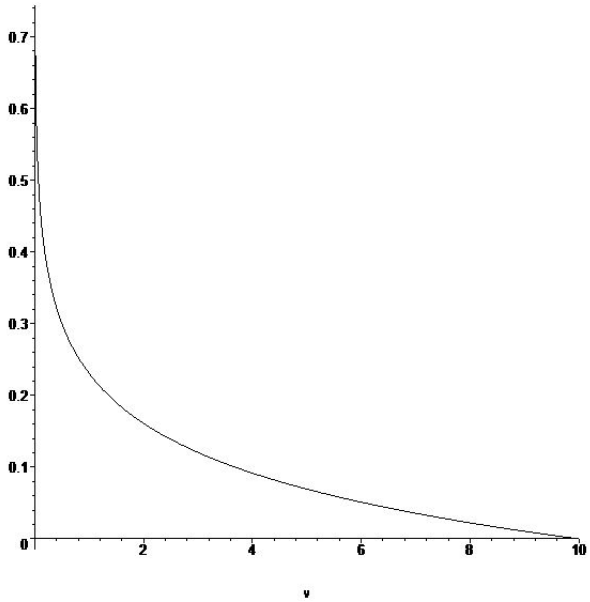
Thus we get

$$f_{XY}(u,v) = f_X(u) f_{Y|X=u}(v) = \begin{cases} \frac{.1}{u} & \text{if } 0 \le v \le u \le 10, \\ 0 & \text{else.} \end{cases}$$



Hence the marginal density on Y is, for $0 \le v \le 10$,

$$f_Y(v) = \int_v^1 \frac{.1}{u}\, du = .1 \ln(u)\Big|_v^1 = .1 \ln(10) - .1 \ln v = .1 \ln(10/v).$$



$f_Y(v) = .1 \ln(10/v)$

You can check that indeed $\int_0^1 0.1 \ln(10/v)\, dv = 1$. It's both logical, and evident from the picture, that Y tends to be on the smaller side of the range $[0, 10]$.

4.2 Law of Conditional Expectation

We can use conditional densities in the Law of Conditional Expectation (and also the Law of Total Probability) in the natural way:

Proposition 17. *If X and Y are continuous random variables, and $h : \mathbb{R} \to \mathbb{R}$, then*

$$\mathbf{E}[Y] = \int \mathbf{E}[Y \mid X = u] f_X(u)\, du.$$

Let's try this out for the example of

$X \leftarrow \text{Unif}[0, 10]$
$Y \leftarrow \text{Unif}[0, X]$.

It is not hard to check the intuitively obvious statement $\mathbf{E}[Y \mid X = u] = u/2$. Hence

$$\mathbf{E}[Y] = \int \mathbf{E}[Y \mid X = u] f_X(u)\, du = \int_0^{10} (u/2) \cdot .1\, du = .1 \frac{1}{4} u^2 \Big|_0^{10} = 2.5.$$

We could alternatively have calculated

$$\int_0^{10} (u/2) \cdot .1\, du = \frac{1}{2} \int_0^{10} u \cdot .1\, du = \frac{1}{2} \mathbf{E}[X] = 5/2 = 2.5.$$

If your calculus skills are good, you can also check that this jives with our calculation $f_Y(v) = .1 \ln(10/v)$:

$$\mathbf{E}[Y] = \int_0^{10} v \cdot .1 \ln(10/v)\, dv = \cdots = 2.5.$$

15-359: Probability and Computing
Fall 2009

Lecture 19: The Poisson Process

In this lecture we'll see the very neat "Poisson Process", which is the basis of continuous-time Markov Chains and queueing theory.

1 Wonderful properties of the exponential distribution

Understanding the Poisson Process requires learning some wonderful properties of the *exponential* distribution. Let's begin with the basic facts about this distribution. Let $X \sim \text{Exponential}(\lambda)$, where $\lambda > 0$.

Fact 1.

- *The PDF is* $f_X(u) = \lambda \exp(-\lambda u)$.
- *The CDF is* $F_X(u) = 1 - \exp(-\lambda u)$.
- *Intuitively,* X *is like the time till the first 'heads' when you flip a coin with heads-probability* $\lambda\delta$ *every* δ*-length time tick... in the limit as* $\delta \to 0$.
- $\mathbf{E}[X] = 1/\lambda$.
- $\mathbf{Var}[X] = 1/\lambda^2$.

On Homework 9, Problem 2b, you also proved the *memoryless* property of exponential random variables:

Theorem 2. *For all* $s, t \geq 0$, $\mathbf{Pr}[X > s + t \mid X > s] = \mathbf{Pr}[X > t]$.

(By the way, it doesn't matter if we write $>$ or $\geq$ here, because for continuous random variables, $\mathbf{Pr}[X = u] = 0$.) Since knowing $\mathbf{Pr}[X > t]$ for each t means you know the CDF, which uniquely defines the distribution, this is precisely saying that if you condition on an Exponential(λ) random variable being greater than s, its conditional distribution is just $s + \text{Exponential}(\lambda)$.

Puzzle: Customer A and Customer B are being served at a bank with two tellers. Then Customer C comes in and waits for the first available teller. Assume all service times for the customers are independent Exponential(λ) random variables. What is the probability Customer C is the last to leave?

Answer: This is a simple version of the "batteries" problem on the homework. At the time Customer C comes in, the conditional distribution of A's and B's remaining service times is still Exponential(λ) (independently), by memorylessness. Now whichever of A or B is served first, and how long that takes, doesn't matter: when the first of them leaves, the other has Exponential(λ) remaining service time, as does Customer C. Hence by symmetry, the probability C leaves last is 1/2.

Another wonderful property you proved in Homework 9, Problem 4a is that the minimum of independent exponentials is exponential:

Theorem 3. *Let $X_i \sim$ Exponential(λ_i) for $i = 1 \dots n$, and assume these random variables are independent. Let $Y = \min(X_1, \dots, X_n)$. Then $Y \sim$ Exponential($\lambda_1 + \cdots + \lambda_n$).*

It is pretty easy to prove this using the CDFs. Here is another 'pseudo-proof', using the fact that exponentials are the continuous analogues of geometrics:

Proof. (Pseudo.) It's enough to prove it for $n = 2$; you can then use induction. For X_1 we imagine flipping a coin with heads-probability $\lambda_1\delta$, every δ-length time tick, until we get a heads. For X_2 it is the same, except the heads-probability is $\lambda_2\delta$. If we imagine flipping the coins *together* each tick, then Y represents the time till we get a heads on *either* coin. The probability of getting heads on either when we flip *both* coins is

$$1 - \mathbf{Pr}[TT] = 1 - (1 - \lambda_1\delta)(1 - \lambda_2\delta) = \lambda_1\delta + \lambda_2\delta - \lambda_1\lambda_2\delta^2.$$

Since we think of δ as tiny, δ^2 is super-tiny and thus negligible. I.e.,

$$\mathbf{Pr}[\text{either is heads}] \approx (\lambda_1 + \lambda_2)\delta.$$

Thus Y is like an exponential random variable with parameter $\lambda_1 + \lambda_2$. □

Let's extend this fact to answer the following question: Given $X_1, \dots, X_n$, what is the probability that the minimum is X_i?

Theorem 4. *In the setting of Theorem 3,*

$$\mathbf{Pr}[\min(X_1, \dots, X_n) = X_i] = \frac{\lambda_i}{\lambda_1 + \cdots + \lambda_n}.$$

Proof. We'll just do the case $n = 2$ again, and leave to you the extension to general n via induction. It also suffices to do the case $i = 1$ (because then the $i = 2$ case follows by symmetry, or by subtracting from 1). Hence we need to prove

$$\mathbf{Pr}[\min(X_1, X_2) = X_1] = \mathbf{Pr}[X_1 \le X_2] = \frac{\lambda_1}{\lambda_1 + \lambda_2}.$$

We can compute this probability in a couple of different ways. One is to simply note that it is the probability of the event $(X_1, X_2) \in A = \{(u, v) : 0 \le u \le v\}$, and compute it by integrating the

joint PDF. Another way is to use the Law of Total Probability ("conditioning on the value of X_1"):

$$\begin{aligned}
\mathbf{Pr}[X_1 \le X_2] &= \int_{u=0}^{\infty} \mathbf{Pr}[X_1 \le X_2 \mid X_1 = u] f_{X_1}(u)\, du \\
&= \int_{0}^{\infty} \mathbf{Pr}[X_2 \ge u] f_{X_1}(u)\, du \qquad \text{(since } X_1 \text{ and } X_2 \text{ are independent)} \\
&= \int_{0}^{\infty} \exp(-\lambda_2 u)\lambda_1 \exp(-\lambda_1 u)\, du \quad \text{(using the CDF and PDF of exponentials)} \\
&= \lambda_1 \int_{0}^{\infty} \exp(-(\lambda_1+\lambda_2)u)\, du \\
&= \frac{\lambda_1}{\lambda_1+\lambda_2} \int_{0}^{\infty} (\lambda_1+\lambda_2)\exp(-(\lambda_1+\lambda_2)u)\, du \\
&= \frac{\lambda_1}{\lambda_1+\lambda_2} \cdot 1,
\end{aligned}$$

as needed. In the last step, we used the fact that $(\lambda_1+\lambda_2)\exp(-(\lambda_1+\lambda_2)u)$ is the PDF of a random variable (namely, an Exponential$(\lambda_1+\lambda_2)$) and hence integrates to 1. (This trick is a useful shortcut in probability calculations — try to be on the lookout for it!) □

2 The Poisson Process

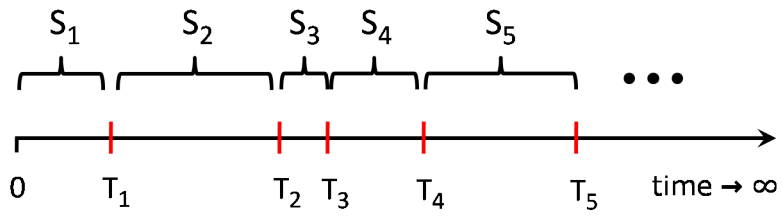
The *Poisson Process* is a very widely used model for the arrival times of jobs at a processor, or packets at a router, or threads at a critical section, or requests at a web server, or phone calls at a cell tower, or beta particles at a geiger counter, or customers at a cashier, or raindrops at a patch of ground, or...

2.1 Definition based on interarrivals

There are several equivalent ways to define the Poisson Process. Most books choose the hardest-to-understand to give first. I will give the easiest to understand first:

Definition 5. *A Poisson Process with* rate/intensity/parameter $\lambda > 0$ *consists of a sequence of* interarrival time *random variables,* $S_1, S_2, S_3, \ldots$. *Each* $S_i \sim$ Exponential(λ), *and these random variables are independent. This gives us a sequence of* arrival times *(AKA* waiting times*)* $T_0, T_1, T_2, T_3, \ldots$, *where* T_0 *is always* 0, *and* $T_\ell = \sum_{i=1}^{\ell} S_i$.

We always like to think of the following picture for a Poisson Process: we have a timeline going from 0 to ∞, and the points represent "arrivals". Remember, the fundamental definition is that the *interarrival* times are independent Exponential(λ)'s. (You might be wondering, therefore, why it's called the *Poisson* Process! We'll see shortly.)



Intuitive interpretation: Given the Exponential(λ) interarrival times, it's as though you are dividing time up into tiny segments of length δ, and flipping a coin with head-probability $\lambda\delta$ for each segment. Every time you get a 'heads', you treat it as having an arrival at that tiny segment.

Observation: Given this interpretation, you can see that in some sense you do not have to flip the coins "in order". Rather, you can imagine each tiny segment of width δ has its own coin, and decides to be an arrival time with probability $\lambda\delta$.

Discussion: Is this a *reasonable* model of things like hit times for a web server, or cell phone calls going into a tower? *Well*, perhaps, perhaps not. If you think of the definition in terms of the observation we just made, it's reasonable if you imagine there are a large number of agents, each of which has a small chance of "arriving"; and, in each tiny segment of time, there is an equal (tiny) chance that an agent will arrive. This is somewhat reasonable for things like arrivals at a web server in a modest stretch of time. (For longer stretches, probability the arrival rate will vary with time of day; there are extensions to the Poisson Process that allow for this.)

2.2 Arrival count $N(t)$

An obvious first question to ask is:

Question: Given a fixed time $t \geq 0$, let $N(t)$ be the random variable counting the number of arrivals up to time t. What is the distribution of this random variable?

Hint: Notice that $N(t)$ is a discrete random variable which has possible values 0, 1, 2, ... Also, notice that it is called the *Poisson* Process.

Intuitive answer: We observed before that we can think of the time segment $[0, t]$ as being divided into tiny segments of length δ, and each segment as flipping a $(\lambda\delta)$-biased coin to determine if there is an arrival in that segment. Thus we would think that the number of arrivals in time $[0, t]$ is Binomial$(t/\delta, \lambda\delta)$. Since t/δ is huge and $\lambda\delta$ is tiny, this is the perfect time for the Poisson Approximation, which suggests that

$$\text{Binomial}(t/\delta, \lambda\delta) \approx \text{Poisson}((t/\delta)(\lambda\delta)) \sim \text{Poisson}(\lambda t).$$

Rigorous answer: The intuitive answer is correct:

$$N(t) \sim \text{Poisson}(\lambda t).$$

The rigorous proof is straightforward, but requires some fiddly integration. See the homework!

Corollary: $\mathbf{E}[N(t)] = \lambda t$. I.e., the expected number of arrivals in time interval $[0, t]$ is λt. This is why the parameter λ is often called the arrival *rate*.

Example problem: Hits come into a web server according to a Poisson Process with rate $\lambda = \frac{1}{10 \text{ msec}}$. What is the expected number of hits in the first second, and what is the variance?

Solution: Letting time be measured in milliseconds, we are interested in $\mathbf{E}[N(1000)]$ and $\mathbf{Var}[N(1000)]$. We know that $N(1000) \sim \text{Poisson}(\lambda 1000) \sim \text{Poisson}(100)$. Hence the expected number of hits is 100 and the variance is also 100.

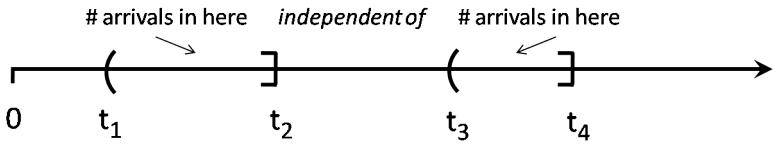
2.3 Definition based on Poisson arrivals

Let's make another observation. Suppose we fix some point in time $s \geq 0$, and consider only the arrivals after this point. You can see that this is *also* a Poisson Process with rate λ; this is because the memorylessness property of exponentials ensures that the interarrival times starting from s are independent Exponential(λ)'s. We can draw the following conclusion:

Fact 6. *Let $s, t \geq 0$. The number of arrivals in the interval $(s, s+t]$, which is equal to $N(s+t) - N(s)$, has distribution Poisson(λt).*

And memorylessness also implies:

Fact 7. (and **Definition.**) *A Poisson Process has* independent increments*, meaning that if $(t_1, t_2]$ and $(t_3, t_4]$ are disjoint intervals, the number of arrivals in the first interval $N(t_2) - N(t_1)$ is independent of the number of arrivals in the second interval $N(t_4) - N(t_3)$.*



Independent Increments

These two properties can actually be taken as an alternate definition of the Poisson Process. First, let's define:

Definition 8. *A* continuous-time stochastic process *is just a collection of (joint) random variables $X(t)$, one for each real $t \geq 0$. It is called a* counting *process if: (i) each $X(t)$ has range $\mathbb{N}$; (ii) $s \leq t$ implies $X(s) \leq X(t)$ always.*

Definition 9. *A Poisson Process with parameter $\lambda > 0$ can alternately be defined as follows: Its arrival counts $N(t)$ are a continuous-time counting stochastic process satisfying:*

1. *$N(0) \equiv 0$.*
2. *Independent increments: if $(t_1, t_2]$ and $(t_3, t_4]$ are disjoint, then $N(t_2) - N(t_1)$ and $N(t_4) - N(t_3)$ are independent.*
3. *$N(s+t) - N(s) \sim$ Poisson(λt) for all $s, t \geq 0$.*

Let us show that this definition based on the arrival counts $N(t)$ implies the other definition based on interarrival times.

Proposition 10. *The definition based on arrival counts implies the one based on interarrival times.*

Proof. Let's look at how the first interarrival time S_1 depends on the $N(t)$'s:

$$S_1 > t \quad \Leftrightarrow \quad N(t) = 0 \quad \Leftrightarrow \quad N(t) - N(0) = 0,$$

where we used Property 1 above. So using Property 3 above,

$$\mathbf{Pr}[S_1 > t] = \mathbf{Pr}[\text{Poisson}(\lambda t) = 0] = \exp(-\lambda t) \quad \Rightarrow \quad F_{S_1}(t) = \mathbf{Pr}[S_1 \leq t] = 1 - \exp(-\lambda t).$$

This is the CDF of an Exponential(λ), hence indeed $S_1 \sim$ Exponential(λ).

Let's look at the second interarrival time S_2, conditioned on $S_1 = s$.

$$\begin{aligned}
\mathbf{Pr}[S_2 > t \mid S_1 = s] &= \mathbf{Pr}[N(s+t) - N(s) = 0 \mid S_1 = s] \\
&= \mathbf{Pr}[N(s+t) - N(s) = 0 \mid N(s) - N(0) = 1, N(s') - N(0) = 0 \ \forall s' < s] \\
&= \mathbf{Pr}[N(s+t) - N(s) = 0] \qquad \text{(independent increments)} \\
&= \mathbf{Pr}[\text{Poisson}(\lambda t) = 0] \\
&= \exp(-\lambda t),
\end{aligned}$$

and notice this does not depend on s. We conclude that S_2 also has the Exponential(λ) distribution and is independent of S_1. We can make similar deductions for $S_3, S_4, S_5, \ldots$. □

2.4 An even more general definition

The second definition of the Poisson Process obviously has Poissons built right into it. And the first definition has Exponentials built into it. Actually, the "traditional" definition of the Poisson Process has no specific distribution built into it; just some simple axioms about arrivals. Writing $N(t)$ for the number of arrivals up to time t, the axioms are:

1. At the beginning there are no arrivals: $N(0) \equiv 0$.
2. Independent increments; i.e., the number of arrivals in one interval does not affect the number in a disjoint interval.
3. The "average" arrival rate is a constant λ; more precisely, for all $t \geq 0$,
$$\mathbf{Pr}[N(t+\delta) - N(t) = 1] = \lambda\delta + o(\delta).$$
4. Arrivals occur "one at a time":
$$\mathbf{Pr}[N(t+\delta) - N(t) \geq 2] \leq o(\delta).$$

(Remember, $o(\delta)$ represents some function of δ, call it $h(\delta)$, with the property that $h(\delta)/\delta \to 0$ as $\delta \to 0$; informally, $h(\delta) \ll \delta$.)

As you can see, this is very much like our intuitive model of the Poisson Process wherein each tiny time segment of length δ flips a $(\lambda\delta)$-biased coin and has an arrival if the coin lands heads. Indeed, with a page or two of basic calculus, you can prove the following:

Theorem 11. *Any stochastic counting process satisfying the above axioms must be the Poisson Process with rate λ.*

So you see, the Poisson Process can arise quite naturally, with no reference to either Poissons or Exponentials.

3 Poisson Process merging and splitting

Having three equivalent definitions of the Poisson Process lets us be very flexible when proving facts about the Poisson Process. Let's prove two basic such facts.

3.1 Poisson Process merging

Imagine the 61A bus arrives at the bus stop in front of The Cut according to a Poisson Process with rate λ_A. Imagine the 61B bus also arrives at this bus stop according to an *independent* Poisson Process with rate λ_B. If you are going to the corner of Forbes and Murray, you can equally well use either bus; so, you really only care about the arrival times of any kind of bus. This process is called the *merging* of two Poisson Processes.

By the way:

Definition 12. *We say Poisson Processes A and B are independent if their interarrival time random variables are independent. Equivalently, the arrival counts $N_A(t)$ and $N_B(u)$ should be independent for all $t, u \geq 0$.*

You may not be surprised to learn:

Theorem 13. *If we merge two independent Poisson Processes with rates λ_A and λ_B, the result is a Poisson Process with rate $\lambda_A + \lambda_B$. (More generally, merging n independent Poisson Processes with rates $\lambda_1, \ldots, \lambda_n$ yields a Poisson Process with rate $\lambda_1 + \cdots + \lambda_n$.)*

Proof. Since we have several ways of thinking about Poisson Processes, there are several ways to prove this. (You should try thinking about them!) Here is the way to prove it based on interarrival times. In the merged process, the first arrival occurs after a period of time U_1 which is the minimum of an Exponential(λ_A) and an independent Exponential(λ_B). As we saw on the homework, $U_1 \sim$ Exponential($\lambda_A + \lambda_B$). Let's think now about conditioning on this first arrival time. What subsequently happens in Process A? Well, if the first arrival was in A, then the distribution on its interarrival time is another Exponential(λ_A). And if the first arrival was in B, then the distribution on the subsequent time for an arrival in Process A is *still* Exponential(λ_A), by the memorylessness property of exponentials. So even conditioned on U_1, Process A continues to act like a Poisson Process with rate λ_A. The exact same considerations hold for Process B. Hence the next interarrival time, U_2, is again the minimum of an Exponential(λ_A) and an independent Exponential(λ_B) — which is Exponential($\lambda_A + \lambda_B$) — and this is also independent of U_1. Etc. Thus the interarrival times in the merged process are independent Exponential($\lambda_A + \lambda_B$) random variables, as required. □

3.2 Poisson Process splitting

The opposite of Poisson Process merging is Poisson Process splitting, and it is slightly more surprising. As an example, suppose you have a server with two processors. Jobs arrive at the server according to a Poisson Process with rate λ. You use a *randomized* load-balancing policy, wherein an arriving job is assigned to the first processor with probability p and the second processor with probability $1 - p$ (independently for each job).

You can now consider the jobs arriving at processor 1 to be one process, and the jobs arriving at processor 2 to be a separate (but presumably dependent) process.

Theorem 14. *In this scenario, the jobs arriving at processor* 1 *form a Poisson Process with rate $p\lambda$, and the jobs arriving at processor* 2 *form a Poisson Process with rate $(1-p)\lambda$. Furthermore, these processes are* independent*!*

Remark 15. *This is one of those rare times when you have random variables which are generated together and thus don't* look *independent (according to the "Principle", at least). But, as it turns out, they* are *independent, according to the official definition. This independence actually proves to be very useful!*

Proof. Let's start with just showing that the jobs arriving at processor 1 for a Poisson Process with rate $p\lambda$. What do the interarrival times look like for the jobs at processor 1? We start with independent Exponential(λ) interarrival times S_1, S_2, S_3, ... However some of these arrivals go to processor 2. Let G be the random variable denoting the index of the first job that goes to processor 1. Then it's clear that $G \sim$ Geometric(p) (and G is independent of the S_i's). Thus the arrival time for the first processor 1 job is

$$U_1 = S_1 + S_2 + \cdots + S_G.$$

Lemma 16. *The sum of a geometric number of exponentials is exponential. Specifically,* $U_1 \sim$ Exponential($p\lambda$).

Proof. We give an intuitive proof, based on thinking of exponentials as the continuous version of geometrics. You will do a rigorous proof on the homework. Let C be a biased coin with heads-probability $\lambda\delta$, where δ is tiny. Intuitively, you can think of $S_1 \sim$ Exponential(λ) as being the time it takes to get a heads when you flip C every δ-length time tick. Suppose you have another coin D with heads-probability p, and every time you flip C you also flip D. As long as C flips tails, you ignore D, but whenever C flips heads, you look at the outcome of D. If D is *also* heads, you stop. If D is tails, you continue flipping both coins.

The stretch of time till the first C-heads is distributed like S_1. When that stretch ends, whether or not you continue depends on a D-flip. If it's tails, you get another stretch of time distributed like $S_2 \sim$ Exponential(λ), and another D-flip, etc. Hence, the total time till you flip heads on both coins is distributed like U_1, a geometric sum of exponentials. And if you think of the two coins C and D together as being like a single coin with heads-probability $p\lambda$, you see that $U_1 \sim$ Exponential($p\lambda$). □

This shows that the first interarrival time for jobs to processor 1 has distribution Exponential($p\lambda$). It's clear that the subsequent interarrival times are also independent Exponential($p\lambda$) random variables. Hence indeed, the jobs arriving at processor 1 form a Poisson Process with rate $p\lambda$.

Similarly, the jobs arriving at processor 2 form a Poisson Process with rate $(1-p)\lambda$.

It remains to show that these are actually *independent* Poisson Processes. To do this, we will show that $N_1(t)$ and $N_2(u)$ are independent for all $t, u \geq 0$. Let's start with the case $t = u$:

Lemma 17. *$N_1(t)$ and $N_2(t)$ are independent for all $t \geq 0$.*

Proof. Suppose we fix some time t. To show that $N_1(t)$ and $N_2(t)$ are independent, we need to show that their joint PMF is the product of their individual PMFs; i.e., for all $a, b \in \mathbb{N}$,

$$\mathbf{Pr}[N_1(t) = a \ \&\ N_2(t) = b] = \mathbf{Pr}[N_1(t) = a]\mathbf{Pr}[N_2(t) = b].$$

Now

$$\mathbf{Pr}[N_1(t) = a \ \&\ N_2(t) = b] = \mathbf{Pr}[N(t) = a + b \ \&\ N_1(t) = a],$$

where $N(t)$ is the arrival count for jobs to the server. We have $N(t) \sim \text{Poisson}(\lambda)$, so $\mathbf{Pr}[N(t) = a+b] = \exp(-\lambda)\lambda^{a+b}/(a+b)!$. Conditioned on $N(t) = a+b$, what is the distribution of $N_1(t)$? We have $a+b$ jobs arrived in the time span, and each gets sent to processor i independently with probability p. So $N_1(t) \mid (N(t) = a+b)$ has distribution Binomial$(a+b, p)$. Thus

$$\begin{aligned}
\mathbf{Pr}[N(t) = a+b \ \&\ N_1(t) = a] &= \exp(-\lambda)\frac{\lambda^{a+b}}{(a+b)!} \cdot \frac{(a+b)!}{a!b!}p^a(1-p)^b \\
&= \exp(-p\lambda)\frac{(p\lambda)^a}{a!} \cdot \exp(-(1-p)\lambda)\frac{((1-p)\lambda)^b}{b!} \\
&= \mathbf{Pr}[N_1(t) = a]\mathbf{Pr}[N_2(t) = b],
\end{aligned}$$

because $N_1(t) \sim \text{Poisson}(p\lambda)$ and $N_2(t) \sim \text{Poisson}((1-p)\lambda)$. This completes the proof. □

Finally, since we know $N_1(t)$ and $N_2(t)$ are independent for all t, it's not too hard to show that $N_1(t)$ and $N_2(u)$ are independent as well. It's a bit of a trick, using the "independent increments" property of the Poisson Process. Assume without loss of generality that $u > t$. Now

$$\mathbf{Pr}[N_1(t) = a \ \&\ N_2(u) = b] = \sum_{c=0}^{b} \mathbf{Pr}[N_1(t) = a \ \&\ N_2(t) = c \ \&\ (N_2(u) - N_2(t) = b - c)],$$

by partitioning according to the number of arrivals at processor 2 by time t. By the "independent increments" property, the arrivals to the server in the time segment $(0, t]$ are independent of the arrivals in time segment $(t, u]$. Hence

$$\begin{aligned}
&\mathbf{Pr}[N_1(t) = a \ \&\ N_2(t) = c \ \&\ (N_2(u) - N_2(t) = b - c)] \\
&\qquad = \mathbf{Pr}[N_1(t) = a \ \&\ N_2(t) = c]\mathbf{Pr}[N_2(u) - N_2(t) = b - c].
\end{aligned}$$

Since $N_1(t)$ and $N_2(t)$ are independent, this equals

$$\mathbf{Pr}[N_1(t) = a]\mathbf{Pr}[N_2(t) = c]\mathbf{Pr}[N_2(u) - N_2(t) = b - c].$$

Again, independent increments implies that $N_2(t)$ and $N_2(u) - N_2(t)$ are independent, so going back again this equals

$$\mathbf{Pr}[N_1(t) = a]\mathbf{Pr}[N_2(t) = c \ \&\ N_2(u) - N_2(t) = b - c].$$

So we have shown

$$\begin{aligned}
\mathbf{Pr}[N_1(t) = a \ \&\ N_2(u) = b] &= \sum_{c=0}^{b} \mathbf{Pr}[N_1(t) = a]\mathbf{Pr}[N_2(t) = c \ \&\ N_2(u) - N_2(t) = b - c] \\
&= \mathbf{Pr}[N_1(t) = a] \sum_{c=0}^{b} \mathbf{Pr}[N_2(t) = c \ \&\ N_2(u) - N_2(t) = b - c] \\
&= \mathbf{Pr}[N_1(t) = a]\mathbf{Pr}[N_2(u) = b],
\end{aligned}$$

and hence $N_1(t)$ and $N_2(u)$ are indeed independent. □

15-359: Probability and Computing
Fall 2009

Lecture 20: Continuous-Time Markov Chains

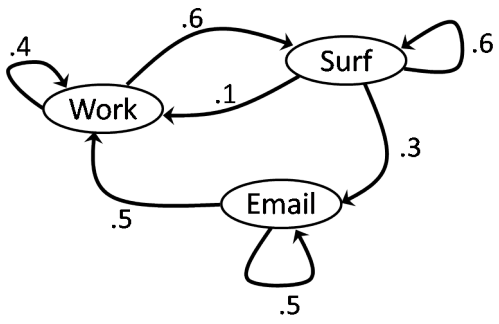
Continuous-Time Markov Chains are a very natural variant of Discrete-Time Markov Chains, and are important for the study of queuing systems. We will describe them in this lecture.

1 Continuous-Time Markov Chains

Let's think about how to define *continuous-time Markov Chains (CTMCs)*.

1.1 The Markov Property

I hope you remember when we studied discrete-time Markov Chains, there were two very different-looking — but equivalent — definitions. The first was a sort of "hands-on" definition: you had a diagram like this,



with corresponding transition matrix K. In the associated random process, you jumped from state to state each time tick according to the transition probabilities.

The alternate definition was more "abstract":

Definition 1. *A discrete-time Markov Chain with (countable) state set S is a discrete-time stochastic process — i.e., a sequence of S-valued random variables X_0, X_1, X_2, ... — with the* Markov Property*: for all $t \in \mathbb{N}$ and $u, v \in S$,*

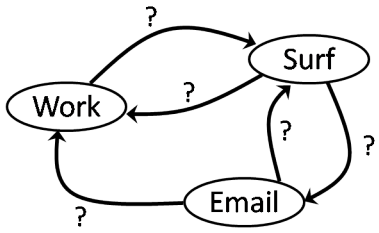
$$\mathbf{Pr}[X_{t+1} = v \mid X_t = u \text{ \& values of } X_{t'},\ t' < t] = \mathbf{Pr}[X_{t+1} = v \mid X_t = u] = K[u, v].$$

I.e., given that you are in state u at time t,

- it does not matter how you got there, and
- it does not matter what the time t is,

your probability of being at state v at time $t+1$ is just $K[u,v]$.

A somewhat non-realistic aspect of my "Work-Surf-Email" chain above was that it imagined I jumped from state to state at discrete time intervals (every minute). In reality, my time is fluid, and I might jump from state to state at any point in (continuous) time. In continuous time, what should the picture look like?



[1]

Hmm. Since it is not immediately obvious how to make a continuous-time version of the "hands-on" definition, let's consider a continuous-time analogue of the "abstract" definition is:

Definition 2. *A* continuous-time Markov Chain *with (countable)*[2] *state set S is a continuous-time stochastic process — i.e., a collection of S-valued random variables X_t, for $t \in \mathbb{R}^{\geq 0}$ — which satisfy the following* Markov Property*: for all $s,t \geq 0$ and $u,v \in S$,*

$$\mathbf{Pr}[X_{t+s} = v \mid X_t = u \text{ \& values of } X_{t'},\ t' < t] = \mathbf{Pr}[X_{t+s} = v \mid X_t = u] = P_s[u,v].$$

I.e., given that you are in state u at time t, it does not matter how you got there, and it doesn't matter what t is, the probability that in time s you are in state v is some number $P_s[u,v]$ depending only on u, v, and s.

Please note, by the way, that each random variable X_t is a *discrete* random variable; it takes values in the countable set S. It's just that we have uncountably many of them. Actually, we will not take Definition 2 too seriously, because it's not exactly clear what it means to condition on uncountably many events $X_{t'} = w$ simultaneously. Instead, we'll just use it as a muse when coming up with the more "hands-on" definition of continuous-time Markov Chains.

1.2 Waiting times

Suppose I am in state u at time t_0. Presumably, for a while I will continue to be in state u; then I'll jump to being in some other state. The amount of time I stay in state u before jumping will be a *continuous* random variable. Let's call it

$$U = \text{amount of time until I change states, given that I'm in state } u \text{ at time } t_0.$$

What does Definition 2 suggest about this random variable U? Could we understand, say,

$$\mathbf{Pr}[U > t]?$$

It's not obvious... But instead, suppose we asked about

$$\mathbf{Pr}[U > t+s \mid U > s].$$

[1] As you can see from the picture, I've decided that I sometimes go from email to surfing.

[2] We will only ever deal with finite or countable state sets.

I.e., what is the probability that it takes longer than $t + s$ time to jump away from state u, given that it took longer than s? Well,

$$\text{“}U > s\text{”} \quad \Leftrightarrow \quad X_{t'} = u \ \forall t_0 \leq t' \leq t_0 + s.$$

But by the Markovian Property, the fact that you were in state u for all times between t_0 and $t_0 + s$ doesn't matter; all that matters is that you were in state u at time $t_0 + s$. Since the Markovian Property also doesn't care about the difference between being in state u at time t_0 and being in state u at time $t_0 + s$, we ought to have

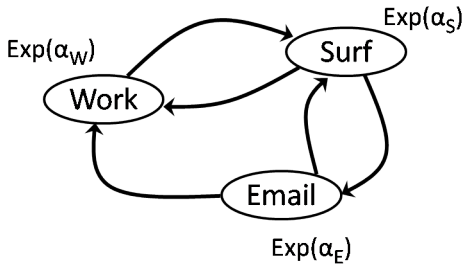
$$\mathbf{Pr}[U > t + s \mid U > s] = \mathbf{Pr}[U > t].$$

I.e, U should have the Memorylessness Property! As you will show on the homework, the *only* continuous random variables which have the Memorylessness Property are exponential random variables. Hence we deduce:

In a continuous-time Markov Chain, the amount of time you wait at each state has an exponential distribution.

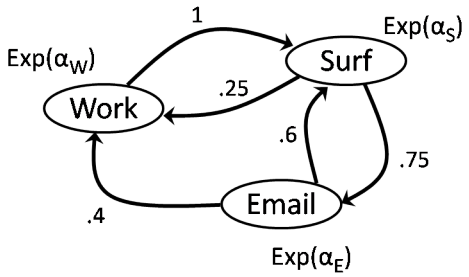
1.3 A hands-on definition

Great! We're well on our way to a nice definition now. First, there's no reason why it should be the *same* exponential distribution for each state. So let's assume that for each state u, I wait for some time with distribution Exponential(α_u).



Well, now as soon as I'm done "waiting" in a state u, I must hop to another state. According to the Markov Property, the probability of me hopping to each other state v should only depend on the states and not on any times. So just as with discrete-time Markov Chains, we could have a probability $p[u, v]$ for each possible arrow.

Note: There is no point in having self-loops. If you were thinking about having hops from u to u, you may as well just increase the waiting time at u by decreasing the rate α_u!



In this diagram we have, e.g., $p[E, W] = .4$. Of course, the sum of the probabilities coming out of each state needs to be 1. We now can make a "hands-on" definition of CTMCs:

Definition 3. *A* continuous-time Markov Chain *with (countable) state set S consists of:*

- *a discrete-time Markov Chain over S with transition matrix P (and no self-loops), called the* skeleton chain*;*
- *a "rate" $\alpha_u > 0$ for each state u.*

We associate the following random process:

`// the initial state,` X_0 `, is some random variable with values in` S
$t \leftarrow 0$
$u \leftarrow X_0$
`loop forever`
  $U \leftarrow \text{Exponential}(\alpha_u)$
  $X_s \leftarrow u$ `for all` $t < s < t + U$
  $t \leftarrow t + U$
  `choose` $V \in S$ `by assigning` $V \leftarrow v$ `with probability` $p[u, v]$
  $X_t \leftarrow V$
  $u \leftarrow V$

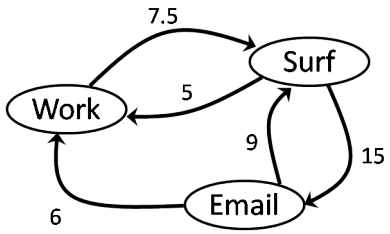
E.g., in the above chain, suppose I start with $X_0 =$ `Work`. First I work for some Exponential(α_W) amount of time. Then I switch to Surfing with probability 1. Then I surf for some Exponential(α_S) amount of time. Then I go back to Work with probability .25 and I switch to Email with probability .75. Etc.

Question: Suppose the rates α_u are all the same across $u \in S$. What can you say about the transition times in the CTMC?

Answer: They form a Poisson Process.

1.4 Alarm clocks

The above definition of a CTMC is a perfectly good one, and is the one you'd probably use if you were writing a simulator. However, if you look around, you will find that people do not draw CTMCs in the above way, with each state labeled by a waiting rate and each arrow labeled by a probability. Instead, they draw them like this, with no numbers on the states, and arbitrary real numbers on the arrows:

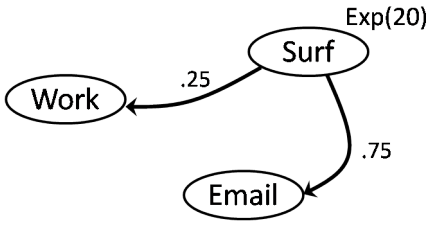


What does this mean?!

Let's go back to the previously defined Work/Surf/Email chain and focus on the Surf state. Let's measure time in hours. Suppose we decide that every time I start surfing, I spend 3 minutes doing it on average. (I may be afflicted with Attention Deficit Disorder...)

Question: What should α_S be?

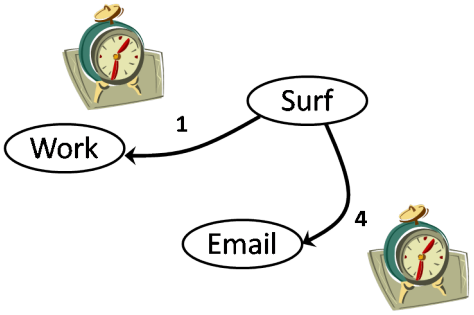
Answer: We want the mean of the Exponential(α_S) random variable to be 1/20 (because 3 minutes is 1/20 of an hour). Thus α_S should be 20. If you think of this 20 as a *rate*, it's saying that I transition *out* of the Surf state at the "rate of 20 transitions per hour".



As you recall, we have that once I transition out of Surfing, I go to Email with probability .75 and to Work with probability .25.

Why? Why might I do this? Here's a plausible explanation: As I'm surfing, one of two things can happen. I can receive an Email, in which case I promptly go address it. Or, I can feel a guilt pang, in which case I get back to work. It's a bit like two **alarm clocks** might go off as I'm surfing: the Email alarm clock and the Work (AKA guilt pang) alarm clock. Whichever one buzzes first draws me into its state.

Indeed, something quite neat happens if we model the waiting times for emails and guilt pangs according as exponential random variables. Suppose we think of a new email arriving in exponential time with parameter 15 (hence mean $60/15 = 4$ minutes) and a guilt pang as arriving in exponential time with parameter 5 (hence mean $60/5 = 12$ minutes). Here is picture:



(I agree that it's not clear why the arrows should point this way and not the opposite way. But this is the way we'll draw it.)

Interpretation: While I'm in the Surf state, there is an Email "alarm clock" that buzzes after a random amount of time with distribution Exponential(15). There is also an (independent) Work alarm clock that buzzes after Exponential(5) time. As soon as one of them buzzes, I go to its state.

Question: In this alarm clock scenario, what is the distribution on time I spend in the Surf state?

Answer: It is the *minimum* of two exponential random variables, with parameters 15 and 5. As we know, the minimum of two exponentials is exponential, with the parameters adding. So the amount of time I spend in the Surf state has distribution Exponential(20) — just as desired!

Question: In the alarm clock scenario, when I eventually transition out of the Surf state, with what probability do I go to Email and with what probability to Work?

Answer: I go to Email if and only if the Email alarm clock buzzes first. I.e., if the alarm times are $A_E \sim$ Exponential(15) and $A_W \sim$ Exponential(5), I go to Email if and only if $\min(A_E, A_W) = A_E$. But remember from last lecture:

Theorem 4. *If $X_1, \ldots, X_n$ are independent random variables with $X_i \sim$ Exponential(λ_i), then*

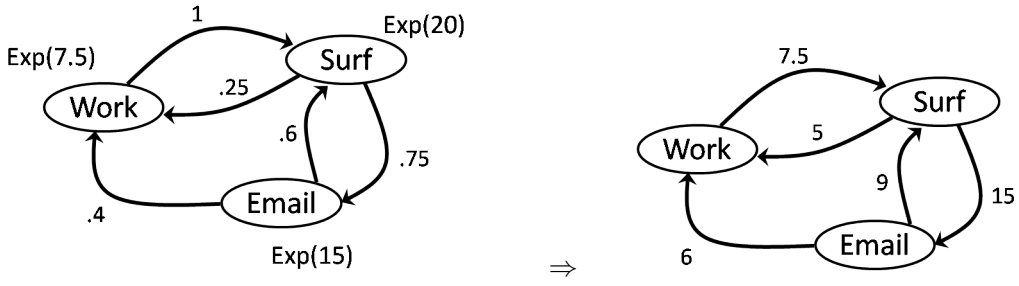
$$\mathbf{Pr}[\min(X_1, \ldots, X_n) = X_i] = \frac{\lambda_i}{\lambda_1 + \cdots + \lambda_n}.$$

So the probability the Email alarm buzzes first is $\frac{15}{15+5} = .75$ and the probability the Work alarm buzzes first is $\frac{5}{15+5} = .25$, just as desired.

In other words, this alarm clock scenario exactly models the original setup of spending Exponential(20) time in the Surf state, then going to Email with probability .75 and Work with probability .25.

1.5 The alarm clock definition

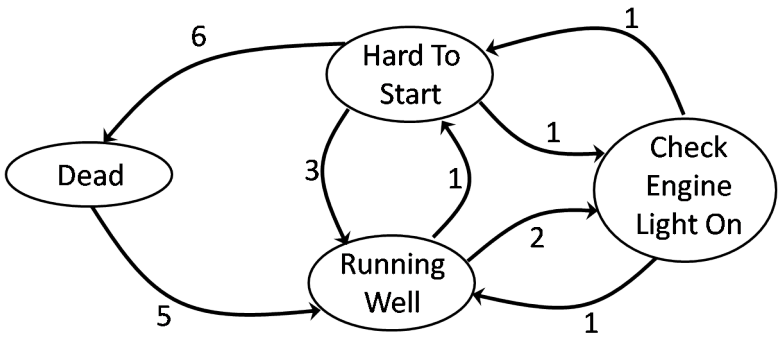
More generally: Instead of drawing CTMCs like the one on the left, we always draw them like the one on the right:



Definition 5. *(Alarm clock definition.) A continuous-time Markov Chain on state set S is defined by a transition diagram such as the one on the right, above (with no self-loops). The directed arc from state u to state v is labeled with a positive number $\alpha[u, v]$ which denotes the* rate of transition from u to v. *We associate the following process:*

Suppose you are in state u. For each arc (u, v), we imagine an independent "alarm clock" for state v which buzzes after an Exponential$(\alpha[u, v])$ *amount of time. Whenever one of the alarm clocks buzzes, we transition to its state.*

Example: Here is how my old car used to treat me. (Think of time in months.)



Question: Once I manage to get my car running, what is the expected amount of time till something goes wrong with it?

Answer: When in the Running Well state, we think of an Exponential(1) alarm clock going into "Hard To Start" and an Exponential(2) alarm clock for going into "Check Engine Light On". Hence the distribution on the time till one of these buzzes (i.e., something goes wrong) is Exponential($1+2$) $\sim$ Exponential(3). The mean of an Exponential(3) random variable is $1/3$. Hence the average amount of time till something goes wrong is $1/3$ of a month; i.e., about 10 days.

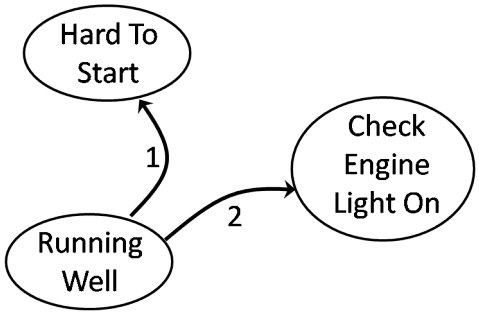
2 Stationary Distributions

Just as with discrete-time Markov Chains, we're frequently interested in the long-time behavior of CTMCs. For example, in the long run, what fraction of the time is my car Running Well (R)? Is there a "steady-state", or stationary distribution, such that if I'm in that distribution on states, I will remain in the same distribution for all time?

To answer these questions rigorously for CTMCs involves some rather long and unenlightening detours that I don't think it's worth getting into. Let me just show you how you can obtain the right answers for finite-state CTMCs and most natural infinite-state chains. The idea is to take the view of exponentials as the limit as $\delta \to 0$ of "coin flips every δ-length time tick", and convert to a discrete-time Markov Chain!

2.1 CTMCs to DTMCs

Let's take my old car's CTMC and focus on state R (= Running Well).

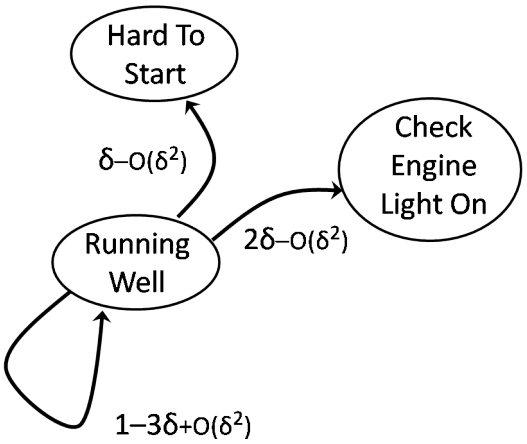


There are two states it can go to, H (= Hard To Start) and C (= Check Engine Light On). We imagine an alarm clock for each; the H clock buzzes after Exponential(1) time, the C clock buzzes after Exponential(2) time. We imagine these exponential random variables as follows. Time is chopped up into tiny segments of length δ. On each time step independently, the H clock buzzes with probability $1 \cdot \delta$ and the C clock buzzes with probability $2 \cdot \delta$. If neither clock buzzes, you stay put. If one clock buzzes, you go to its state. If both clocks buzz... well, there's only a freakishly small chance of that, $2\delta^2$. Generally, we think of δ as tiny and hence anything $O(\delta^2)$ is unimaginably tiny; we will end up neglecting $O(\delta^2)$ quantities. For definiteness, we'll assume that if both clocks buzz you stay put.

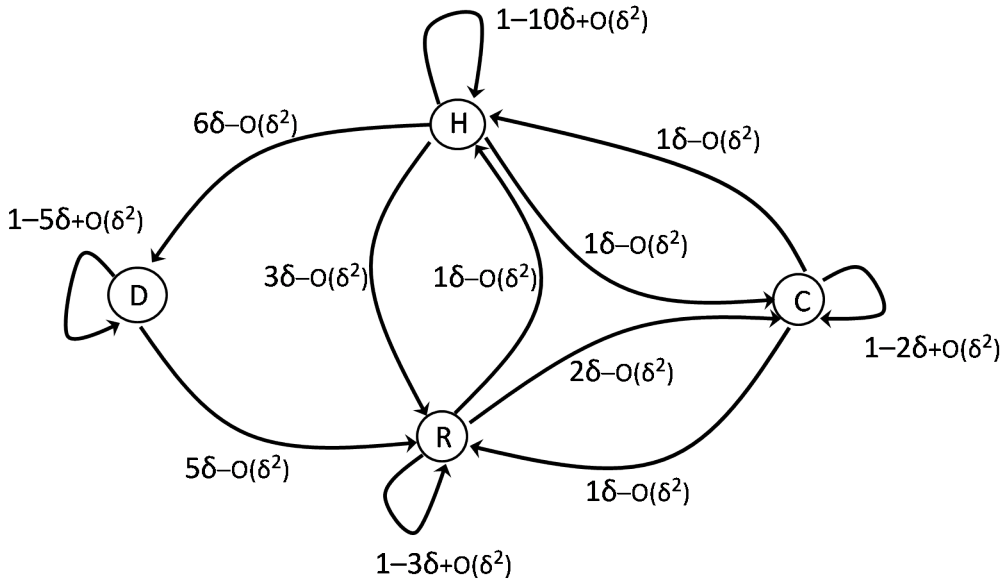
To be precise,

$$\begin{aligned}
\mathbf{Pr}[C \text{ buzzes and } H \text{ does } \textit{not} \text{ buzz}] &= 2\delta(1-\delta) = 2\delta - O(\delta^2). \\
\mathbf{Pr}[H \text{ buzzes and } C \text{ does } \textit{not} \text{ buzz}] &= \delta(1-2\delta) = \delta - O(\delta^2). \\
\mathbf{Pr}[\text{stay put}] = \mathbf{Pr}[0 \text{ or } > 1 \text{ buzzes}] &= 1 - \mathbf{Pr}[\text{exactly 1 buzz}] \\
&= 1 - (2\delta - O(\delta^2)) - (\delta - O(\delta^2)) = 1 - 3\delta + O(\delta^2).
\end{aligned}$$

Hence we discretize this part of the chain to the following:



By analogous reasoning, we can discretize the whole chain as follows:



We can now solve this DTMC!

Question: Will the DTMC be *regular*? I.e., will it be 'irreducible' and 'aperiodic'?

Answer: Irreducibility is always easy to check: it's just checking whether the graph is totally connected. As for aperiodicity, something very nice happens. The DTMC will ***always be aperiodic***. Why? Because there are self-loops on every state! Hence if you can go from state u to state v at in T steps, then you can also do it in $T+s$ steps for every $s \geq 0$. So cycling is not possible.

This is very nice, because (at least for finite-state chains) it means the only thing you have to worry about is reducibility/disconnectedness, which is easy to check.

2.2 Balance equations

So assuming the graph is connected (and finite), the Fundamental Theorem tells us that there is a unique stationary distribution, it has nonzero probability for each state, and it is the long-time limiting distribution. We solve for it via the *stationary equations*:

$$\pi = \pi K \qquad \text{(and also } \sum_{u \in S} \pi[u] = 1\text{)}.$$

Let's think about these equations a little bit abstractly, before we go plugging any numbers in.

Key insight: When there are self-loops with huge probabilities (as there are when converting CTMCs to DTMCs), you should think about the stationary equations a little differently.

Specifically, let's take one of the equation for one state, say R:

$$\pi[R] = \sum_u \pi[u] K[u, R].$$

Let's separate the big 'self-loop' quantity, $K[u,u]$:

$$\pi[R] = \sum_{u \neq R} \pi[u]K[u,R] + \pi[R]K[R,R] \quad \Rightarrow \quad \pi[R](1 - K[R,R]) = \sum_{u \neq R} \pi[u]K[u,R].$$

From state R, you have to go somewhere:

$$\sum_{v} K[R,v] = 1$$

(I.e., all rows of K add up to 1.) Hence

$$1 - K[R,R] = \sum_{v \neq R} K[R,v].$$

So we conclude

$$\sum_{v \neq R} \pi[R]K[R,v] = \sum_{u \neq R} \pi[u]K[u,R]. \tag{1}$$

Definition 6. *The equation 1 is called the* balance equation *for state R.*

Note: This is *NOT* the same as the time-reversibility equations!

Interpretation: The quantity on the left is the long-term *fraction of transitions leaving R*. The quantity on the right is the long-term *fraction of transitions entering R*. (In fact, it's obvious that this equation must hold in the long-term limit: every time you enter R, you must leave R, and vice versa. So in a long run of a DTMC, the number of entrances to R must differ from the number of leavings from R by at most 1. Hence we can deduce (1) the same way we reasoned about the time-reversibility equations in Lecture 15 Section 6.)

Note: As we just saw, the stationary equation for state R is easily equivalent to the balance equation for state R. You're just subtracting the self-loop rate from both sides. So solving the stationary equations and solving the balance equations is equivalent.

2.3 The balance equations for discretized CTMCs

The reason we bring up the balance equations is that when you discretize a CTMC to a DTMC, the balance equations are much easier to solve. Let's observe this in the old car chain. The rate of transitions entering R — i.e., the LHS of (1) — is

$$\pi[D](5\delta - O(\delta^2)) + \pi[H](3\delta - O(\delta^2)) + \pi[C](1\delta - O(\delta^2)) = 5\delta\pi[D] + 3\delta\pi[H] + 1\delta\pi[C] - O(\delta^2).$$

The rate of transitions leaving R — i.e., the RHS of (1) — is

$$\pi[R](1\delta - O(\delta^2) + 2\delta - O(\delta^2)) = 3\delta\pi[R] - O(\delta^2).$$

So the balance equation for state R is

$$5\delta\pi[D] + 3\delta\pi[H] + 1\delta\pi[C] - O(\delta^2) = 3\delta\pi[R] - O(\delta^2).$$

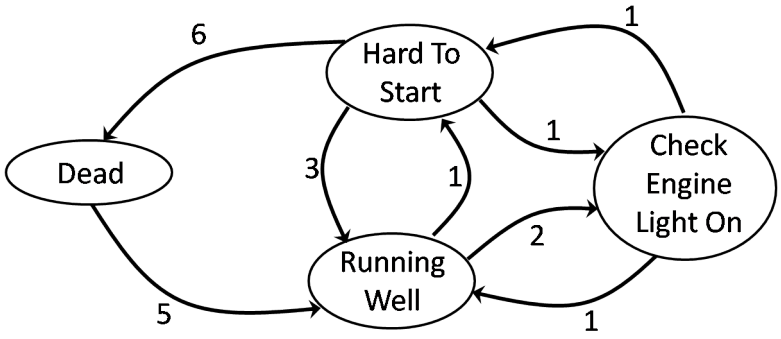
But now we can divide by δ! This gives

$$5\pi[D] + 3\pi[H] + 1\pi[C] - O(\delta) = 3\pi[R] - O(\delta).$$

The $O(\delta)$'s now become negligible as $\delta \to 0$, and we just get

$$5\pi[D] + 3\pi[H] + 1\pi[C] = 3\pi[R]. \tag{2}$$

Let's look at this in light of the original CTMC:



We see that the balance equation (2) for R is easily read off of the CTMC diagram, just by saying that the "rate of transitions into state R" should equal the "rate of transitions out of state R". **Even though the rates in the CMTC diagram are not probabilities, this still works.**

The remaining balance equations, for states H, C, and D respectively, are:

$$\begin{aligned} 1\pi[C] + 1\pi[R] &= 10\pi[H] \\ 1\pi[H] + 2\pi[R] &= 2\pi[C] \\ 6\pi[H] &= 5\pi[D]. \end{aligned}$$

When we also include the balance equation (2), as well as the usual

$$\pi[R] + \pi[H] + \pi[C] + \pi[D],$$

we can solve for the stationary distribution. This turns out to give

$$\pi[R] \approx .39, \quad \pi[C] \approx .43, \quad \pi[H] \approx .08, \quad \pi[D] \approx .10.$$

I.e., in the long run, my car is Running Well about 39% of the time, running with the Check Engine light on about 43% of the time, being Hard To Start about 8% of the time, and Dead about 10% of the time. Yeah, that's about how I remember it.

2.4 Summary

Theorem 7. *Suppose we have a CTMC on finite state set S which is irreducible (every state can reach every other state). Then there is a unique solution π to the* balance equations

$$\sum_{w \neq u} \pi[w]\alpha[w,u] = \sum_{v \neq u} \pi[u]\alpha[u,v] \quad \forall u \in S,$$

$$\sum_{u \in S} \pi[u] = 1.$$

This probability vector has $\pi[u] > 0$ for all states u, and represents the long-time limiting probability of being in each state.

For irreducible CTMCs on infinite state sets, if *the balance equations have a solution with $\pi[u] > 0$ for all $u \in S$, then the solution is unique and these are the limiting probabilities.*

15-359: Probability and Computing
Fall 2009

Lecture 21: Intro to Queuing Theory

1 Recap of the main theorem for CTMCs

For this lecture, we'll need to recall the theorem about continuous-time Markov Chains which we ended on last time:

Theorem 1. *Suppose we have a CTMC on finite state set S which is irreducible (every state can reach every other state). Then there is a unique solution π to the* balance equations

$$\sum_{w \neq u} \pi[w]\alpha[w,u] = \sum_{v \neq u} \pi[u]\alpha[u,v] \quad \forall u \in S,$$
$$\sum_{u \in S} \pi[u] = 1.$$

This probability vector has $\pi[u] > 0$ for all states u, and is both the stationary distribution and the long-time limiting probability of being in each state.

*For irreducible CTMCs on infinite state sets, **if** the balance equations have a solution with $\pi[u] > 0$ for all $u \in S$, then the solution is unique and these are the steady-state/limiting probabilities.*

2 Queuing Theory

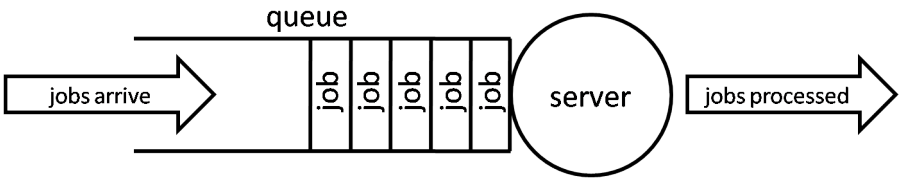
This lecture is about *Queuing Theory*. Queuing Theory — the study of waiting in lines — is a major branch of mathematical computer science/operations research. It was originally developed mainly by people working on telephone systems (!), especially in the 1950s. Today it's commonly used to study e.g., requests to a web server, packets coming into routers, threads queuing for critical sections, processes arriving at CPUs — as well as real-world queues of people at hospitals, grocery stores, etc.

The hardest part about Queuing Theory is, of course, spelling 'queuing'. Luckily, 'queuing' and 'queueing' are both considered acceptable spellings (the former is more American, the latter is more British). Yes, the latter version has five (5) consecutive vowels, and is the only common English word with this property.

2.1 Queuing theory terminology and diagrams

Queuing theory has its own set of funny notation and diagrams.

The things that queue up to be processed are called **jobs**.
The things that process them are called **servers**. If there's more than one, it's a **server network**.
These items are usually drawn as follows:



Queuing systems are characterized by the following choices:

Distribution of job *interarrival* times.
Distribution of job *service* times. Usually the r.v. $\boldsymbol{S}$ denotes the time a job takes to process.
Number of servers. Usually denoted $\boldsymbol{m}$.
Capacity of each server's queue.
Total number of jobs.
Service policy: e.g., **FCFS** (= First-Come First-Served = First-In First-Out).

Today we will focus on the *simplest realistic queuing model*, which is called **M/M/1**; aka, the single-server Markovian queue.

Question: What does this mean? What is up with the bizarre terminology 'M/M/1'?

Answer: This is "Kendall notation", invented by the probabilist David G. Kendall. The first 'M' stands for Exponential.[1] This means that the interarrival times are Exponential(λ) for some parameter λ. I.e., they form a Poisson Process of rate λ. The second 'M' stands for Exponential as well. It means that each job requires an Exponential(μ) amount of time to process, for some parameter μ. The '1' means there is one server. The other three parameters are assumed to be their "default" values: the server has a potentially infinite queue, the number of jobs arriving jobs is unbounded, and the service policy is FCFS.[2]

Question: What about server speeds?

Answer: Good question. In queuing theory, you don't talk about server speeds. Instead, you *fold this into the service time distribution*. E.g., if you want to imagine a server being sped up, instead you imagine the service time (distribution) being shrunk.

The *parameters* of a queuing system:

$\boldsymbol{\lambda}$: the (long-term) average number of arrivals per unit time.
$\boldsymbol{\mu}$: the (long-term) average number of service-completions per time, assuming an always-full queue.

In the "long-term limit", we expect $\mu = 1/\mathbf{E}[S]$, where S is the service time random variable. This holds for the M/M/1 queue we will study, where the interarrival times are Exponential(λ) and the service times are Exponential(μ).

[1] I know, doesn't Exponential start with 'E'?! 'M' here stands for Markovian, or Memoryless.
[2] By the way, you saw the discretized version of this queuing system before, in Homework 8, Problem 5.

3 Analyzing the M/M/1 queue

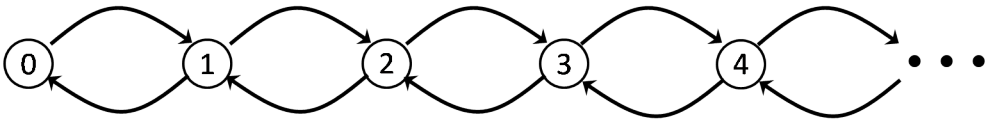
3.1 The M/M/1 queue as a CTMC

The M/M/1 queue is particularly nice because *it can be modeled with a continuous-time Markov Chain*. The state set is $S = \mathbb{N}$, with the states representing the number of jobs in the system.

Note: If there are $u > 0$ jobs in the system, 1 of them is at the server and $u - 1$ of them are waiting in the queue. We say the server is *busy*. If there are $u = 0$ jobs in the system, we say the server is *idle*.

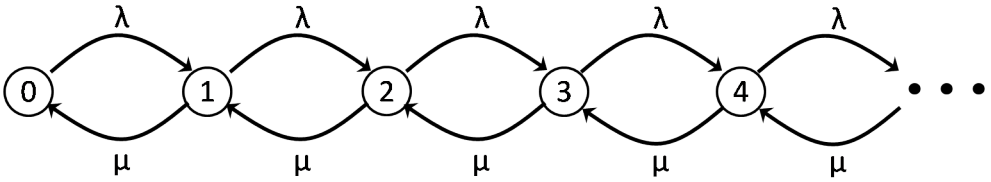
Why is the M/M/1 queue a CTMC? The *memorylessness* of both the interarrival times and the service times mean that the process has the *Markovian Property*. Think about it: Suppose I tell you that at a certain point in time t_0 there are u jobs in the system. By memorylessness, you know that the time till the next arrival is still Exponential(λ), and the time till the current job is finished is still Exponential(μ) (assuming $u > 0$). These facts do not depend on what states you were in prior to u, nor on how long you've been in state u, nor on the point in time t_0.

Thus the chain's state diagram looks like this:



Question: What are the transition rates for the arrows?

Answer: This is easy to see from the 'alarm clock' point of view. Say at a certain point in time there are $u > 0$ jobs in the system. There are two things that can happen next: (i) A new job arrives; this happens when an 'Exponential(λ) alarm clock buzzes'; (ii) The job currently being processed is finished; this happens when an 'Exponential(μ) alarm clock buzzes'. Hence the CTMC for the M/M/1 queue looks like this:



(Note that for state 0 — when the server is idle, the only thing that can happen is that a new job arrives.)

3.2 Solving for the stationary distribution

Let's solve for the stationary distribution, AKA steady-state distribution, of this CTMC. Remember, we should solve the *balance equations*: the rate of transitions leaving a state should equal the rate of transitions entering the state.

Balance equation for state 0: $\lambda\pi[0] = \mu\pi[1]$

Balance equation for state 1: $(\lambda + \mu)\pi[1] = \lambda\pi[0] + \mu\pi[2]$

Balance equation for state 2: $(\lambda+\mu)\pi[2] = \lambda\pi[1] + \mu\pi[3]$
Balance equation for state 3: $(\lambda+\mu)\pi[3] = \lambda\pi[2] + \mu\pi[4]$
Etc.

We start solving from the first of these, trying to write everything in terms of $\pi[0]$.

$$\begin{aligned}
\pi[1] &= \frac{\lambda}{\mu}\pi[0] \\
(\lambda+\mu)\frac{\lambda}{\mu}\pi[0] &= \lambda\,\pi[0] + \mu\pi[2] \quad \Leftrightarrow \quad \frac{\lambda^2}{\mu}\pi[0] = \mu\pi[2] \quad \Leftrightarrow \quad \pi[2] = \left(\frac{\lambda}{\mu}\right)^2\pi[0] \\
(\lambda+\mu)\left(\frac{\lambda}{\mu}\right)^2\pi[0] &= \frac{\lambda^2}{\mu}\pi[0] + \mu\pi[3] \quad \Leftrightarrow \quad \frac{\lambda^3}{\mu^2}\pi[0] = \mu\pi[3] \quad \Leftrightarrow \quad \pi[3] = \left(\frac{\lambda}{\mu}\right)^3\pi[0] \\
(\lambda+\mu)\left(\frac{\lambda}{\mu}\right)^3\pi[0] &= \frac{\lambda^3}{\mu^2}\pi[0] + \mu\pi[4] \quad \Leftrightarrow \quad \frac{\lambda^4}{\mu^3}\pi[0] = \mu\pi[4] \quad \Leftrightarrow \quad \pi[4] = \left(\frac{\lambda}{\mu}\right)^4\pi[0],
\end{aligned}$$

etc. Clearly (prove it by induction, if you like), we have

$$\pi[u] = \left(\frac{\lambda}{\mu}\right)^u \pi[0]$$

for each $u \in \mathbb{N}$. It may look like we're stuck now, but of course, we haven't yet used one equation!

$$\sum_{u\in\mathbb{N}} \pi[u] = 1.$$

Substituting in what we've deduced so far, we get

$$\pi[0]\left(1 + \frac{\lambda}{\mu} + \left(\frac{\lambda}{\mu}\right)^2 + \left(\frac{\lambda}{\mu}\right)^3 + \cdots\right) = 1.$$

The quantity inside the parentheses is an (infinite) geometric series.

Question: When does it converge?

Answer: If and only if $\lambda/\mu < 1$: i.e., $\lambda < \mu$.

Assumption: Let's assume that $\lambda < \mu$. We'll discuss this assumption shortly.

Given this assumption, the infinite series adds up to $\dfrac{1}{1-\frac{\lambda}{\mu}}$. Hence we get

$$\pi[0] \cdot \frac{1}{1-\frac{\lambda}{\mu}} = 1 \quad \Leftrightarrow \quad \pi[0] = 1 - \frac{\lambda}{\mu},$$

and hence

$$\pi[u] = \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^u \qquad \forall u \in \mathbb{N}.$$

For safety, you can go back and check that if you plug in this solution, the balance equations are indeed satisfied. These probabilities $\pi[u]$ are all positive, so we have found the unique stationary/steady-state distribution, by our theorem on CTMCs.

3.3 The assumption $\lambda < \mu$

The assumption we made, $\lambda < \mu$, is actually a pretty natural one. It is saying that the rate of jobs arriving to the server is strictly less than the rate of jobs being serviced. If this is *not* true, then it seems clear that that jobs will just keep building up and building up in the queue; in other words, the server will become *overwhelmed* as time goes on. At a formal level, when $\lambda \geq \mu$, the *balance equations have no solution*, and there is no stationary distribution. We have a "non-recurrent" CTMC here, one of the "irregular" kinds of Markov Chains that we don't bother to deal with.

3.4 Thinking about the steady-state distribution

We now have a nice formula for the steady-state distribution. It's even pleasanter if you introduce the following letter:

$$\rho := \frac{\lambda}{\mu}.$$

Then in the long-term, the fraction of time for which there are u jobs in the system is

$$\pi[u] = (1-\rho)\rho^u. \tag{1}$$

One way to think about ρ is as follows: In the long run,

$$\mathbf{Pr}[0 \text{ jobs in system}] = 1-\rho \quad \Rightarrow \quad \rho = \mathbf{Pr}[> 0 \text{ jobs in system}] = \mathbf{Pr}[\text{server is busy}].$$

Definition 2. *For any server in a queuing system, we write ρ for the long-term (limiting) fraction of time the server is busy.*[3] *This is called the* utilization *of the server. For the M/M/1 queue, we saw that $\rho = \lambda/\mu$.*

The equation (1) should also look sort of familiar. Remember, the stationary probabilities $\pi[\cdot]$ are really a PMF for a discrete random variable, representing the state under a steady-state distribution.

Question: What is this the PMF of?

Answer: Not *quite* Geometric$(1-\rho)$. Instead, if N is the steady-state distribution on states, then $N \sim (\text{Geometric}(1-\rho) - 1)$; i.e., N is one less than a Geometric random variable with parameter $1-\rho$.

Using this fact, it is easy to deduce the following:

Theorem 3. *Let L denote the expected number of jobs in the system, in the steady-state. Then*

$$L = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}.$$

Proof. We have

$$L = \sum_{u \in \mathbb{N}} \pi[u]u.$$

We could compute this using our formula for π. Or we could simply note that

$$L = \mathbf{E}[N] = \mathbf{E}[\text{Geometric}(1-\rho) - 1] = \mathbf{E}[\text{Geometric}(1-\rho)] - 1 = \frac{1}{1-\rho} - 1 = \frac{\rho}{1-\rho},$$

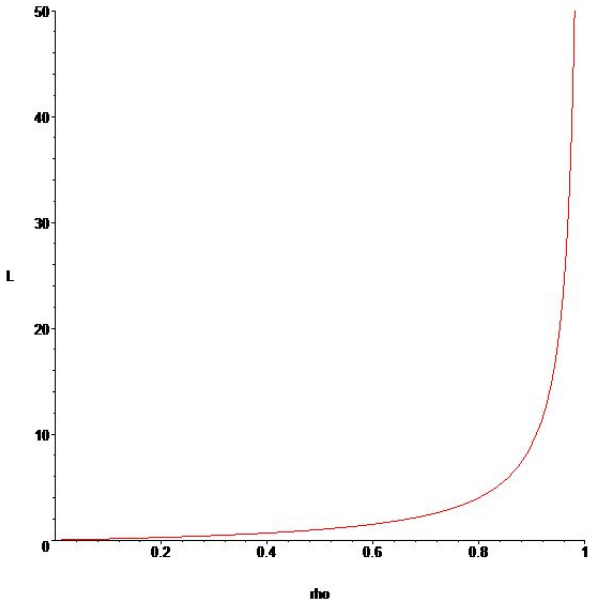
[3] Provided this limit exists.

as claimed. And

$$\frac{\rho}{1-\rho} = \frac{\lambda/\mu}{1-\lambda/\mu} = \frac{\lambda}{\mu-\lambda}.$$

□

The plot of L vs. ρ is:



E.g., if the utilization is .01 — i.e., the server is busy 1% of the time — then the average number of jobs in the system (in the long run) is $\frac{.01}{.99} \approx .01$. On the other hand, if the utilization is .95 — i.e., the server is busy 95% of the time — then the average number of jobs in the system is $\frac{.95}{.05} = 19$.

Exercise 4. *Show that the* variance *of the number of jobs in the system, in the steady state, is* $\frac{\rho}{(1-\rho)^2}$.

3.5 Towards response time

Let's try a slightly harder problem.

Definition 5. *The* response time *for a job is the total amount of time it spends in the system — the time spent queuing plus the time spent being processed.*

Suppose the system is in steady-state and a new job arrives. Let's define the random variable

$$T = \text{response time for the newly arrived job,}$$

and

$$W = \mathbf{E}[T].$$

How can we compute W?

The natural thing to do is to *condition on the number of jobs already in the system when the new job arrives.*

Assuming the system is in the steady-state, let's define the events

$$L_k = \text{“there are } k \text{ jobs in the system when the new job arrives”}.$$

By the law of total expectation, we have

$$W = \mathbf{E}[T] = \sum_{k=0}^{\infty} \mathbf{E}[T \mid L_k]\mathbf{Pr}[L_k]. \tag{2}$$

Now $\mathbf{E}[T \mid L_k]$ is not hard to compute. If a new job arrives and there are k jobs in the system, then the new job will be completed after $k+1$ services (the other k jobs', and its own). Each service time is Exponential(μ), which has mean μ, and so by linearity of expectation,

$$\mathbf{E}[T \mid L_k] = \frac{k+1}{\mu}. \tag{3}$$

Next, we need to compute $\mathbf{Pr}[L_k]$.

3.6 PASTA

We want to compute $\mathbf{Pr}[L_k]$, the probability that a newly arriving job into the steady-state system encounters k jobs.

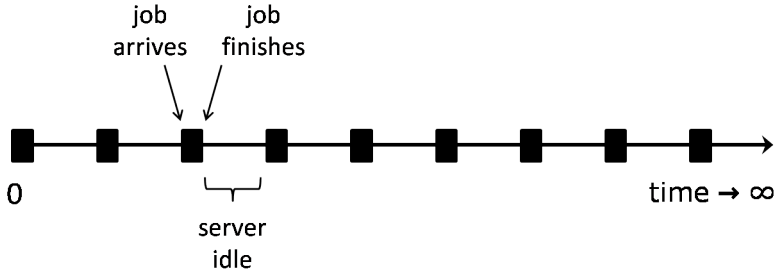
Question: Isn't this just $\pi[k]$?

Answer: Yes! Remember, if the system is in the steady-state (stationary) distribution, then it remains in that distribution *for all time*. In particular, whenever a job arrives, it is in the steady-state distribution.

This feature is called **PASTA** — **P**oisson **A**rrivals **S**ee **T**ime **A**verages.[4] There is a bit of a subtlety going on with it. The key point is the first two words, *Poisson Arrivals*.

PASTA Theorem: *In* any queuing system *where the arrivals are a Poisson Process, the long-term probability that an arriving job sees k jobs in the system equals the long-term probability of there being k jobs in the system.*

If you still think this looks tautological, think again. Consider a queuing system where the interarrival times are *always* 4 seconds and the processing times are *always* 1 second. (Such a queue is termed “D/D/1”, with D standing for Deterministic.)



[4]This goofy name brought to you by Ronald Wolff of Berkeley.

Clearly, the long-term fraction of the time the system has k jobs is $1/4$ for $k = 1$ and $3/4$ for $k = 0$. But in this system, whenever a job arrives it sees 0 jobs in the system with probability 1!

As mentioned, the key to PASTA is *Poisson Arrivals*; more specifically, the fact that the number of arrivals in a certain unit of time is independent of the number of jobs in the system. For example, PASTA holds in an M/D/1 queue, where the arrivals are a Poisson Process and the process times are all deterministically 1 second.

Exercise: Prove this.

3.7 Response time and summary of performance metrics

Finally, we can plug equation (3) and $\mathbf{Pr}[L_k] = \pi[k]$ into equation (2) to get

$$\begin{aligned} W \quad = \quad \mathbf{E}[T] \quad &= \quad \sum_{k=0}^{\infty} \frac{k+1}{\mu} \pi[k] \\ &= \quad \frac{1}{\mu} \left(\sum_{k=0}^{\infty} k\pi[k] + \sum_{k=0}^{\infty} k\pi[k] \right) \\ &= \quad \frac{1}{\mu} (L+1) . \end{aligned}$$

Using the formula in Theorem 3 twice, this implies

$$W \quad = \quad \frac{1}{\mu} \left(\frac{\lambda}{\mu - \lambda} + 1 \right) \quad = \quad \frac{1}{\mu - \lambda} \quad = \quad \frac{L}{\lambda}.$$

We summarize:

Theorem 6. *In the M/M/1 queuing system with arrival rate λ and service rate μ, the utilization is*

$$\rho = \frac{\lambda}{\mu},$$

the long-term average number of jobs in the system is

$$L = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda},$$

and the long-term average response time for an arriving job is

$$W = \frac{1}{\mu - \lambda}.$$

In particular,

$$L = \lambda W. \tag{4}$$

3.8 Little's Law

Equation (4), $L = \lambda W$, is a very famous one. It is called *Little's Law*. It holds not just for the M/M/1 queuing system, but for essentially *every* "queuing system" imaginable. Here is the more general theorem:

Theorem 7. *(Little's Law.) Suppose jobs arrive into a "system" at times $0 \le t_1 \le t_2 \le t_3 \le \cdots$, and suppose the ith job spends T_i time in the system before departing. Define*

$$\begin{aligned}
N(T) &= \text{number of arrivals in time range } [0,T], \\
\lambda(T) &= \frac{1}{T}N(T) \quad = \quad \text{arrival rate in time range } [0,T], \\
X(t) &= \text{number of jobs in system at instant } t, \\
L(T) &= \frac{1}{T}\int_0^T X(t)\,dt \quad = \quad \text{avg. \# jobs in the system in time range } [0,T], \\
W(n) &= \frac{1}{n}\sum_{i=1}^{n} T_i \quad = \quad \text{avg. waiting time of the first } n \text{ jobs.}
\end{aligned}$$

Finally, assume that the following limits exist and are finite:

$$L = \lim_{T\to\infty} L(T), \quad \lambda = \lim_{T\to\infty} \lambda(T), \quad W = \lim_{n\to\infty} W(n).$$

Then

$$L = \lambda W.$$

The proof of this theorem is not too hard.

Little's Law should, of course, be called Little's *Theorem*, but I guess the pull of alliteration was too tempting. It was proved by John Little in 1961, when he was an assistant professor of Operations Research over at our neighbor, Case Western University.[5] He had been teaching a course on queuing theory for four years. In his fourth year, he showed that $L = \lambda W$ holds for M/M/1 queues, and also for a few other queues, and remarked in class that it seemed to hold in a lot of situations. After that class, a bunch of students were talking with him, and one asked, "Do you think it would be hard to prove $L = \lambda W$ in general?" Little said, "Hmm, I guess it shouldn't be too hard." He proved it that summer while on a beach vacation. He also almost immediately quit working on operations research. He is now much more famous for his research on marketing, and currently works at the business school of MIT.[6] Here's a little picture:

[5] Actually, it was called Case Institute of Technology then, and CMU was called Carnegie Institute of Technology.
[6] Massachusetts Institute of Technology.

15-359: Probability and Computing
Fall 2009

Lecture 22: Intro to Gaussian random variables

In this lecture we will learn about 'Gaussian', or 'normal' random variables. They are usually considered the most important class of random variables in all of probability and statistics, due to the "Central Limit Theorem", which roughly says that summing up many independent random variables of *any* type gives you a Gaussian distribution in the limit.

1 The limit of Binomials

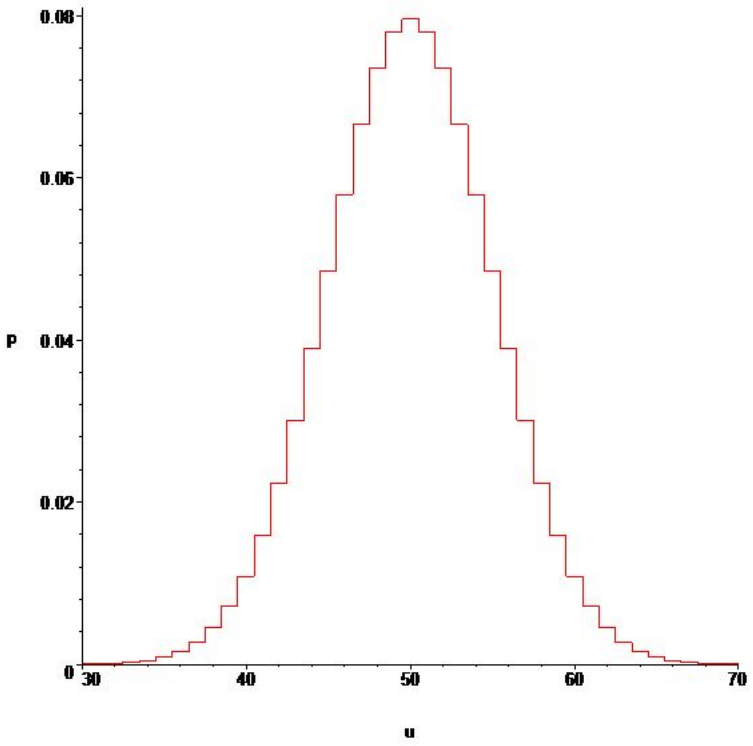
In previous lectures we saw how the Uniform random variable could be seen as the "continuous limit" of RandInt(N), appropriately scaled, and how the Exponential random variable could be seen as the "continuous limit" of Geometric($1/N$), appropriately scaled. We will begin this lecture by investigating the continuous limit of the Binomial random variable.

1.1 Binomials

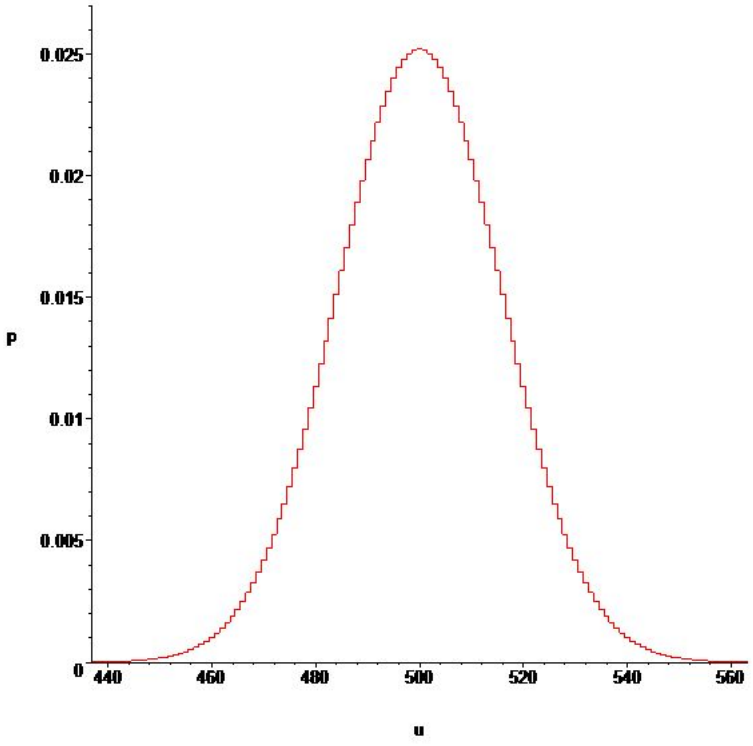
So let's talk about our old friend, $X \sim \text{Binomial}(N, 1/2)$, the number of heads in N fair coin flips. Does it have some kind of limit as $N \to \infty$? Remember, the PMF of X is

$$p_X(u) = \frac{\binom{N}{u}}{2^N}, \quad u = 0, 1, \ldots, N.$$

Here are two plots of $p_X(u)$ I made with Maple, one with $N = 100$ and one with $N = 1000$.



$N = 100$



$N = 1000$

This *does* look like it's "converging", and to the well known "bell curve" at that. On the other hand, you can see that I chose the axes in this plot very carefully: I kept the u axis range pretty near the mean $N/2$, and I cut the p_X axis so that the curve would go to the top.

It looks like what we'll want to do is this:

First: *Translate the random variable so that the mean is always* 0. The way to do this is just to subtract the mean; i.e., define

$$Y = X - N/2.$$

Second: *Scale the random variable so that the variance is* 1. (Equivalently, the standard deviation is 1.) The way to do this is to just divide by the standard deviation. Recall that $\mathbf{Var}[X] = N(1/2)(1/2) = N/4$. Since Y is X minus a constant, we also have $\mathbf{Var}[Y] = N/4$, and hence $\mathbf{stddev}[Y] = \sqrt{N}/2$. So we define

$$Z = \frac{X - N/2}{\sqrt{N}/2}. \tag{1}$$

Remember, we usually think of a Binomial as

$$X = X_1 + X_2 + \cdots + X_N,$$

where the X_i's are independent Bernoulli(1/2) random variables. Hence we can also think

$$Y = Y_1 + \cdots + Y_N, \qquad Y_i = \begin{cases} -1/2 & \text{w.p. } 1/2, \\ +1/2 & \text{w.p. } 1/2, \end{cases}$$

$$Z = Z_1 + \cdots + Z_N, \qquad Z_i = \begin{cases} -1/\sqrt{N} & \text{w.p. } 1/2, \\ +1/\sqrt{N} & \text{w.p. } 1/2. \end{cases}$$

In particular, note that the Z_i's are independent, have mean 0, and have variance $1/N$. Hence the random variable Z has mean 0 and variance 1.

So as $N \to \infty$, it seems like Z might be a good candidate for the "continuous limit" of the Binomial distribution. The associated real-valued random variable would have mean 0 and variance 1. The range of the discrete version is

$$\left\{\frac{-N}{\sqrt{N}}, \frac{-N+2}{\sqrt{N}}, \frac{-N+4}{\sqrt{N}}, \cdots, \frac{+N}{\sqrt{N}}\right\} = \left\{-\sqrt{N}, -\sqrt{N}+\frac{2}{\sqrt{N}}, -\sqrt{N}+\frac{4}{\sqrt{N}}, \cdots, \sqrt{N}\right\}. \quad (2)$$

Since $\sqrt{N} \to \infty$ as $N \to \infty$, it seems like the continuous r.v. Z will have range $(-\infty, \infty)$. But to really understand it, we'll need to work out what Z's PDF (or CDF) should be...

1.2 The limit of Binomials

Let's write φ (the Greek version of f) for the PDF of our hypothesized-to-exist continuous version of the random variable Z. Given $t \in \mathbb{R}$, we want

$$\varphi(t)dt \approx \mathbf{Pr}[t \leq Z < t + dt].$$

Since the discrete random variable Z has range with jumps of size $2/\sqrt{N}$ (see (2)), let's try taking "$dt = 2/\sqrt{N}$". So we want

$$\varphi(t)(2/\sqrt{N}) \approx \mathbf{Pr}[t \leq Z < t + 2/\sqrt{N}].$$

Recall the definition of Z from (1). We get

$$\begin{aligned}
\varphi(t)(2/\sqrt{N}) &\approx \mathbf{Pr}\left[t \leq \frac{X - N/2}{\sqrt{N}/2} < t + \frac{2}{\sqrt{N}}\right] \\
&= \mathbf{Pr}\left[t(\sqrt{N}/2) \leq X - N/2 < t(\sqrt{N}/2) + 1\right] \\
&= \mathbf{Pr}[N/2 + t(\sqrt{N}/2) \leq X < N/2 + t(\sqrt{N}/2) + 1] \\
&= \mathbf{Pr}\left[X = \lceil N/2 + t(\sqrt{N}/2)\rceil\right],
\end{aligned}$$

where the last step used the fact that X takes on integer values. Let's assume that $N/2 + t(\sqrt{N}/2)$ *is* in fact some integer u:

$$u = N/2 + t(\sqrt{N}/2). \quad (3)$$

So

$$\varphi(t) \approx (\sqrt{N}/2) \cdot \mathbf{Pr}[\text{Binomial}(N, 1/2) = u]. \quad (4)$$

1.3 Stirling's Formula

So far so good. As an example, we might imagine the case $t = 0$, which makes $u = N/2$. (Assume N is even.) Hence

$$\varphi(0) \approx (\sqrt{N}/2) \cdot \mathbf{Pr}[\text{Binomial}(N, 1/2) = N/2].$$

On Homework 5 Problem 5, we managed to show that the probability of exactly half heads in N coin flips was equal to $\Theta(1/\sqrt{N})$. (We didn't manage to nail down the constant factors here though.) So we get

$$\varphi(0) \approx (\sqrt{N}/2) \cdot \Theta(1/\sqrt{N}) = \Theta(1).$$

Hmm. Not very informative, but it *does illustrate that we did the scaling correctly*: the N's canceled themselves out in the limit.

But if we already had some difficulty for $t = 0$, won't things be even harder for general t? We know

$$\varphi(t) \approx (\sqrt{N}/2) \cdot \mathbf{Pr}[\text{Binomial}(N, 1/2) = u] = (\sqrt{N}/2) \cdot \binom{N}{u} \Big/ 2^N = (\sqrt{N}/2) \cdot \frac{N!}{u!(N-u)!2^N}.$$

We could plug in $u = N/2 + t(\sqrt{N}/2)$ here, but it's really may not be clear how the N's will cancel out as $N \to \infty$, leaving a function of just t.

The key is to come up with a good approximation for $u!$. This is done with *Stirling's Formula*.

Theorem 1. *De Moivre's Formula:*

$$u! = \Theta(\sqrt{u}(u/e)^u). \quad \textit{I.e.,} \lim_{u\to\infty} \frac{u!}{\sqrt{u}(u/e)^u} = c, \ \textit{a constant.}$$

Stirling's Formula:[1] $c = \sqrt{2\pi}$.

You will prove De Moivre's Formula on the homework. So let's use it (and not the stronger Stirling's Formula) for now. Thus

$$\begin{aligned} \varphi(t) &\sim (\sqrt{N}/2) \cdot \frac{N!}{u!(N-u)!2^N} \\ &\sim (\sqrt{N}/2) \cdot \frac{c\sqrt{N}(N/e)^N}{c\sqrt{u}(u/e)^u \cdot c\sqrt{N-u}((N-u)/e)^{N-u} \cdot 2^N} \\ &= \frac{1}{c} \cdot \frac{N}{2\sqrt{u(N-u)}} \cdot \frac{(N/2)^N}{u^u(N-u)^{N-u}}. \end{aligned} \tag{5}$$

Recall from (3) that

$$u = N/2 + t(\sqrt{N}/2) = \tfrac{N}{2}(1 + \tfrac{t}{\sqrt{N}}), \quad \text{hence } N - u = \tfrac{N}{2}(1 - \tfrac{t}{\sqrt{N}}).$$

Let's write

$$\epsilon = \frac{t}{\sqrt{N}}, \qquad \text{so } u = \tfrac{N}{2}(1+\epsilon), \quad N - u = \tfrac{N}{2}(1-\epsilon).$$

Let's now heroically analyze the two fractions in (5) involving u. First,

$$\frac{N}{2\sqrt{u(N-u)}} = \frac{N}{2\sqrt{\frac{N}{2}(1+\epsilon) \cdot \frac{N}{2}(1-\epsilon)}} = \frac{1}{\sqrt{1-\epsilon^2}}.$$

[1] It was Abraham de Moivre, a French mathematician, who came up with all this limiting Binomial stuff in 1730–1733. He didn't manage to get a closed form for c; the Scottish mathematician James Stirling worked it out (by a very different method) around the same time. Stirling was an interesting character. By age 25 he had become a well known mathematician, and got a job offer from the university in Venice. So he traveled out there from the UK but for an unknown reason the job offer fell through. He kicked around Italy for a bit and soon thereafter became the chair at the University of Padua. However, it is said that during his 3 years there, he learned certain secrets of glassmaking and then had to flee back to Scotland after assassination attempts by Italian glaziers who didn't want the secrets to get out.

But $\epsilon = \frac{t}{\sqrt{N}} \to 0$ as $N \to \infty$ (for any fixed t). Thus $\frac{1}{\sqrt{1-\epsilon^2}} \to 1$. Great! The first fraction involving u drops out in the limit! Next up,

$$\frac{(N/2)^N}{u^u(N-u)^{N-u}} = \frac{\left(\frac{N}{2}\right)^N}{\left(\frac{N}{2}(1+\epsilon)\right)^{(N/2)(1+\epsilon)} \cdot \left(\frac{N}{2}(1-\epsilon)\right)^{(N/2)(1-\epsilon)}}.$$

Check it out: the powers of $\frac{N}{2}$ exactly cancel out here, and the above equals

$$\frac{1}{(1+\epsilon)^{(N/2)(1+\epsilon)} \cdot (1-\epsilon)^{(N/2)(1-\epsilon)}} = \left((1+\epsilon)^{1+\epsilon}(1-\epsilon)^{1-\epsilon}\right)^{-N/2} = \exp(J(\epsilon))^{-N/2}, \tag{6}$$

where

$$J(\epsilon) = \ln\left((1+\epsilon)^{1+\epsilon}(1-\epsilon)^{1-\epsilon}\right) = (1+\epsilon)\ln(1+\epsilon) + (1-\epsilon)\ln(1-\epsilon).$$

Using the Taylor series $\ln(1+x) = x - x^2/2 + x^3/3 - x^4/4 + \cdots$, we get

$$J(\epsilon) = (1+\epsilon)(\epsilon - \epsilon^2/2 + \epsilon^3/3 - O(\epsilon^4)) + (1-\epsilon)(-\epsilon - \epsilon^2/2 - \epsilon^3/3 - O(\epsilon^4)) = \epsilon^2 \pm O(\epsilon^4) = \frac{t^2}{N} \pm O(\tfrac{t^4}{N^2}),$$

where we used the fact that $\epsilon = t/\sqrt{N}$. Finally, substituting this into (6) we get

$$\exp(J(\epsilon))^{-N/2} = \exp\left(\frac{t^2}{N} \pm O(\tfrac{t^4}{N^2})\right)^{-N/2} = \exp\left(-\frac{t^2}{2} \pm O(\tfrac{t^4}{2N})\right) \to \exp\left(-\frac{t^2}{2}\right),$$

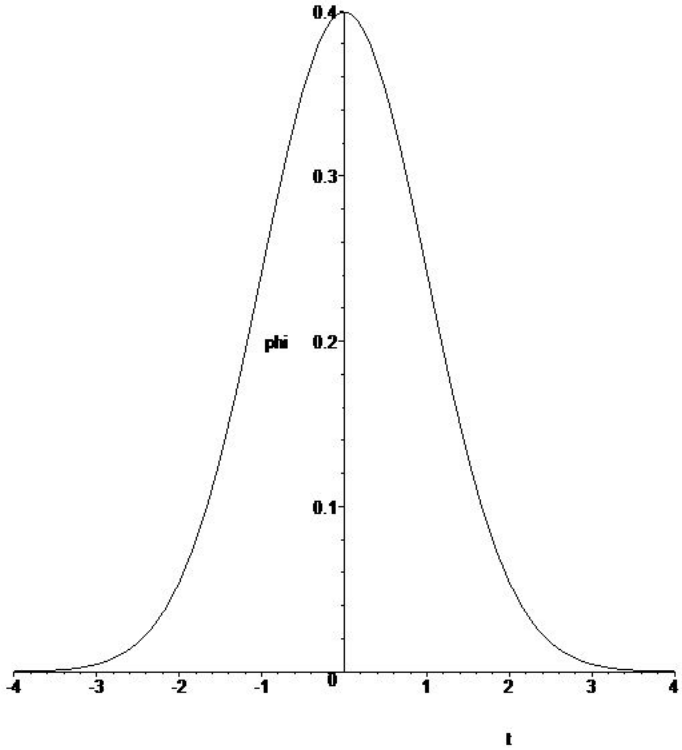
as $N \to \infty$ (for any fixed t).

Finally, plugging all this into (5), we have concluded that as $N \to \infty$,

$$\varphi(t) \to \tfrac{1}{c}\, e^{-t^2/2}.$$

And if you remember Stirling's Formula, $c = \sqrt{2\pi}$. This all suggests that the continuous limit of the discrete random variable Z should have PDF

$$\varphi(t) = \tfrac{1}{\sqrt{2\pi}}\, e^{-t^2/2}.$$



A plot of $\varphi(t)$ vs. t.

This is the PDF of the *Gaussian distribution*, which we will spend some time studying. As mentioned at the beginning of the lecture, the reason it is important is that it's not just the limiting distribution when you add up a bunch of independent Bernoulli(1/2)'s — it's also the limiting distribution when you add up a bunch of independent random variables (with finitely many values), and most continuous distributions! We will see this when we study the "Central Limit Theorem".[2]

2 The Gaussian distribution

2.1 The Gaussian PDF

As usual with continuous random variables, we introduce them by fiat:

Definition 2. *If Z is the random variable with PDF*

$$f_Z(t) = \varphi(t) = \tfrac{1}{\sqrt{2\pi}}\, e^{-t^2/2},$$

then we call Z a standard Gaussian *(or* normal*) random variable. We write $Z \sim N(0,1)$.*

Remark: 'Gaussian' = 'Normal'; the two terms are used interchangeably. The 'N' in $N(0,1)$ stands for 'Normal', and the 0 and 1 are the mean and variance of Z, respectively.

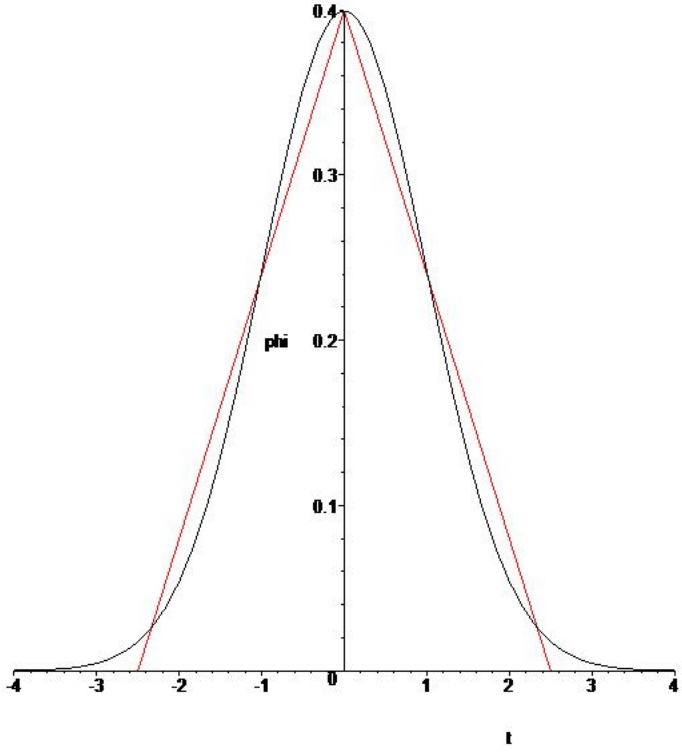
[2] By the way, do you recognize this expression a bit already from the Chernoff Bound?...

Remark: The Gaussian is my favorite kind of continuous random variable! I know it looks painful because of the crazy formula for $\varphi(t)$, but it's really not so bad. I hope it'll become your favorite too!

In any case, in order for this definition to make sense, we need the following theorem:

Theorem 3. $\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt = 1.$

This probably doesn't seem obvious to you. Lord Kelvin reputedly once said "a mathematician is one to whom *that* is as obvious as that twice two makes four".[3] Man, what a pompous jerk! It's not at all obvious! I think the following figure at least makes Theorem 3 plausible:



The area under the curve looks about the same as the area of the triangle that has base 2×2.5 and height .4 — and *that* area is 1.

Anyway, since we just did some long calculations, we'll defer the proof of Theorem 3 to the beginning of the next lecture.

2.2 The Gaussian CDF

You might be asking, why are we deferring the proof of Theorem 3 till later? Why not just integrate as usual? You know, find the antiderivative of $e^{-t^2/2}$, subtract the limiting values, etc. Well, as a matter of fact, it's known that $e^{-t^2/2}$ does *not have an antiderivative* that can be expressed with elementary functions like exp, ln, sin, cos, polynomials, etc. So you simply can't evaluate it this way. Nevertheless, we'll still be able to deduce Theorem 3 by a somewhat unexpected alternate

[3]Strictly speaking, he was referring to the fact that $\int_{-\infty}^{\infty} e^{-t^2}\, dt = \sqrt{\pi}$, but this is obviously the same fact, after a change of variables ;)

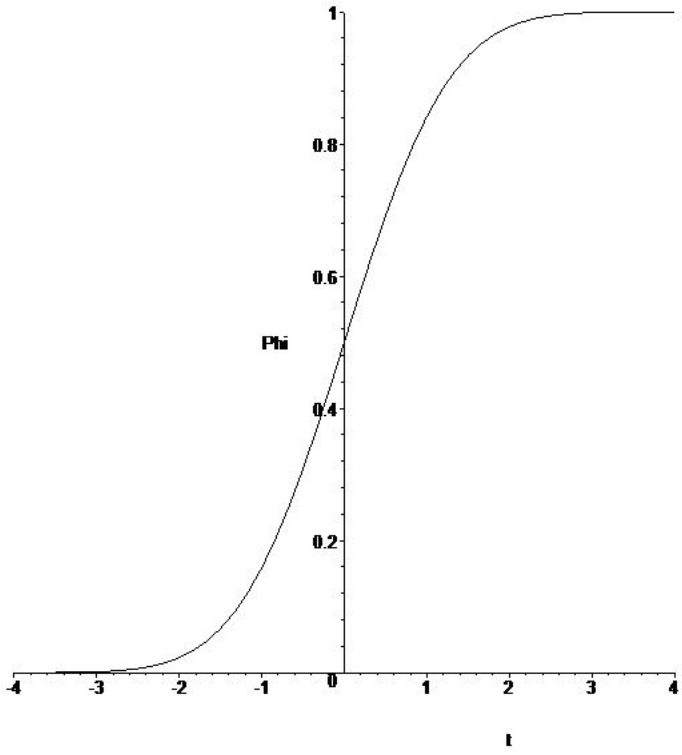
method.

The fact that $\varphi(t)$ has no easy-to-write-down antiderivative means that we can't do much more than define the Gaussian CDF by saying, "it is what it is".

Definition 4. *If $Z \sim N(0,1)$ is a standard Gaussian, its CDF $F_Z(t)$ is usually written written $\Phi(t)$. We have*

$$\Phi(t) = \mathbf{Pr}[Z \leq t] = \int_{-\infty}^{t} \varphi(u)du,$$

but there is no other closed form for this quantity.



Rather than being disappointed by this, you should just strive to get used to it. I mean, you got used to the function $\sin(t)$, right? It's some weird function of t, you know what its plot looks like, you know its derivative, you know you can compute it by plugging numbers into your calculator or your computer. Same thing goes for $\Phi(t)$.

Super-annoyingly, instead of there being a "Φ" button on your calculator, for some crazy reason the engineers decided to have an "erf" button. I'm not sure who invented the function $\mathrm{erf}(t)$, but you should think of it as being defined by the following equation/formula,

$$\Phi(t) = \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{t}{\sqrt{2}}\right). \tag{7}$$

So if you want to compute $\Phi(t)$ for some number t, it's typically fastest to use the above formula along with your calculator/computer's "erf" button. E.g., if I want to know $\Phi(-.3)$, I'll go into Maple or `www.wolframalpha.com` and type

```
1/2 + 1/2 * erf(-.3/sqrt(2))
```

and I'll get back .382. This means that for $Z \sim N(0,1)$ we have

$$\Phi(-.3) = \mathbf{Pr}[Z \leq -.3] \approx .382.$$

Similarly, if I want to know

$$\mathbf{Pr}[-1 \leq Z \leq 2],$$

that's

$$\mathbf{Pr}[Z \leq 2] - \mathbf{Pr}[Z \leq -1] = \Phi(2) - \Phi(-1),$$

which one can work out equals .81859 or so, using (7) and Wolfram Alpha.

Believe it or not, even today teachers in probability classes worldwide still hand out sheets of paper with a table of values for Φ on it, and reading this table is considered an important skill to learn. But that's like handing out sheets of paper with log tables, or sin tables on it! You all know how to use calculators and computers: if you want to know $\Phi(t)$, use formula (7)!

One more comment on integration: Although we told you that there is no closed form for the antiderivative of $\varphi(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}$, there *are* closed forms for very closely related expressions. For example, you can check that

$$\int t \cdot \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt = -\frac{1}{\sqrt{2\pi}} e^{-t^2/2}.$$

(Just differentiate the expression on the right, or use integration by parts on the left.) In fact, I would say that for *a great many* integrals of the form $\int h(t)\varphi(t)dt$, you can compute them by a combination of: a) integration by parts, and b) using Theorem 3 as a black box. In other words, just because $\int \varphi(t)dt$ can't be done, don't think you're off the hook for computing $\int h(t)\varphi(t)dt$ precisely!

2.3 Mean and variance

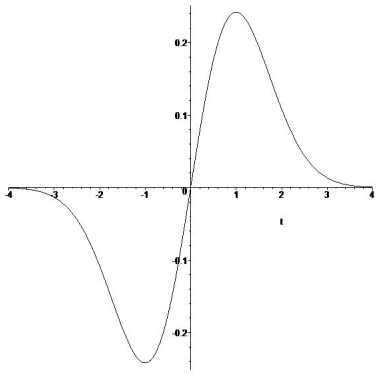
As usual, whenever we have a new kind of random variable, we like to calculate its mean and variance. Since we derived $Z \sim N(0,1)$ as the limit of some discrete random variable which had mean 0 and variance 1, it seems clear that Z will have mean 0 and variance 1. This is true.

Theorem 5. *If $Z \sim N(0,1)$ is a standard Gaussian, then $\mathbf{E}[Z] = 0$ and $\mathbf{Var}[Z] = 1$.*

Proof. We can compute the mean without any calculations: just symmetry. Notice that $\varphi(-t)$ is the same as $\varphi(t)$, because of the presence of the t^2 in the formula. Thus Z is a *symmetric* random variable; i.e., $-Z$ has the same distribution as Z, the standard Gaussian distribution. It follows that the mean of Z must be 0 (because, e.g., $\mathbf{E}[Z] = \mathbf{E}[-Z] = -\mathbf{E}[Z]$). If you wanted to be more formal, you could compute

$$\mathbf{E}[Z] = \int_{-\infty}^{\infty} t \cdot \tfrac{1}{\sqrt{2\pi}} e^{-t^2/2} dt,$$

but again, it's obvious that this is 0 from the picture:



$t \cdot \frac{1}{\sqrt{2\pi}} e^{-t^2/2}$ vs. t

As for showing the variance is 1, since the mean is 0 it suffices to show that $\mathbf{E}[Z^2] = 1$. It's not very hard to verify this using integration by parts. We will see another way at the beginning of the next lecture. □

15-359: Probability and Computing
Fall 2009

Lecture 23: The Central Limit Theorem

In this lecture we will see the famous *Central Limit Theorem*. This is often described as the "crown jewel of Probability Theory". I mean, it's right there in the name: the **Central** Limit Theorem.[1] It will be the last probability *theory* topic in the course. Future lectures will be about computer science applications (although, we will also learn about a very unexpected connection between the Central Limit Theorem and the subject of Computer Science at the end of the lecture).

1 Recap of the standard Gaussian distribution

Remember that the standard normal (or Gaussian) random variable $Z \sim N(0,1)$ has the PDF

$$\varphi(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}.$$

We derived this as the limit of Binomial$(N, 1/2)$ distributions as $N \to \infty$, suitably translated and scaled so that all the discrete distributions had mean 0 and variance 1. De Moivre's Formula,

$$u! = \Theta(\sqrt{u} \cdot (u/e)^u),$$

told us that $\varphi(t)$ had to be of the form

$$\varphi(t) = \frac{1}{c} e^{-t^2/2},$$

where c is the constant hidden in the $\Theta(1)$ in De Moivre's Formula. Stirling's Formula implied that $c = \sqrt{2\pi}$. What we'll do now is prove that $\varphi(t)$ really is a proper PDF:

Theorem 1.

$$\int_{-\infty}^{\infty} e^{-t^2/2} dt = \sqrt{2\pi}.$$

Incidentally, one of the "textbook" ways to prove Stirling's Formula is via the following two steps: a) prove De Moivre's Formula (which you will do on the homework); b) prove Theorem 1. This actually implies that the constant c *has to be* $\sqrt{2\pi}$!

1.1 Proving Theorem 1

As you remember, there is no closed form for $\int e^{-t^2/2} dt$, so the proof of Theorem 1 has to be a little bit sneaky. Here is the idea:

[1] Yep, that's why George Pólya named it that in 1920, a name that stuck.

Idea: To understand a single Gaussian random variable, it's often useful to study *two independent* Gaussians.

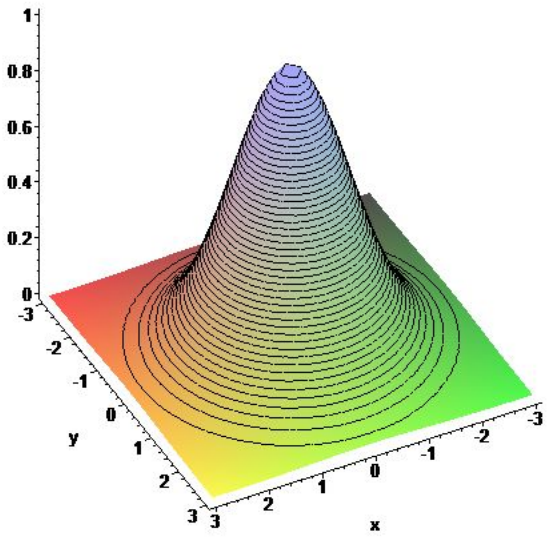
To prove Theorem 1, we'll actually prove the equivalent statement

$$\left(\int_{-\infty}^{\infty} e^{-x^2/2}\,dx\right)^2 = 2\pi.$$

First,

$$\left(\int_{-\infty}^{\infty} e^{-x^2/2}\,dx\right)^2 = \left(\int_{-\infty}^{\infty} e^{-x^2/2}\,dx\right)\left(\int_{-\infty}^{\infty} e^{-y^2/2}\,dy\right)$$
$$= \iint_{\mathbb{R}^2} e^{-x^2/2}e^{-y^2/2}\,dx\,dy = \iint_{\mathbb{R}^2} e^{-(x^2+y^2)/2}\,dx\,dy.$$

Here is the 3-d plot of the integrand:



The key point is:

The function being integrated is rotationally symmetric. Why? Because its value at (x, y) is

$$e^{-r^2/2}, \qquad \text{where } r = \sqrt{x^2 + y^2}.$$

I.e., the function's value at (x, y) only depends on the *distance of* (x, y) *from the origin.*

Whenever you are doing a 2-d integral with such a function you should always think one thing: *polar coordinates*. I.e., you should[2] make the change of variables

$$r = \sqrt{x^2 + y^2}, \qquad \theta = \text{angle of the vector } (x, y) \text{ from the } x\text{-axis.}$$

As I hope you remember from multivariable calculus, with this change of variables, $dx\,dy = r\,dr\,d\theta$. Hence

$$\int_{r=0}^{\infty} \int_{\theta=0}^{2\pi} e^{-r^2/2}\, r\, d\theta\, dr.$$

The integrand now doesn't even depend on θ. Since $\int_0^{2\pi} d\theta = 2\pi$, the above equals

$$2\pi \int_{r=0}^{\infty} r e^{-r^2/2}\, dr.$$

But this is an integral we can do! The antiderivative of $re^{-r^2/2}$ is $-e^{-r^2/2}$, which you can check by differentiating. Hence the above is

$$2\pi(-e^{-r^2/2})\Big|_0^{\infty} = 2\pi(-0 - (-1)) = 2\pi,$$

completing the proof.

1.2 The relationship to Exponentials

What we just exploited is the fact that if you have two independent Gaussians, X and Y, then their joint PDF is

$$f_{XY}(x, y) = \varphi(x)\varphi(y) = \frac{1}{2\pi} e^{-(x^2+y^2)/2}.$$

Let's solve the following problem:

Problem: What is $\mathbf{Pr}[X^2 + Y^2 > t]$?

Solution: To solve this, we integrate the joint PDF over the event in question, $A = \{(x, y) : x^2 + y^2 > t\} = \{(x, y) : \sqrt{x^2 + y^2} > \sqrt{t}\}$ — i.e., everything outside the circle of radius $\sqrt{t}$ centered at the origin.

$$\mathbf{Pr}[X^2 + Y^2 > t] = \iint\limits_{\sqrt{x^2+y^2}>\sqrt{t}} \frac{1}{2\pi} e^{-(x^2+y^2)/2}\, dx\, dy.$$

Changing to polar coordinates again, this equals

$$\iint\limits_{r>\sqrt{t}} \frac{1}{2\pi} e^{-r^2/2} r\, d\theta\, dr = \iint\limits_{r>\sqrt{t}} r e^{-r^2/2}\, dr = (-e^{-r^2/2})\Big|_{\sqrt{t}}^{\infty} = 0 + e^{-\sqrt{t}^2/2} = e^{-t/2}.$$

So we have shown that if $Z = X^2 + Y^2$, then $\mathbf{Pr}[Z > t] = e^{-t/2}$.

Question: What does this mean about the continuous random variable Z?

[2] Exercise: if you don't like polar coordinates, show that this integral can be computed with the substitution $u = x$, $v = y/x$.

Answer: Its distribution is Exponential(1/2)!

We have just shown:

Theorem 2. *If X and Y are independent standard Gaussians, and $Z = X^2 + Y^2$, then $Z \sim$* Exponential(1/2)*.*

We can use this to give a slick proof that a Gaussian has variance 1:

$$\mathbf{E}[X^2] = \frac{1}{2}\left(\mathbf{E}[X^2] + \mathbf{E}[X^2]\right) = \frac{1}{2}\left(\mathbf{E}[X^2] + \mathbf{E}[Y^2]\right) = \frac{1}{2}\left(\mathbf{E}[X^2 + Y^2]\right) = \frac{1}{2}\mathbf{E}[Z] = \frac{1}{2} \cdot 2 = 1.$$

2 The general Gaussian distribution

So far we have only talked about the "standard" normal distribution, denoted $N(0,1)$. Now we'll define the general normal distribution.

Remember that in deriving the Gaussian distribution, we started with $X \sim$ Binomial$(n, 1/2)$. We first *translated* it to get the mean-0 variable Y, and then we *scaled* it to get the mean-0 distribution Z:

$$Z = \frac{X - n/2}{\sqrt{n}/2} \quad \Leftrightarrow \quad X = n/2 + (\sqrt{n}/2)Z.$$

This suggests that if we take

$$Z \sim N(0,1),$$

then

$$X = n/2 + (\sqrt{n}/2)Z$$

should be a good approximation to the Binomial$(n, 1/2)$ distribution, when n is large. More generally, this suggests studying random variables of the form

$$W = a + bZ,$$

where $a, b \in \mathbb{R}$. Of course,

$$\mathbf{E}[W] = a + b\mathbf{E}[Z] = a, \qquad \mathbf{Var}[W] = \mathbf{Var}[bZ] = b^2\mathbf{Var}[Z] = b^2.$$

Because of this, it's a bit more natural to write μ instead of a and σ^2 instead of b^2. We will also call random variables like W "Gaussian". Just as we have a one-parameter family of distributions Exponential(λ), we have a two-parameter family of Gaussian distributions.

Definition 3. *Let W be a random variable of the form $a + bZ$, where $Z \sim N(0,1)$ is a standard Gaussian, $a, b \in \mathbb{R}$, and $b \neq 0$. Then we call W a* Gaussian random variable with mean a and variance b^2*. We also write $W \sim N(a, b^2)$. It is more usual to write $\mu = a$ and $\sigma^2 = b^2$, so $N(\mu, \sigma^2)$ denotes a Gaussian with mean μ and variance σ^2.*

Let's figure out the PDF and CDF of $W = \mu + \sigma Z$.

Theorem 4. *The PDF of $W \sim N(\mu, \sigma^2)$ is*

$$f_W(t) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right),$$

and the CDF of W is

$$F_W(t) = \Phi\left(\frac{t-\mu}{\sigma}\right).$$

Proof. As usual, we start with the CDF:

$$\begin{aligned} F_W(t) = \mathbf{Pr}[W \leq t] &= \mathbf{Pr}[\mu + \sigma Z \leq t] \\ &= \mathbf{Pr}\left[Z \leq \frac{t-\mu}{\sigma}\right] \qquad \text{(we used } \sigma > 0 \text{ here)} \\ &= \Phi\left(\frac{t-\mu}{\sigma}\right) \quad = \quad \int_{-\infty}^{(t-\mu)/\sigma} \varphi(u)du. \end{aligned}$$

Having gotten the CDF, we can continue by making a change of variables:

$$v = \mu + \sigma u \quad \Leftrightarrow \quad u = (v-\mu)/\sigma.$$

Then u ranging from $-\infty$ to $(t-\mu)/\sigma$ is the same as v ranging from $-\infty$ to t, and we also have

$$du = \frac{1}{\sigma}dv.$$

Hence:

$$\int_{-\infty}^{(t-\mu)/\sigma} \varphi(u)du \quad = \quad \int_{-\infty}^{t} \frac{1}{\sigma}\varphi\left(\frac{v-\mu}{\sigma}\right)dv.$$

Thus by the Fundamental Theorem of Calculus, we have

$$f_Z(t) = \frac{d}{dt}F_W(t) = \frac{1}{\sigma}\varphi\left(\frac{v-\mu}{\sigma}\right) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\left(\frac{t-\mu}{\sigma}\right)^2 \Big/ 2\right),$$

as claimed. □

To be perfectly honest, I do not always knock myself out remembering these formulas. When working with a random variable $W \sim N(\mu, \sigma^2)$, I almost always immediately "**standardize it**", which means write it as $\mu + \sigma Z$, where $Z \sim N(0,1)$. For example:

Question: Suppose $W \sim N(\mu, \sigma^2)$. What is the probability that $W \leq 3$?

Answer: Writing $W = \mu + \sigma Z$, where $Z \sim N(0,1)$, we have

$$W \leq 3 \quad \Leftrightarrow \quad \mu + \sigma Z \leq 3 \quad \Leftrightarrow \quad \sigma Z \leq 3 - \mu \quad \Leftrightarrow \quad \leq Z \leq \frac{3-\mu}{\sigma}.$$

Hence

$$\mathbf{Pr}[W \leq 3] = \Phi((3-\mu)/\sigma),$$

which one can work out with a calculator and the "erf" formula for any given μ and σ.

3 The sum of independent Gaussians is Gaussian

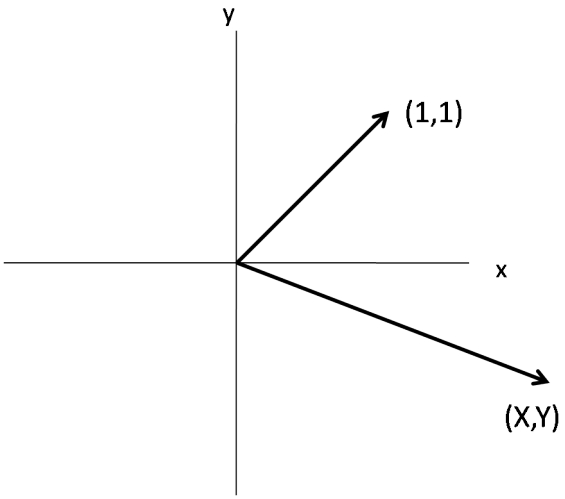
We originally derived the Gaussian as the limit of translated/scaled Binomial$(n, 1/2)$ random variables, with n large. Now suppose X and Y are independent Binomial$(n, 1/2)$ (and hence are approximately $N(n/2, n/4)$). Of course, $X + Y$ has distribution Binomial$(2n, 1/2)$: it's just the number of heads when you flip n fair coins, then flip n more fair coins. This suggests that the sum of two independent Gaussians should also be Gaussian. This is true! Let's do the simplest case:

Theorem 5. *Let X and Y be independent standard Gaussians. Then $Z = X + Y$ has distribution $N(0, 2)$.*

Proof. We will take advantage of the *rotational symmetry* of two independent Gaussians. Think of (X, Y) as a random point/vector in $\mathbb{R}^2$. Then

$$Z = (X, Y) \bullet (1, 1),$$

where $\bullet$ denotes the usual dot product.



Notice that the vector $(1, 1)$ is at an angle of 45° from the x-axis. Let's write Rot_{45} for the operation of rotating a vector by 45° clockwise. For example, $\mathrm{Rot}_{45}(1, 1) = (\sqrt{2}, 0)$.

Now you know that if you rotate two vectors $\vec{a}$ and $\vec{b}$ by the same amount, it doesn't change the dot product. (This is because the dot product depends only on the lengths of the vectors and the angle *between* them: $\vec{a} \bullet \vec{b} = \|\vec{a}\|\|\vec{b}\| \cos \measuredangle(\vec{a}, \vec{b})$.) Therefore,

$$Z = \mathrm{Rot}_{45}(X, Y) \bullet (\sqrt{2}, 0).$$

Let us write (X', Y') for $\mathrm{Rot}_{45}(X, Y)$; here X' and Y' are new continuous random variables.

Now what is the distribution of (X', Y')? The key is to remember a fact from the beginning of this lecture:

The distribution of two independent standard Gaussians is rotationally symmetric. Remember, this is because the joint PDF, $\frac{1}{2\pi} e^{-(x^2+y^2)/2}$, only depends on the *length* (squared) of (x, y), not on its angle. Therefore

$(X', Y') = \mathrm{Rot}_{45}(X, Y)$ is also distributed as two independent standard Gaussians!

Thus

$$Z = (X', Y') \bullet (\sqrt{2}, 0) = \sqrt{2}X' \sim \sqrt{2}N(0, 1) = N(0, 2),$$

as claimed. □

On the homework, you will generalize this as follows:

Theorem 6. *Suppose $Z_1, Z_2, \ldots, Z_m$ are independent, and $Z_i \sim N(\mu_i, \sigma_i^2)$. If $W = Z_1 + Z_2 + \cdots + Z_m$, then*

$$W \sim N(\mu_1 + \cdots + \mu_m, \sigma_1^2 + \cdots + \sigma_m^2).$$

By the way, the fact that the random variable W has mean $\mu_1 + \cdots + \mu_m$ follows from linearity of expectation, and the fact that it has variance $\sigma_1^2 + \cdots + \sigma_m^2$ follows from the fact that variance is additive for *independent* random variables. The point of the theorem is that W is a *Gaussian* with these parameters.

4 The Central Limit Theorem

4.1 Statement of the CLT

We came to Gaussians by noting that as $n \to \infty$, the sum of independent Bernoulli(1/2) random variables (i.e., a Binomial) has a distribution which tends to a Gaussian. We also just saw that if you add up a bunch of independent *Gaussian* random variables, you again get a Gaussian. Could it be that the sum of a bunch of *any* independent random variables converges to a Gaussian?

Yes! This is the content of the Central Limit Theorem (CLT). This theorem was first quasi-proved by Laplace in 1812. (You will give roughly the same quasi-proof on the homework!) It was first proved properly by our old friend Poisson, in 1824–1829.

Setting: Let X be a random variable. Let $X_1, X_2, X_3, \ldots$ be independent "copies" of X (rv's with the same distribution).

Assumption: We need to assume that the random variables have *finite mean and variance*: $\mu := \mathbf{E}[X] < \infty$, $\sigma^2 := \mathbf{Var}[X] < \infty$.

We are interested in the *sums*, when you add up n copies of the random variable:

$$S_n = X_1 + X_2 + \cdots + X_n.$$

In order to investigate convergence, we should "standardize" S_n; i.e., translate/scale it so it has mean 0 and variance 1. We have $\mathbf{E}[S_n] = n\mu$ and $\mathbf{Var}[S_n] = n\sigma^2$, hence $\mathbf{stddev}[S_n] = \sqrt{n}\sigma$. We therefore define

$$Z_n = \frac{S_n - n\mu}{\sqrt{n}\sigma},$$

so that $\mathbf{E}[Z_n] = 0$, $\mathbf{Var}[Z_n] = 1$. The CLT says that Z_n tends to a standard Gaussian random variable, in the following sense:

Theorem 7. *(Central Limit Theorem.) Let F_{Z_n} be the CDF of Z_n. Then*

$$\lim_{n\to\infty} F_{Z_n}(t) = \Phi(t) \qquad \textit{for every } t \in \mathbb{R}.$$

I.e.,

$$\lim_{n\to\infty} \mathbf{Pr}[Z_n \le t] = \Phi(t) = \mathbf{Pr}[N(0,1) \le t] \qquad \textit{for every } t \in \mathbb{R}.$$

4.2 Example

Here is a typical use of the CLT:

Problem: Twitter processes 1 million tweets per day.[3] Let's assume for simplicity that the tweets are independent, and each consists of a uniformly random number of characters between 10 and 140. What is the (approximate) probability that Twitter processes between 74.9 million and 75.1 million characters on a given day?

Solution: Let X be a random variable which is uniform on $\{10, 11, 12, \ldots, 140\}$. Clearly $\mu = \mathbf{E}[X] = 75$. With a little calculation (which we'll skip), you can show that $\sigma^2 = \mathbf{Var}[X] = 1430$. Hence $\sigma = \sqrt{1430} \approx 37.82$. Let us write X_i for the number of characters in the ith tweet, $i = 1 \ldots n$, where $n = 10^6$. Let

$$S_n = X_1 + X_2 + \cdots + X_n$$

be the total number of characters processed on a given day. We want to know

$$p = \mathbf{Pr}[74.9 \times 10^6 \leq S_n \leq 75.1 \times 10^6].$$

We now standardize, defining

$$Z_n = \frac{S_n - n\mu}{\sqrt{n}\sigma}, \quad \Leftrightarrow \quad S_n = n\mu + \sqrt{n}\sigma Z_n = 75 \times 10^6 + 10^3 \times 37.82 \times Z_n.$$

Thus

$$p = \mathbf{Pr}[74.9 \times 10^6 \leq 75 \times 10^6 + 10^3 \times 37.8 \times Z_n \leq 75.1 \times 10^6] = \mathbf{Pr}[-\tfrac{.1}{37.8} \times 10^3 \leq Z_n \leq \tfrac{.1}{37.8} \times 10^3].$$

We have $\frac{.1}{37.8} \times 10^3 = 2.64$. Hence

$$p = \mathbf{Pr}[-2.64 \leq Z_n \leq 2.64] = \mathbf{Pr}[Z_n \leq 2.64] - \mathbf{Pr}[Z_n < -2.64].$$

Since n is so large, by the Central Limit Theorem we may reasonably approximate

$$\mathbf{Pr}[Z_n \leq 2.64] \approx \Phi(2.64), \quad \mathbf{Pr}[Z_n < -2.64] \approx \Phi(-2.64).$$

Finally, we compute $\Phi(2.64) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}(2.64/\sqrt{2}) \approx .996$, using a calculator or computer math program, and also $\Phi(-2.64) \approx .004$. Hence

$$p \approx .996 - .004 = .992.$$

Thus there is roughly a 99.2% chance that the number of characters will be between 74.9 million and 75.1 million.

5 Generalizations and improvements

5.1 Error bounds for The rate of convergence

One nagging question that comes up with the CLT is: "Yes, we get convergence in the limit, but how fast is the convergence? How does the error depend on n?" We sort of skipped over this issue in the previous problem when we said "we can approximate Z_n by $N(0, 1)$". How good is this approximation? This is a very computer sciencey question, by the way — we really care about asymptotic dependence on n, not just "limiting" statements. Many basic probability textbooks will not tell you the answer, but we will see it!

This is the content of the "Berry–Esseen Theorem".[4]

[3] Actually, it's more like 2 million, as of November, 2009.

[4] Proved independently by Andrew Berry and Carl-Gustav Esseen in 1941–42.

Theorem 8. *(Berry–Esseen) In the setting of the CLT, let's assume (without loss of generality by translation/scaling) that* $\mathbf{E}[X] = 0$ *and* $\mathbf{Var}[X] = 1$*. Thus* Z_n *is simply* $S_n/\sqrt{n}$*. Define*

$$\beta_3 = \mathbf{E}[|X|^3].$$

Then for all $t \in \mathbb{R}$*,*

$$\left|F_{Z_n}(t) - \Phi(t)\right| = \left|\mathbf{Pr}\left[\frac{S_n}{\sqrt{n}} \leq t\right] - \Phi(t)\right| \leq O(\beta_3/\sqrt{n}).$$

So basically, the error is $O(1/\sqrt{n})$. For example, suppose X is the "Rademacher random variable", ± 1 with probability $1/2$ each. This indeed has $\mathbf{E}[X] = 0$, $\mathbf{Var}[X] = 1$. It has $\beta_3 = \mathbf{E}[|X|^3] = 1$, because $|X|^3$ is always 1. Thus the Berry-Esseen theorem says that $(\sum_{i=1}^n X_n)/\sqrt{n}$ is very close to being $N(0,1)$: the two CDFs differ by at most $O(1/\sqrt{n})$ at every point t.

If you really care about the constant, you can use Shevtsova's Theorem:[5]

Theorem 9. *(Shevtsova's Theorem) The right-hand side of the Berry–Esseen inequality can be replaced by* $.7056\beta/\sqrt{n}$*.*

5.2 Not all the same distribution

Another question is: Suppose $X_1, X_2, X_3, \ldots$ are independent, but they don't necessarily all have the same distribution. Is it still true that the standardized sum Z_n tends to a Gaussian random variable?

Well, the answer is that it depends. Sometimes the answer is No. For example, if X_1 is a Rademacher (± 1 with probability $1/2$ each) and $X_i \equiv 0$ for every $i > 1$, then of course each sum S_n just equals X_1, and so $Z_n = S_n = X_1$ does not converge to a Gaussian — it's always just a Rademacher!

The problem here is that one random variable (namely, X_1) "dominates" the sum. If you make some assumptions that prevent one random variable from dominating, then the CLT continues to hold. For example:

Lyapunov's Assumption: WOLOG, assume $\mathbf{E}[X_i] = 0$ for each i. Let $\sigma_i^2 = \mathbf{Var}[X_i]$, and define $\beta_i = \mathbf{E}[|X_i|^3]$. Assume

$$\frac{\sum_{i=1}^n \beta_i}{(\sum_{i=1}^n \sigma_i^2)^{3/2}} \to 0 \text{ as } n \to \infty.$$

Theorem 10. *(Lyapunov's CLT.*[6]*) Assuming Lyapunov's Assumption, the CLT's conclusion still holds.*

5.3 Further improvements

The following is a true story. Once upon a time — more precisely, fall of 1933 — there was an undergraduate student at the University of Cambridge. He was a junior at the time. He went to a lecture by a famous astrophysicist, Sir Arthur Eddington, who talked about how in natural

[5] Proven in 2006 by Irina Shevtsova.

[6] Proved by Aleksandr Lyapunov in 1901. Actually, he proved something a bit stronger.

scientific experiments, the measurement errors tended to a Gaussian distribution. Eddington only made some vague mentions of the Central Limit Theorem. To be honest, mathematicians in the UK were just not interested in probability, and hardly knew anything about it. (The Russians were the real leaders in probability back then.) The student was a smart guy, though, and realized this "probability" stuff was really important. So he took it upon himself to investigate what was going on with this CLT.

As I said, though, hardly anyone in the UK knew much about probability, his math professors included. So the student got a couple of textbooks — some in English, some in German — and did self-study. At the time, the CLT itself at least appeared in textbooks. (I'm not sure whether or not Lyapunov's theorem appeared.)

By very early spring of 1934, the junior had proved some great results, all by himself! He proved a version of the CLT which was even *stronger* than Lyapunov's — basically, he showed that a relaxed version of the Lyapunov Assumption was still sufficient for CLT to hold. He also made some progress on showing that his relaxed assumption was *necessary*.

But what to do with this cool research? His professors didn't really know anything about this newfangled probability area. What would you do?

Well, he did what was pretty much the only thing he could do. He wrote a letter to the author of his favorite textbook! This was a Czech mathematician by the name of Emanuel Czuber. He got Czuber's address, in Vienna, out of the textbook. I'm not sure we know what it said, exactly — probably something like "Sehr geehrter Herr Professor Czuber: I proved this cool generalization of the CLT — what do you think?" The only reason we know about it is because we have a copy of a letter the student wrote to his *mother*, in April 1934. After all the "Dear Mom," stuff, he wrote,

> I am sending some research I did last year to Czuber in Vienna, not having found anyone in Cambridge who is interested in it. I am afraid however that he may be dead, as he was writing books in 1891.

!!!! I love the fact that he was like, "uh, yeah, he might be dead." There were no homepages to check back then! And the kicker is, Czuber *was* dead — he'd been dead for 9 years!

Eventually the student corresponded with some people that knew probability, and unfortunately, he was eventually informed that actually, his results were mostly already known. :(The bulk of them had been proved by a Finnish mathematician called Jarl Lindeberg, in 1922. (Yes, news traveled slowly back then. Also, Lindeberg himself had died in '32.) There were some pieces that the student had done that were new, though.

Anyway, once his professors had found out that this student had independently proved some super-important probability results that had originally taken more than 10 years to solve, they were extremely impressed. They basically said, "This is worth a Ph.D. Write it up as a dissertation." So the student did, finishing it up and submitting it in the first semester of his senior year. He actually never got around to publishing the work in a journal, since he figured it was mostly not a new result. But by the end of his senior year, his dissertation *was* accepted, and the student skipped straight to being a post-doc.

But before this dissertation turned out to be accepted, the student still had his final semester as a senior to finish. His first research project obviously had some ups and downs, but he wasn't discouraged. In this final undergrad semester, while waiting to hear about his thesis, he went to another talk at Cambridge, this time by a topologist who described the two Gödel Incompleteness Theorems that had just been proved three years ago. The topologist ended by saying, "There's still one major open problem in the area: decidability. Is there some kind of decision procedure that will always let you tell if a given mathematical theorem is provable or not?".

"Hmm," said the undergraduate, "this decidability stuff sounds interesting. I guess the first step is to formalize what an "algorithm" is, and then maybe think if this Gödel guy's Diagonal Argument can be useful for proving things decidable or undecidable."

Yep. The undergrad was Alan Turing. He had it solved by the end of his senior year.

15-359: Probability and Computing
Fall 2009

Lecture 24: Intro to Computational Learning Theory

1 What is learning?

There are plenty of tasks out there which it seems like computers ought to be able to do — and yet, it's not obvious how to program them to do so. Here are just a few of many examples:

- Face recognition: given a photo of a face, whose face is it? Or even simpler: given a photo, does it contain a face?
- Handwriting recognition. Or even simpler: recognition just for the 10 digits.
- Autonomous driving.
- Detecting spam email.
- Recommending movies to a person based on past preferences.
- Translating text between English and Spanish.

Question: Why "ought" computers be able to do these things?

Answer: Almost all human brains can do them pretty easily. (Except for the last one, but anyone raised bilingually in English and Spanish does it after 5 or 6 years.)

Human brains contain about 10^{11} neurons. Each one is connected to about 10^3 to 10^4 others. So if you think of the brain as a circuit, there are maybe a few hundred trillion wires. That's not so many! Various companies have databases with over a petabyte of storage. Traffic on the Internet is a few hundred TB/second. So why shouldn't we have computers that solve all of these tasks...?

The trouble is it's not obvious how to program a computer "from scratch" to do any of these tasks. We somehow know that there *are* reasonably-sized programs that can do these tasks, we just don't know what they are. Somehow human brains "learn" how to do these tasks, or are "trained" to do them. *Computational Learning Theory* is the research area devoted to designing computer algorithms that learn *rules* or tasks from *examples*.

1.1 Questions, questions

There are several immediate questions that arise when discussing a particular learning task.

- How many "examples" does a learner need before it has a good chance of learning the correct rules?
- Where do these examples come from?
- How should the learner go from the examples to a learned rule or "hypothesis"?
- How can the learner know if its hypothesis is a good one?

To answer these questions scientifically and algorithmically, we need a well-defined mathematical model.

2 A formal model for learning tasks

We will describe a basic model for computational learning. (Many many other variations and models are possible.) As a running example we will use the following learning task:

Running example: Important-Emails. A company providing a web-based email application decides to have a new method of displaying unread emails to the user. It will have a special pane at the top devoted to what it thinks are the "*important*" unread emails (from the user's perspective). It therefore needs a way of automatically classifying (for each user) incoming emails as "important" or not.

2.1 Definitions

In general, a learning (AKA "supervised learning" AKA "classification") task has the following components:

Instances: These are the basic "objects" in the learning problem. In our running example, these are incoming email messages. For face recognition or handwriting recognition, the instances are images. For English-Spanish translation, the instances are (say) sentences. For autonomous driving, the instances would probably be "states of the car+road system".

Features: (AKA attributes.) Instances are abstracted/broken down into a vector of "features". When the instances are images, the features might be the colors of each pixel. When the instances are movies (in the movie recommendation task), the features might be the tuple "(genre, director, rating, length, year, inColor?, etc.)". For English text instances, such as the emails in our running example, it is quite common for the features to be the *presence or absence of each possible word*. This means that we fix a dictionary of all n possible English words, and then we convert

$$\text{text} \quad \Rightarrow \quad \text{binary vector in } \{0,1\}^n,$$

where the ith bit is 1 if and only if the text contains the ith word in the dictionary. This is called the **bag of words** abstraction, since it loses the information about which word was where.[1] In general, abstracting instances down to a good vector of features ("feature extraction") is an important art.

[1] Try searching for a literal interpretation of the "bag of words" on the 8th floor of the Gates Center.

Instance space: This means the set of all possible feature vectors. When all the features are binary-valued (which you can insist on without loss of generality, if you really want), the instance space is $\{0,1\}^n$. In the remainder of this lecture we will assume this is the case. So when we speak of an instance, this corresponds to a binary string in $\{0,1\}^n$.

Label: The "label" of an instance is the "correct answer". In this lecture we will consider the basic case where the labels are Yes or No, i.e., 1 or 0. This is the case for, e.g., face detection, or our problem of determining if an email is important. For English-Spanish translation, the instances might be pairs of sentences, and the label is 0 or 1 depending on whether they are "correct" translations of one another. For handwritten digit recognition, the labels would not be 0 or 1, but would instead be $\{0,1,\ldots,9\}$. In the case of binary labels, it's also common to generalize to the case of *probabilities*. E.g., in a medical diagnosis scenario, instead of labeling a patient as coronary-risk or not-coronary-risk, one might label them with a probability-of-coronary.

Target function: For most learning scenarios, one assumes that there exists a mapping from instances to true labels. This mapping is called the *target function*. Since we are focusing in this lecture on binary features and labels, our target functions are of the form

$$f : \{0,1\}^n \to \{0,1\}.$$

For example, for face detection on black and white images, the target function f maps a given image (thought of as a binary vector of pixel values) into a Yes (1) or No (0) answer. *The target function is unknown to the learner — indeed, it's what the learner is trying to learn.*

We also tend to assume the target function is *relatively simple*. For example, the user him/herself has the ability so say whether an incoming email is important or not relatively quickly; they are using some relatively simple rule in their brain to map each email into a label. We do *not* usually worry about trying to learn, e.g., a totally random function $f : \{0,1\}^n \to \{0,1\}$.

The assumption that a target function exists might not be a good one, especially if the features don't perfectly capture the real instances. For example, in the Important-Emails problem, if we abstract out emails as binary vectors with the bag of words approach, maybe some bags of words could correspond both to an important email and to a non-important one, depending on the actual word structure. We will talk about this issue again later when we discuss *noise*.

Labeled example: This is a pair $\langle x, y\rangle$, where x is an instance and y is a label; in this lecture, $x \in \{0,1\}^n$, $y \in \{0,1\}$. In our running example, the pair is of the form $\langle \text{email}, \text{isImportant?}\rangle$.

Training data: This is a "batch" of "correctly" labeled examples, $\langle x^1, f(x^1)\rangle, \ldots, \langle x^m, f(y^m)\rangle$. This is the data that a learning algorithm is trying to learn from. Here we are assuming the example instances are correctly labeled by the (unknown-to-the-learner) target function. Again, we will discuss this assumption shortly.

2.2 Enter probability

Before we even get to the algorithmic learning task, we need to discuss an important basic question: *where does the training data come from*? This can be divided into two questions: Where do the training instances $x^i \in \{0,1\}^n$ come from? And where do the "correct" labels $y^i = f(x^i)$ come from?

Let's start with the first question. The short answer is "Nature": there is usually a "natural" distribution on instances. This is where probability enters the picture:

Modeling assumption: There is a probability distribution $\mathcal{D}$ on instances from $\{0,1\}^n$. The distribution $\mathcal{D}$ is *unknown* to the learner, but the learner can obtain *independent* draws from it. This is also the distribution that the learner will be *tested* on.

You can think of this as saying there is some randomized code $\mathcal{D}()$ which outputs instances $x \in \{0,1\}^n$. You can think of its return value as a $\{0,1\}^n$-valued random variable X. We assume that the learner is allowed to make independent calls to $\mathcal{D}()$ to get its training instances; i.e., it gets the values of independent copies $X^1, \ldots, X^m$ of the random variable X.

This model is quite reasonable in many real-world cases.[2] For example, in the Important-Email scenario there is the natural distribution of emails that a particular user receives. The email client can observe a succession of emails from this distribution. For a face recognition learning task, one might imagine a learner simply picking a sequence of random images independently from Flickr. The assumption that the distribution is unknown to the learner also makes sense: it does not seem reasonable to assume that a learning algorithm will know, e.g., the probability that a user will receive an email forming a particular bag of words.

2.3 Labeled examples?

Next, let's return to the question of where the learner gets the *labels* $y^i = f(x^i)$ for its training data. Getting correct labels for training data is usually a difficult and expensive "real-world" problem. In many cases, a human will go through the training instances and label them by hand. (E.g., "yep, this image contains a face; nope, this image does not contain a face".) This obviously doesn't scale very well, although things like Prof. von Ahn's ESP Game (now Google Image Labeler), which gets hundreds of thousands of images labeled by humans through an online game, can help. In our scenario of labeling emails as important or not, there are several things the company could try. First, it could try annoying the user somewhat by asking him or her to explicitly label a week's worth of emails as important or not. Second, it could try to guess the answer ex post facto for a bunch of training emails by observing things like: i) whether the user clicked on the email first; ii) whether the user sent a response; iii) whether the user deleted the message; iv) whether the user spent a long time reading the message, etc.

Conclusion: We allow the learner to obtaining training data $\langle x^1, f(x^1)\rangle, \ldots, \langle x^m, f(x^m)\rangle$, with the x^i's being drawn independently from an unknown-to-the-learner probability distribution $\mathcal{D}$. However, because getting good training labels is usually expensive, we strive to design algorithms that use as few training examples as possible.

2.4 Noise

In reality, it is unlikely that in the training data we will perfectly have $y^i = f(x^i)$ for every i; even careful humans will sometimes make mistakes in doing hand-labeling. Furthermore, as we discussed earlier, sometimes it doesn't even make sense to assume that there *is* a target function f mapping each instance to a unique label. Perhaps some small fraction of instances don't really have a "correct answer". A more general and more realistic framework for learning is the following:

Noisy data: We continue to assume that there is a relatively simple target function $f : \{0,1\}^n \to \{0,1\}$. However, we assume that some small fraction η of the training data is *noisy*; i.e., has

[2] The ability to get *independent* samples could be questioned.

$y^i \neq f(x^i)$.

For this lecture, we will stick to the simple case where there is *no noise*: $\eta = 0$ and we assume the training data is of the form $\langle x_1, f(x^1)\rangle, \ldots, \langle x_m, f(x^m)\rangle$ for some target function f.

3 The learning task

The basic task of the learning algorithm:

Task: Given the training data, output a low-error *hypothesis*.

Definition 1. *A* hypothesis *is a map h from the instance space to the label set. In our case, it is a function $h : \{0,1\}^n \to \{0,1\}$.*

This should make sense: e.g., the learner is trying to come up with a rule which classifies each new email as Important or Not.

Note: Notice that we *didn't* say that it is the learner's task to identify f. It only has to come up with a "good" or "low-error" hypothesis.

The notion of "error" is slightly subtle; please pay attention to the following definition:

Definition 2. *In a learning scenario, the* true error *of hypothesis h is*

$$\mathrm{Err}(h) = \mathop{\mathbf{Pr}}_{X \leftarrow \mathcal{D}}[h(X) = f(X)].$$

The empirical error *of hypothesis h is*

$$\widehat{\mathrm{Err}}(h) = \textit{fraction of training examples } \langle x_i, y_i\rangle \text{ for which } h(x_i) = y_i.$$

The goal is an algorithm which produces a hypothesis with *true error* as close to 0 as possible.[3] An obvious strategy is to find a hypothesis with low *empirical error*, and hope that it also has low true error.

Question: What is the simplest and best way to find a hypothesis with low *empirical error*?

Answer: Output the following hypothesis:

$$h(x) = \begin{cases} y^i & \text{if } x = x^i \text{ for some training instance } x^i, \\ 0 & \text{else.} \end{cases}$$

This hypothesis is just a simple lookup table based on the training data, and obviously has empirical error 0.

You might be skeptical, though, as to whether this "trivial" solution will actually have low true error. It doesn't really seem like it's doing "learning". We are led to several questions.

[3] When we have noisy training data, the goal is to find a hypothesis with true error as close to η as possible.

Questions:

1. Given a hypothesis, can an algorithm *check* if it has low true error?
2. Under what circumstances can we expect a hypothesis with low empirical error to also have low true error?
3. Given training data, are there algorithms with fast running times for finding hypotheses with low empirical error?

We will investigate these questions in turn.

4 Hypothesis validation

Suppose a learning algorithm looks at training data and (somehow) comes up with a hypothesis h that looks pretty good. Is there anyway to decide if it's actually good?

Theorem 3. *Suppose after m training examples we produce hypothesis $h : \{0,1\}^n \to \{0,1\}$. Then with $v = \lceil \frac{3}{\epsilon^2} \ln(2/\delta) \rceil$* additional *labeled examples, we can estimate the* true error $\mathrm{Err}(h)$ *to within $\pm\epsilon$, with confidence $1 - \delta$.*

I.e., with not too many more labeled examples, we can (with high probability) estimate how good our proposed hypothesis is. This process of vetting a candidate hypothesis is called *validation*.

We described validation according to the following process: 1. Get training data and learn. 2. Get *more* labeled examples, and validate. But usually one thinks of it as follows: 1. Get training data. 2. Pick some fraction of it (randomly, perhaps) and hold that aside as validation data. 3. Train on the remaining data, obtaining hypothesis h. 4. Validate h using the saved validation data.

Proof. The proof of theorem uses the idea of *sampling* (AKA polling), from way back in Lecture 10. Given the hypothesis h, suppose the algorithm draws fresh labeled examples $\langle X^1, f(X^1)\rangle$, ..., $\langle X^v, f(X^v)\rangle$, where the X^i's are independently drawn from $\mathcal{D}$. For each i, the algorithm also computes $h(X^i)$ and see whether it equals $f(X^i)$. Let

$$E_i = \text{indicator random variable that } h(X^i) = f(X^i).$$

Then the algorithm will estimate the true error of h as

$$\overline{E} := \frac{1}{v} \sum_{i=1}^{v} E_i.$$

The key observation is that the distribution of the random variable E_i is

$$E_i \sim \text{Bernoulli}(\mathrm{Err}(h)).$$

Also, the E_i's are independent, since the X^i's are. Hence it's just like sampling/polling, and we can apply the Sampling Theorem to get the claimed result. (It's like we're polling the instances x to see whether they think $h(x) \neq f(x)$.) □

This is a classic use of the Chernoff Bound in Learning Theory.

5 Occam's Razor

Okay, so if we have a hypothesis, it's not too expensive to check whether or not it's got low true error. But what is the relationship between true error and empirical error? When do we expect a hypothesis with low empirical error to also have low true error? Let's focus on the case of hypotheses with *zero* empirical error:

Definition 4. *A* consistent hypothesis *is one which is correct on* all *training examples; i.e., it has empirical error* $\widehat{\mathrm{Err}}(h) = 0$.

As we saw, it's easy to output a consistent hypothesis: just output a lookup table based on the training data. But it really seems doubtful that this hypothesis will do well on future data ("generalize", is the terminology). On the other hand, suppose I have a large set of training data,

$$m \begin{cases} \langle 1011001010111101 & , \quad 1\rangle \\ \langle 0101101011101011 & , \quad 0\rangle \\ \langle 1101101011101011 & , \quad 1\rangle \\ \langle 0101101011101011 & , \quad 0\rangle \\ \langle 1110001100100011 & , \quad 1\rangle \\ \cdots & \\ \langle 0011101010000101 & , \quad 1\rangle, \end{cases}$$

and I (somehow) notice that

$$h(x) = x_1 \vee x_3$$

is a consistent hypothesis. That seems kind of convincing! If m is 10,000, and every single training example is correctly labeled by the hypothesis h, it seems like it would almost be too much of a coincidence for this to be very wrong.

This leads to the following definitions:

Definition 5. Occam*, refers to William of Ockham, a 14th century Franciscan friar and logician from Surrey, UK.*

Definition 6. Occam's Razor *is a quotation of his, "entia non sunt multiplicanda praeter necessitatem".*

Of course, what Occam meant by this was the following:[4]

Theorem 7. *(Also called* Occam's Razor*.) Suppose A is a learning algorithm which takes m training examples and is guaranteed to output a* simple *hypothesis; say, one that it describes using s bits. Assume*

$$m \geq \frac{1}{\epsilon}\left((\ln 2)s + \ln(1/\delta) + 1\right).$$

Then

$$\mathbf{Pr}[A \text{ outputs a consistent output yet } \mathrm{Err}(h) > \epsilon] < \delta.$$

I.e., if an algorithm finds a consistent, simple hypothesis with zero empirical error on a large enough set of training data, then with high probability its true error is small.

[4]Okay, okay, it actually means "entities must not be multiplied beyond necessity"; i.e., "the simplest explanation is likely to be correct". The "razor" part refers to shaving away unnecessary assumptions.

Example: Suppose that our algorithm always outputs a *monotone disjunction*. This is just a fancy way of saying it outputs an OR of some subset of the n variables, like $x_1 \vee x_3$ from the previous example. Then we may take $s = n$, because it is possible to specify an arbitrary monotone disjunction with n bits (one bit for each variable, to say whether or not it is included in the disjunction). Say we want $\epsilon = 1\%$ and $\delta = 5\%$. Then Occam's Razor's requirement is

$$m \geq \frac{1}{.01}\left((\ln 2)n + \ln(1/.05) + 1\right) \approx 69n + 400.$$

Conclusion: If you have n-bit instances, and $69n + 300$ training examples, and you manage to find a monotone disjunction hypothesis consistent with all of the training data, then 95% of the time that hypothesis has true error at most 1%.

Proof. (of Occam's Razor.) Before we even think about running algorithm A, let's recall that by assumption, its output hypothesis will be describable by at most s bits. Therefore, there are at most

$$2^0 + 2^1 + 2^2 + \cdots + 2^s \leq 2^{s+1} \text{ different hypotheses}$$

that A might ever output. Let's write these potential hypotheses as

$$h_1, h_2, \ldots, h_{2^{s+1}}.$$

Let's call a hypothesis *bad* if its true error exceeds ϵ. Now, some of the potential hypotheses h_i are bad and some are good. To prove the theorem, we will show

$$\mathop{\mathbf{Pr}}_{\text{training data}}[\textit{exists} \text{ a consistent hypothesis } h_i \text{ which is actually bad}] < \delta.$$

Suppose that the bad hypotheses among $h_1, \ldots, h_{2^{s+1}}$ are $b_1, \ldots, b_t$, where $t \leq 2^{s+1}$. (Notice that "badness" does not depend on the training data.) Consider a *fixed* bad hypothesis, b_j. We have

$$\mathbf{Pr}[b_j \text{ consistent with the first training example}] < 1 - \epsilon,$$

by the definition of b_j being a bad hypothesis. Hence

$$\begin{aligned}
& \mathbf{Pr}[b_j \text{ consistent with all } m \text{ training examples}] \\
< \;& (1-\epsilon)^m \\
\leq \;& \exp(-\epsilon)^m \qquad \text{(since } 1 - \epsilon \leq \exp(-\epsilon) \text{ always, cf. the Most Useful Approx Ever)} \\
\leq \;& \exp(-(\ln 2)s - \ln(1/\delta) - 1) \qquad \text{(by assumption on } m) \\
= \;& 2^{-s} \cdot \delta \cdot e^{-1} \\
< \;& 2^{-(s+1)} \cdot \delta.
\end{aligned}$$

We can now apply a Union Bound over all bad hypotheses:

$$\mathbf{Pr}[\text{any bad hypothesis is consistent with all training data}] \leq 2^{s+1} \cdot 2^{-(s+1)} \cdot \delta = \delta,$$

since there were are at most 2^{s+1} bad hypotheses. □

More generally, with the Chernoff + Union Bound Method, you can show that if A is an algorithm which always outputs a hypothesis describable by s bits, and $m \geq O(s/\epsilon^2)\ln(2/\delta)$, then the empirical error A's hypothesis will be within ϵ of the true error, except with probability at most δ. I.e., roughly speaking, *for simple hypotheses, the empirical error is close to the true error.*

6 Algorithmic learning theory

Great — so all we have to do is find a good-looking *simple* hypothesis, and we will have done a great job of learning.

Of course, this assumes there *is* a good-looking simple hypothesis to find. This might not be the case in general. But remember, we typically assume that the *target function itself* is simple. If we additionally make the assumption that there is *no noise*, then that means there will *always* be a simple hypothesis with zero empirical error (and true error).

In this case, are we done? *Not necessarily.* Because algorithmically speaking, it might be computationally hard to *find* a consistent hypothesis.

Let's consider a simple example. Suppose I promise you the target function is a *monotone disjunction*. Your goal is to design a learning algorithm which finds a monotone disjunction consistent with some given training data. (You know there will always exist at least one such monotone disjunction.) If you can do this, then you will have a good learning algorithm, by Occam's Razor. This leads to the following algorithmic task:

Algorithmic Task: Given as input is m pairs $\langle x^i, y^i \rangle$, with $x^i \in \{0,1\}^n$ and $y^i \in \{0,1\}$. You are promised that there exists some function f, an OR of coordinates, such that $y^i = f(x^i)$ for all i. The task is to *find* an OR of coordinates h with this property.

I'll let this be a little puzzle for you:

Exercise: Show this task can be solved in $O(mn)$ time.

As Occam's Razor tells us we only need to take m to be about $\Theta(n/\epsilon)$, we conclude that monotone disjunctions are learnable in roughly $O(n^2/\epsilon)$ time, which is polynomial in both n and $1/\epsilon$.

6.1 Noise

Suppose there is a bit of noise in this problem, though. Then what is the algorithmic task like?

Algorithmic Task: Given as input is m pairs $\langle x^i, y^i \rangle$, with $x^i \in \{0,1\}^n$ and $y^i \in \{0,1\}$. You are promised that there exists some function f, an OR of coordinates, such that $y^i = f(x^i)$ for at least 99% of the i's. The task is to *find* an OR of coordinates h with empirical error as close to 1% as possible.

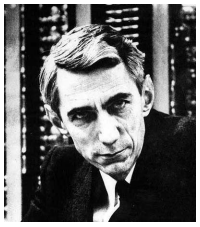
Although it doesn't look much different, it turns out This algorithms problem is way, **way** harder! (You should think about it!) In fact, in 2006, it was shown that to find a hypothesis h with empirical error less than even 49% is NP-hard!

Professor Guruswami has several research papers along these lines, showing that easy-seeming learning-with-noise tasks are actually NP-hard.

15-359: Probability and Computing
Fall 2009

Lecture 25: Elements of Information Theory

Information Theory is a major branch of applied mathematics, studied by electrical engineers, computer scientists, and mathematicians among others. Almost everyone agrees that it was founded by one person alone, and indeed by one research paper alone: Claude Elwood Shannon and his work "A Mathematical Theory of Communication", published in 1948 in the Bell System Technical Journal.



(Shannon was a Michigan-born, MIT-trained mathematician who was working at Bell Labs at the time. Shannon also basically invented the idea of using Boolean logic to analyze and create real-world circuits. He also came up with Problem 2 on the practice midterm when trying to build real circuits using unreliable switches.) The field of information theory has led to innumerable *practical* benefits, from data compression, to hard drive and CD-ROM technology, to deep space communication.

Information theory is fundamentally concerned with two distinct topics:

Compression: This refers to the idea of *removing* redundancy from a source of information, so as to (losslessly) store it using as few bits as possible.

Coding: This refers to the idea of *adding* redundancy to a source of information, so that the original message can be recovered if some of the stored bits are corrupted.

In this lecture we will talk only about compression. We will see that the least number of bits you need to store a large source of random data is essentially equal to its *entropy*.

1 Entropy

1.1 A surprise party

Before getting directly into entropy, let's talk about a related concept: "surprise". Suppose I have some experiment (randomized code) which generates a discrete random variable X. Perhaps

$$X = \begin{cases} a & \text{with probability .5,} \\ b & \text{with probability .25,} \\ c & \text{with probability .125,} \\ d & \text{with probability .125.} \end{cases}$$

Here a, b, c, and d are some distinct real numbers.

Scenario: I am repeatedly and independently instantiating X (i.e., running the code) and telling you things about the result. The question is, for each thing I tell you, how *surprised* are you? Please note that you know the PMF of X, but the only thing you know about my particular instantiations of X are what I tell you.

Question: I instantiate X and tell you it was either a, b, c, or d. How surprised are you?

Answer: Not at all surprised. Zero surprise. You know the PMF of X!

Question: I instantiate X and tell you it was b. How surprised are you?

Answer: I dunno, somewhat?

Question: I instantiate X and tell you it was d. How surprised are you?

Answer: Well, more surprised than with b. It seems like the level of surprise should only depend on the probability of the outcome. So perhaps we could say the amount of surprise was $S(.25)$ for outcome b and $S(.125)$ for outcome d, where $S(.125)$ is greater than $S(.25)$. (And $S(1)$ for the first-mentioned event, $\Omega = \{a, b, c, d\}$.)

Question: Suppose that d had probability .1249999999 rather than .125. How much would that change your surprise level for an outcome of d.

Answer: Hardly at all. I guess you be very slightly more surprised, but basically you'd imagine that $S(.125) \approx S(.1249999999)$.

Question: Suppose I instantiate X once, tell you it was a, then I instantiate X again (independently) and tell you it was c. How surprised are you?

Answer: On one hand, you just experienced an overall event with probability $.5 \cdot .125$, so it seems your surprise level would be $S(.5 \cdot .125)$. On the other hand, you could look at it this way: When you first heard about a, you experienced $S(.5)$ surprise. Then, because you know I'm instantiating X independently, this shouldn't change the fact that when you hear about the second draw being c, you experience $S(.125)$ surprise. I.e., we should have $S(.5 \cdot .125) = S(.5) + S(.125)$.

1.2 Surprise, axiomatically

The preceding discussion suggests that whatever this notion of "surprise" is, it should be a function

$$S : [0, 1] \to [0, \infty)$$

which satisfies the following axioms:

Axiom 1: $S(1) = 0$. (If an event with probability 1 occurs, it is not surprising at all.)

Axiom 2: $S(q) > S(p)$ if $q < p$. (When more unlikely outcomes occur, it is more surprising.)

Axiom 3: $S(p)$ is a continuous function of p. (If an outcome's probability changes by a tiny amount, the corresponding surprise should not change by a big amount.)

Axiom 4: $S(pq) = S(p) + S(q)$. (Surprise is additive for independent outcomes.)

Theorem 1. *If S satisfies the four axioms, then*

$$S(p) = C \log_2(1/p)$$

for some constant $C > 0$.

(You can also check that any $S(p) = C \log_2(1/p)$ satisfies the axioms.)

Let's just sketch the proof here, because the theorem is virtually identical to Homework 11, Problem 1 (proving that the only memoryless continuous random variables are Exponentials). Suppose we write

$$S(1/2) = C.$$

This number C has to be positive, because $S(1/2) > S(1) = 0$ by Axioms 1 and 2. Now by Axiom 4,

$$S((1/2)^2) = 2C, \quad S((1/2)^3) = 3C, \quad \text{etc.}, \quad S((1/2)^m) = mC.$$

Also from Axiom 4,

$$S(\sqrt{1/2}\sqrt{1/2}) = S(\sqrt{1/2}) + S(\sqrt{1/2}) \quad \Rightarrow \quad S(1/2) = 2S(\sqrt{1/2}) \quad \Leftrightarrow \quad S(\sqrt{1/2}) = \frac{1}{2}C,$$

and similarly,

$$S(\sqrt[3]{1/2}) = \frac{1}{3}C, \quad S(\sqrt[4]{1/2}) = \frac{1}{4}C, \quad \text{etc.}, \quad S(\sqrt[n]{1/2}) = \frac{1}{n}C.$$

Combining these two types of deductions leads to

$$S((1/2)^{m/n}) = (m/n)C$$

for all positive integers m, n. Since m/n can be an any positive rational, and thus be made arbitrarily close to a given positive *real*, by Axiom 3 we conclude that indeed

$$S((1/2)^x) = xC \quad \text{for all real } x > 0.$$

But we can express

$$p = (1/2)^x \quad \Leftrightarrow \quad x = \log_{1/2}(p) = \log_2(1/p),$$

and hence indeed

$$S(p) = C \log_2(1/p).$$

1.3 Entropy defined

So any measure of "surprise" must in fact be $S(p) = C \log_2(1/p)$, where $C > 0$ is just a scaling. Let's go ahead and assume we take

$$C = 1.$$

This is a nice choice because it means that if X is a 50-50 coin flip, you experience 1 unit of "surprise" for both heads and tails. Alternatively (and suggestively), you can think of me telling you that I generated a heads as giving you 1 *bit* of surprising new information.

Question: Suppose X is a discrete random variable and I instantiate it once. What is the *expected surprise* the outcome generates for you?

Answer: Well, if p is the PMF of X, then it's just $\mathbf{E}[\log_2(1/p(X))]$.

This quantity is the called the *entropy* of X.

Definition 2. *Let X be a discrete random variable. The* (Shannon) entropy *of X, denoted $H(X)$, equals the nonnegative number*

$$\sum_{x \in range(X)} p_X(x) \log_2(1/p_X(x)).$$

The associated unit of measurement is bits.[1]

Convention: In entropy calculations, we always define $0 \log_2(1/0) = 0$. That way it is even okay to sum over x's not in the range of X (for which $p_X(x) = 0$).[2]

Example: For our original a, b, c, d random variable X, we have

$$\begin{aligned} H(X) &= .5 \log_2(1/.5) + .25 \log_2(1/.25) + .125 \log_2(1/.125) + .125 \log_2(1/.125) \\ &= \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = \frac{14}{8} = 1.75 \text{ bits.} \end{aligned}$$

1.4 Jensen's Inequality

Another way to think about the entropy ("average surprise"): It measures the *uncertainty* in the value of X. Here are two basic facts that suggest this:

Fact 3. *$H(X) = 0$ if and only if X is a* constant *random variable.*

Fact 4. *Suppose X is a random variable with range $\{a_1, a_2, \ldots, a_n\}$; i.e., it can take on n different values. Then the maximum possible value of $H(X)$ is $\log_2 n$.*

Notice that the maximum entropy $\log_2 n$ occurs when we have the *uniform distribution*: i.e., $p_X(a_i) = 1/n$ for all i.

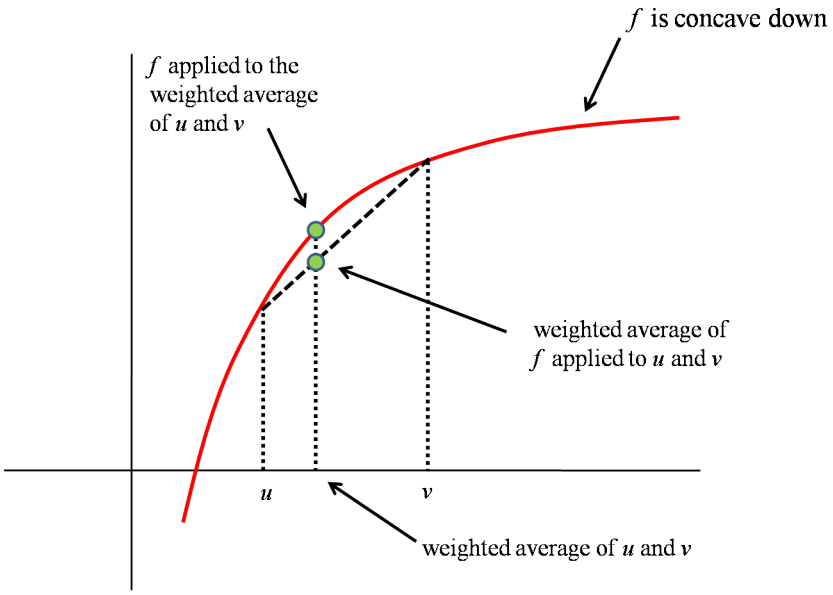
The first fact you should prove for yourself; it's extremely easy. To prove the second fact we need to use a form of *Jensen's Inequality*. Jensen's Inequality says that if f is a *concave down* function, then for any discrete random variable Y,

$$f(\mathbf{E}[Y]) \geq \mathbf{E}[f(Y)].$$

In words, when you take the weighted average of some numbers and then apply f, it's greater than or equal to what you would get if you took the same weighted average of f-applied-to-those-numbers. Here is the "proof by picture" for when you're averaging two numbers:

[1] Amusingly, if you make the definition with ln instead of $\log_2$, you call the unit of measurement *nats*. 1 nat = about 1.44 bits.

[2] This convention never leads to trouble, because indeed $q \log_2(1/q) \to 0$ as $q \to 0$.



In particular, since $\log_2(x)$ is a concave down function, we get:

Theorem 5. *For any discrete random variable Y, $\log_2 \mathbf{E}[Y] \geq \mathbf{E}[\log_2 Y]$.*

We can now prove Fact 4. We'll write p for the PMF of X. Then

$$\begin{aligned} H(X) &= \mathbf{E}[\log_2(1/p(X))] \\ &\leq \log_2 \mathbf{E}[1/p(X)] \qquad \text{(by applying Jensen with the r.v. } 1/p(X)\text{)} \\ &= \log_2 \sum_{i=1}^{n} p(a_i) \cdot (1/p(a_i)) \\ &= \log_2 \sum_{i=1}^{n} 1 \quad = \quad \log_2 n. \end{aligned}$$

2 (Lossless) data compression

Yet another way to think about entropy is in terms of *information content*. There is a sense in which, given one instantiation of the random variable X, you can on average extract $H(X)$ random bits from it. Let's try to understand this through the lens of *data compression*. More specifically, we'll see the reverse perspective: that in order to store the results of many independent draws of X, you need to use $H(X)$ bits per draw, on average.

2.1 On compression

Let's talk about data compression (you know, like in .zip, .tar, or .mp3 files). We will only consider the case of *lossless* compression, meaning we want to always be able to perfectly recover the source data from the compressed version. This problem is also called *source coding*.

Imagine our source data is a sequence of symbols — say, English letters. With the ASCII encoding, English letters are stored using 7 bits each. For example, 'A' is 1000001 and 'Q' is 1010001. On the other hand, with Morse Code, each letter is stored using a sequence of dots and dashes

(except for the 'space' character, which is stored with a space). For example, 'A' is • — whereas 'Q' is — — • —. The idea behind data compression is like that of Morse Code: encode the more common things with short sequences and the less common things with long sequences. Then for a "typical" (probable) sequence of symbols, you will use less storage than if you just used a fixed number of bits per symbol.

We are going to investigate the most efficient way to make such a compression scheme. To do this, we need a probabilistic model.

Assumption: The sequence of symbols we are trying to compress is generated by making *independent* draws from a random variable.

Note that this is *not* a good assumption for English text. In English text, the letters *do* have some probabilistic frequency, but the succession of letters is *not* independent. Things get better though if you consider *pairs* or *triples* of letters to be a single symbol — we'll get back to that. On the other hand, sometimes this *is* a good model. Perhaps you are looking at a specific spot in the genetic code of a protein, and recording the amino acid that is there across thousands of samples. There are 20 different amino acids, represented by a letter, and for a given spot there is some frequency of seeing each. How many bits will you need to record a typical long such sequence? (The goal is to beat the trivial $\lceil \log_2 20 \rceil = 5$ bits per amino acid.[3])

2.2 Prefix-free codes

One elegant way to encode data for compression is with a *prefix-free* code:

Definition 6. *Given a set Σ of symbols, a binary* prefix-free code *(AKA* instantaneous code*) is a way of mapping each symbol to a "codeword" sequence of bits so that* no codeword is a prefix of any other codeword.

For example, if there are 4 symbols A, B, C, D, then mapping them as

$$\begin{aligned} A &\leftrightarrow 11 \\ B &\leftrightarrow 101 \\ C &\leftrightarrow 100 \\ D &\leftrightarrow 00 \end{aligned}$$

is a valid prefix-free code. On the other hand, if we changed C to 110, that would *not* be a valid prefix-free code, because then A's codeword 11 would be a prefix of C's codeword 110.

The beauty of a prefix-free code is that you *don't need any additional markers for the boundary between codewords.* Because of the prefix-free property, you can just concatenate all the encodings together. For example, to compress the sequence

$$BADDAD$$

with the above prefix-code, you would produce

$$1011100001100.$$

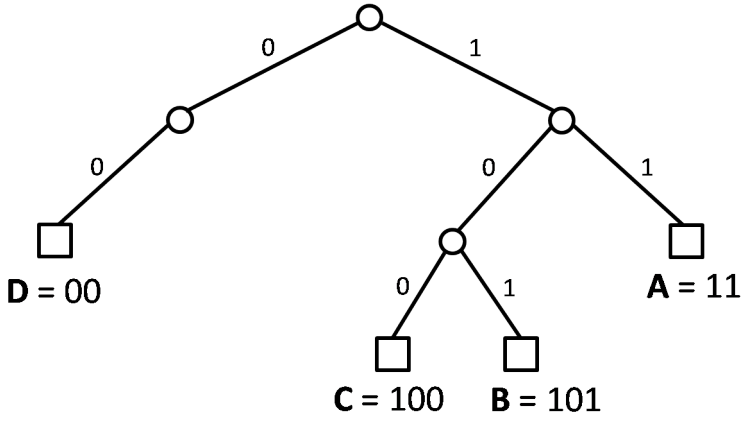
[3]Actually, in real life, each amino acid is stored with 3 nucleotides, AKA 6 bits.

Prefix-codes are also called instantaneous codes because with a left-to-right scan, you can decode each symbol 'instantaneously'. I.e., you will know as soon as you have scanned a whole encoded symbol; there is no need for lookahead.

Again, you don't *have* to compress data using a prefix-free code, but it's one elegant way to do it.

2.3 Prefix-free codes as binary trees

Another way to look at prefix-free codes is as binary trees. Each *leaf* corresponds to a symbol, and the associated codeword is given by the path from the root to that leaf. I trust you will get the idea from the following picture, corresponding to the 4-symbol code we were discussing:



The fact that the code is prefix-free corresponds to the fact that the codewords are at the leaves, and no leaf is a descendant of any other leaf.

2.4 Kraft's inequality

Kraft's Inequality is a very nice property of prefix-free codes.

Theorem 7. *(Kraft's Inequality.) Suppose we have a prefix-free code for n symbols, and the ith symbol is encoded by a string of length ℓ_i. Then*

$$\sum_{i=1}^{n} 2^{-\ell_i} \leq 1. \tag{1}$$

Conversely, if you have any positive integers ℓ_i satisfying (1)*, there is a prefix-free code which has them as its codeword lengths.*

Although there is no probability in this statement, my favorite proof of the first part *uses probability*! Suppose we have a prefix-free code with given codeword lengths. Think of the corresponding binary tree. Suppose we make a random walk down the tree, starting from the root. At each step, we flip a fair coin to decide if we should walk left (0) or right (1). We keep walking randomly until we hit a leaf or until we "fall off the tree" (e.g., if we flipped 0 then 1 in the above tree). In the end, you either hit a leaf or fall off, so clearly

$$\mathbf{Pr}[\text{hit a leaf}] \leq 1.$$

On the other hand,

$$\mathbf{Pr}[\text{hit a leaf}] = \sum_{i=1}^{n} \mathbf{Pr}[\text{hit the leaf for symbol } i],$$

because these are mutually exclusive events. But if the codeword for symbol i has length ℓ_i (i.e., the leaf is at depth ℓ_i), then the probability the random walk reaches it is just $2^{-\ell_i}$. (E.g., the probability of reaching leaf 101 is just 1/8, the probability of flipping 1 then 0 then 1.)

This proves the first statement, that prefix-free codes satisfy (1). To prove the second statement, suppose we are given positive integers $\ell_1, \ldots, \ell_n$ satisfying (1). Here is how you can construct a prefix-free code with these as the code lengths:

- Reorder the symbols so that $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_n$. Let $L = \ell_n$.
- Start with the complete binary tree of depth L.
- Pick any node at depth ℓ_1 and make it into a leaf by deleting all its descendants. Label the leaf with symbol 1.
- Pick any unlabeled node at depth ℓ_2 and make it into a leaf by deleting all its descendants. Label the leaf with symbol 2.
- Pick any unlabeled node at depth ℓ_3 and make it into a leaf by deleting all its descendants. Label the leaf with symbol 3.

etc.

We need to show that inequality (1) implies this algorithm never gets stuck; i.e., there is always an unlabeled node of the appropriate depth to pick. To prove this, let us call all the depth-L nodes "buds". Initially, there are 2^L buds. We begin by picking a depth-ℓ_1 node and making it a leaf. This node has distance $L - \ell_1$ from the buds, so it has $2^{L-\ell_1}$ bud descendants. Thus when we make it a leaf, we kill off a

$$\frac{2^{L-\ell_1}}{2^L} = 2^{-\ell_1}$$

fraction of the initial buds. Similarly, when we next turn a depth-ℓ_2 node into a leaf, we kill off a $2^{-\ell_2}$ fraction of the initial buds. Etc.

Now as long as there is at least one bud left, we are able to continue the process. This is because the buds are at depth L, so if we need to choose an unlabeled node at depth ℓ_i (which is at most L), we can choose an ancestor of any bud left in the tree (or the bud itself if $\ell_i = L$). Finally, the inequality (1) means that we will never kill off a 1-fraction of all the initial buds (except possibly after we have made the last step, placing symbol n).

2.5 The Shannon-Fano code

Now we'll see the *Shannon-Fano code*, which gives a decent prefix-free code for compressing a long sequence of independent instantiations of a random variable X. (The code appeared in Shannon's

famous 1948 paper, but he attributed it to Robert Fano.) Say that

$$X = \begin{cases} \text{symbol } 1 & \text{with probability } p_1, \\ \text{symbol } 2 & \text{with probability } p_2, \\ \dots & \\ \text{symbol } n & \text{with probability } p_n. \end{cases}$$

Let

$$\ell_i = \lceil \log_2(1/p_i) \rceil$$

(the ceiling of the "surprise" $S(p_i)$ of p_i).

Claim 8. *These positive integers satisfy the Kraft Inequality* (1).

Proof.

$$\begin{aligned} \sum_{i=1}^{n} 2^{-\ell_i} &= \sum_{i=1}^{n} 2^{-\lceil \log_2(1/p_i) \rceil} \\ &\leq \sum_{i=1}^{n} 2^{-\log_2(1/p_i)} \\ &= \sum_{i=1}^{n} p_i \quad = \quad 1. \end{aligned}$$

□

Therefore there exists a prefix-free code having the ℓ_i's as its codeword lengths. This is the *Shannon-Fano code*.

The Shannon-Fano code is pretty decent:

Question: What can you say about the expected number of bits used to encode one (random) symbol?

Answer: By definition, it's

$$\sum_{i=1}^{n} p_i \ell_i = \sum_{i=1}^{n} p_i \lceil \log_2(1/p_i) \rceil \leq \sum_{i=1}^{n} p_i \left(\log_2(1/p_i) + 1 \right) = \sum_{i=1}^{n} p_i \log_2(1/p_i) + \sum_{i=1}^{n} p_i = H(X) + 1.$$

That was an upper bound. For a lower bound, we have

$$\sum_{i=1}^{n} p_i \ell_i = \sum_{i=1}^{n} p_i \lceil \log_2(1/p_i) \rceil \geq \sum_{i=1}^{n} p_i \log_2(1/p_i) = H(X).$$

If all the probabilities p_i are of the form $1/2^c$ for $c \in \mathbb{N}$, then the Shannon-Fano code uses exactly $H(X)$ bits per symbol, on average. In the worst case, if the round-off is very bad, it may use up to $H(X) + 1$ bits per symbol. This extra $+1$ may be either reasonable or quite bad, depending on $H(X)$. It's sometimes quoted that the entropy of a single letter of English text is somewhere in the range of 1 to 1.5 bits. So in this case, an extra $+1$ is a lot. On the other hand, for more complicated random variables, the entropy $H(X)$ will be very high, so the $+1$ may be negligible.

2.6 A lower bound for *any* prefix-free compression scheme

Was it just a coincidence that the entropy of X showed up for the Shannon-Fano code? And could we do a lot better than Shannon-Fano? The answer to both questions is no. In fact, the Shannon-Fano code is close to optimal:

Theorem 9. *Suppose we have* any *prefix-free code for X. Then the expected length of a codeword is* at least *the entropy $H(X)$.*

Proof. In this proof, using considerable foresight we will look at the entropy minus the expected length of a codeword:

$$\begin{aligned}
& H(X) - \Big(\text{expected length of a codeword}\Big) \\
= \;& \sum_{i=1}^{n} p_i \log_2(1/p_i) - \sum_{i=1}^{n} p_i \ell_i \\
= \;& \sum_{i=1}^{n} p_i (\log_2(1/p_i) - \ell_i) \\
= \;& \sum_{i=1}^{n} p_i (\log_2(1/p_i) - \log_2(2^{\ell_i})) \\
= \;& \sum_{i=1}^{n} p_i \log_2 \left(\frac{1}{p_i 2^{\ell_i}} \right) \\
= \;& \mathbf{E}\left[\log_2 \left(\frac{1}{p_X 2^{\ell_X}} \right) \right].
\end{aligned}$$

Here we are thinking of X as a random variable that takes value i with probability p_i. We now apply Theorem 5, the corollary of Jensen's Inequality, to conclude:

$$\begin{aligned}
& \mathbf{E}\left[\log_2 \left(\frac{1}{p_X 2^{\ell_X}} \right) \right] \\
\leq \;& \log_2 \mathbf{E}\left[\frac{1}{p_X 2^{\ell_X}} \right] \\
= \;& \log_2 \left(\sum_{i=1}^{n} p_i \cdot \frac{1}{p_i 2^{\ell_i}} \right) \\
= \;& \log_2 \left(\sum_{i=1}^{n} 2^{-\ell_i} \right) \\
\leq \;& \log_2 (1),
\end{aligned}$$

where we used Kraft's Inequality on the codeword lengths of a prefix-free code. But $\log_2(1) = 0$. Hence we have shown

$$H(X) - \Big(\text{expected length of a codeword}\Big) \leq 0,$$

i.e., the expected codeword length must be at least the entropy $H(X)$. □

2.7 Conclusion

In conclusion, we have shown that *any* prefix-free way of encoding a random variable X requires at least $H(X)$ bits on average. Furthermore, the Shannon-Fano code comes close to achieving this, using between $H(X)$ and $H(X) + 1$ bits on average. Is it possible to get rid of this $+1$? Yes! We leave it to you to try to show the following:

Theorem 10. *Suppose we have a long sequence of independent instantiations of X. First, we group the results into blocks of size k. Second, we apply the Shannon-Fano code to the k-blocks.*

Then the expected number of bits per symbol in each block will be between $H(X)$ and $H(X)+1/k$. Hence by making k large, we can approach the compression limit — i.e., the entropy $H(X)$.

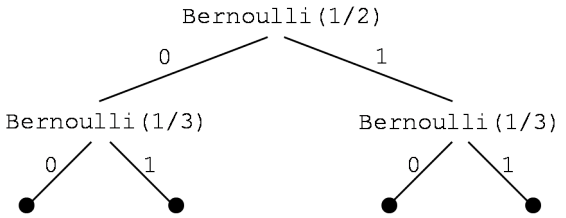
15-359: Probability and Computing
Fall 2009

Lecture 26: Generating Random Variables

1 On generating random variables

We began the class by introducing probability in terms of randomized computer code. Remember this stuff from Lecture 2?

```
flip1 ← Bernoulli(1/2)
flip2 ← Bernoulli(1/3)
```



The first few lectures were about the probability of *events*, but as we got more sophisticated, we came to spend more time talking about *random variables*. At first, our random variables came out of explicit experiments; for example, we got a Geometric(p) random variable as follows:

$X \leftarrow 1$
`while Bernoulli`$(p) = 0$,
    $X \leftarrow X + 1$

But again, as we got more sophisticated, we started defining random variables not via explicit experiments, but rather just by stating their PMFs — e.g.,

$$X \sim \text{Poisson}(\lambda) \quad \Leftrightarrow \quad p_X(u) = \frac{e^{-\lambda}\lambda^u}{u!}$$

— or by their PDFs — e.g.,

$$X \sim N(0,1) \quad \Leftrightarrow \quad f_X(u) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2}.$$

This raises an important question, which we skirted up until now. How do actually *get* such random variables with computer code? This is a very important topic! If you want to do theoretical experiments in queuing theory, you've got to be able to generate Exponential(λ) random variables. If you want to run Michel and David's algorithm for Max-Cut (see Homework 12, #4), you have to be able to generate $N(0,1)$ random variables. If you want to write a simulator for radioactive decay (or 19th-century Prussian cavalry units, see Lecture 11), you'll need to be able to generate Poisson(λ) random variables.

In this lecture we'll investigate 3 basic questions:

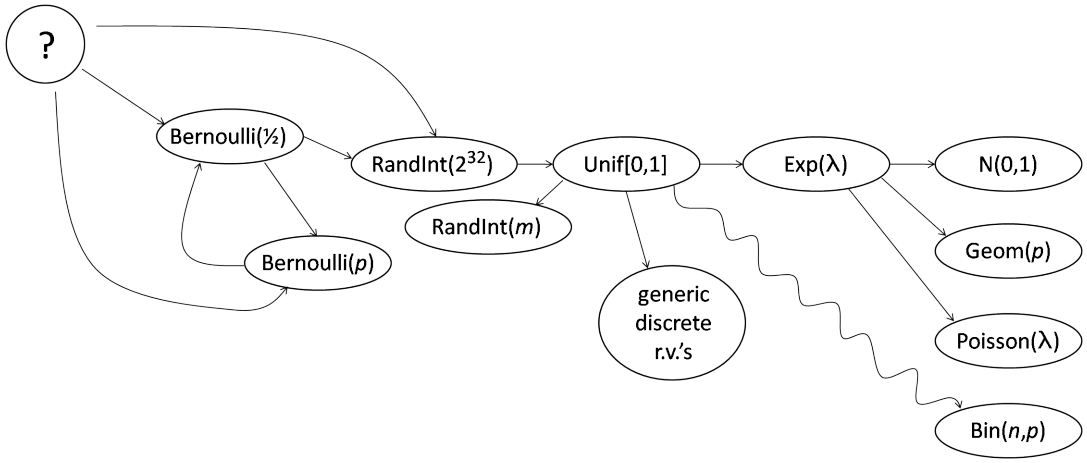
Question 1: Given access to, say, Unif$[0,1]$ or RandInt(m), how can we generate other types of random variables?

Question 2: How can we generate these different types of random variables *efficiently*?

Question 3: How do we even get started? E.g., how can we even get a single random bit?

1.1 The plan

Here is a basic outline for how to generate various kinds of random variables. (Note that we will *not* describe all the possibilities; in some cases, there are better generation methods than the ones we discuss!)



We will describe this diagram in a somewhat random order ;)

2 The starting point: RandInt(2^{32})

In practice, the starting point for (pseudo-)random number generation is some function such as RandInt(2^{32}). Most programming languages (C, java, python, whatever) have some kind of built-in function that returns a uniformly random (supposedly) 32-bit (or 16-bit or 64-bit or whatever) number.[1] At the end of the lecture, we'll come back to this point, but let's just assume we indeed have such a built-in function.

3 From RandInt(2^{32}) to Unif$[0,1]$

We talked about this when we introduced continuous random variables: The notion of generating a uniformly random real between 0 and 1 is an utter mathematical abstraction (albeit, a very useful one!). This is because in the real world, on a finite computer in a finite universe, you can never perfectly represent a real number. But for practical purposes, since 2^{32} (or 2^{64}) is so big, we just do

$X \leftarrow$ RandInt(2^{32})

[1] According to our conventions, this would technically be a draw from RandInt$(2^{32}) - 1$.

$U \leftarrow X/2^{32}$

and figure that this is close enough to $U \sim \text{Unif}[0,1]$. Obviously, you can increase the 32 to a larger number to make it closer.

4 From Unif$[0,1]$ back to RandInt(m)

Sometimes, your programming language only gives you Unif$[0,1]$. As I'm sure you well know, the way then to implement $R \sim \text{RandInt}(m)$ is:

$U \leftarrow \text{Unif}[0,1]$
$R \leftarrow \lceil U \cdot m \rceil$.

5 From Unif$[0,1]$ to Exponential(λ) — the Inverse Transform Method

Once we have the ability to draw $U \sim \text{Unif}[0,1]$, what can we do with it? Let's do some brief exploration. Suppose we do:

$U \leftarrow \text{Unif}[0,1]$
$W \leftarrow 10U^4$.

This gives us a new random variable W, with range $[0,10]$. What distribution does it have? As you'll recall, it's better to work out the CDF of W first. For $0 \le t \le 10$, we have

$$\mathbf{Pr}[W \le t] = \mathbf{Pr}[10U^4 \le t] = \mathbf{Pr}[U^4 \le t/10] = \mathbf{Pr}[U \le (t/10)^{1/4}] = (t/10)^{1/4}.$$

So the CDF of W (on the range $[0,10]$) equals $(t/10)^{1/4}$.

Question: What is the relationship between the transform we made, $10U^4$, and the resulting CDF, $(t/10)^{1/4}$?

Answer: They are inverse functions. You get the CDF by solving $t = 10u^4$ for u.

Question: Okay, so instead of this weird W, suppose we wanted to get $Y \sim$ Exponential(λ). Could we use a similar trick?

Answer: Yes! The CDF of an Exponential(λ) is $1 - e^{-\lambda t}$. We write

$$u = 1 - e^{-\lambda t} \quad \Rightarrow \quad 1 - u = e^{-\lambda t} \quad \Rightarrow \quad \ln(1-u) = -\lambda t \quad \Rightarrow \quad t = -\frac{1}{\lambda}\ln(1-u).$$

Thus to generate a $Y \sim$ Exponential(λ) from a Unif$[0,1]$, we can do

$U \leftarrow \text{Unif}[0,1]$
$Y \leftarrow -\frac{1}{\lambda}\ln(1-U)$.

Just to check that this works... Since $U \in [0,1]$, $\ln(1-U) \in [-\infty, 0]$, hence $-\frac{1}{\lambda}\ln(1-U) \in [0,\infty]$ so that Y has the correct range.[2] And then (making sure to handle the signs carefully!):

$$\begin{aligned}\mathbf{Pr}[Y \le t] = \mathbf{Pr}\left[-\frac{1}{\lambda}\ln(1-U) \le t\right] = \mathbf{Pr}[\ln(1-U) \ge -\lambda t] = \mathbf{Pr}[1-U \ge e^{-\lambda t}] \\ = \mathbf{Pr}[U \le 1 - e^{-\lambda t}] = 1 - e^{-\lambda t},\end{aligned}$$

as desired.

Even simpler: Notice that if $U \sim \text{Unif}[0,1]$, then $1-U$ also has the distribution $\text{Unif}[0,1]$. Hence you can simplify the generation of an Exponential(λ) as follows:

$U \leftarrow \text{Unif}[0,1]$
$Y \leftarrow -\frac{1}{\lambda}\ln U.$

Summary: For a general continuous random variable W, you can simulate it by drawing $u \sim U$ and then outputting $F_W^{-1}(u)$, where F_W^{-1} denotes the inverse function of the W's CDF F_W.

6 From $\text{Unif}[0,1]$ to generic discrete random variables

Actually, the Inverse Transform method works equally well for generic discrete random variables! Suppose we have access to $U \sim \text{Unif}[0,1]$, and we want to simulate a discrete random variable

$$X = \begin{cases} 1 & \text{with probability } p_1, \\ 2 & \text{with probability } p_2, \\ 3 & \text{with probability } p_3, \\ \dots & \\ n & \text{with probability } p_n. \end{cases}$$

(I am just using the values $1, 2, \dots, n$ here for simplicity; they could of course be general values $a_1, \dots, a_n$.)

The simplest way to do it (which you essentially saw on Homework 1 Problem 2) is:

$U \leftarrow \text{Unif}[0,1]$
`if` $U < p_1$ `then` $X \leftarrow 1$
`else if` $U < p_1 + p_2$ `then` $X \leftarrow 2$
`else if` $U < p_1 + p_2 + p_3$ `then` $X \leftarrow 3$
`etc.`

[2] Don't worry, $Y = \infty$ iff $U = 0$, which has probability 0.

This is fact the Inverse Transform method! Think about it. It's because the CDF of X is

$$F_X(t) = \begin{cases} p_1 & \text{if } t < 1, \\ p_1 + p_2 & \text{if } t < 2 \\ p_1 + p_2 + p_3 & \text{if } t < 3 \\ \text{etc.} \end{cases}$$

Downside: A drawback of this method is that it takes $O(n)$ time in the worst-case. The reason the Inverse Transform method was so great for Exponential random variables was that there was a simple and snappy formula for the inverse of the CDF: $-\frac{1}{\lambda}\ln(1-u)$. If you don't have such a formula, it can be a real drag to go cycle through up to all n values in the generic discrete case. (And it's impossible to do in the continuous case!)

Method of Aliases: One thing that somewhat helps is the so-called Method of Aliases, invented by Alastair Walker in 1977. Suppose you have some discrete random variable X, takes on n values, and you plan on sampling from it many times. The Method of Aliases gives a way of constructing a couple of arrays based on X, with the following properties: 1. It takes $O(n)$ time to construct the arrays. 2. But, once you have them, you can sample from X in $O(1)$ time using one draw from Unif$[0,1]$.

It is not too hard to describe the Method of Aliases, but we will skip it due to lack of time.

7 From Exponentials to $N(0,1)$

As we've seen, the Inverse Transform always works, but it is not very useful it it is hard to compute the inverse CDF. In particular, suppose we wanted to generate a Gaussian, $N(0,1)$. Its CDF is

$$\Phi(t) = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-u^2/2}\, du,$$

which already looks very hard to compute, let alone having to compute the inverse! Is there another way?

Idea: Use the relationship between Exponentials and Gaussians!

Remember back in Lecture 23 we saw the following:

Theorem 1. *Let X and Y be independent $N(0,1)$ Gaussians. Then:*

1. *The distribution of $(X,Y) \in \mathbb{R}^2$ is **rotationally symmetric**.*
2. *If $Z = X^2 + Y^2$ is the squared distance of (X,Y) from the origin, then $Z \sim$ Exponential$(1/2)$.*

The trick is to just think of this theorem in reverse!

Corollary 2. *Suppose we pick an angle $\Theta \in [0, 2\pi)$ uniformly at random, and we also pick $Z \sim$ Exponential$(1/2)$. Let $(X,Y) \in \mathbb{R}^2$ be the vector of length $\sqrt{Z}$ and angle Θ to the x-axis. Then X and Y are independent standard Gaussians.*

Hence we get a method for generating two independent Gaussians. (We only wanted one, but hey, why not take two?)

$\Theta \leftarrow 2\pi \cdot \text{Unif}[0,1]$
$Z \leftarrow -2\ln(\text{Unif}[0,1])$ `// so` $Z \sim \text{Exponential}(1/2)$
$X \leftarrow Z\sin(\Theta)$
$Y \leftarrow Z\cos(\Theta)$

Neat, huh? There is a slight variant (called the "polar method") on this little algorithm which doesn't require the use of trigonometric functions. (It still requires computing a logarithm.) This used to be the most popular way of generating Gaussians, but now there is an even better way called the *Ziggurat Method*, invented by George Marsaglia and Wai Wan Tsang in 2000. It is sort of a generalization of the Method of Aliases; after generating a certain lookup table, you can draw from $N(0,1)$ with one draw from $\text{Unif}[0,1]$, a couple of multiplications, an 'if', and a table lookup.

8 From Exponentials to Geometrics

Of course, you can generate $G \sim \text{Geometric}(q)$ from $\text{Unif}[0,1]$ in the obvious way:

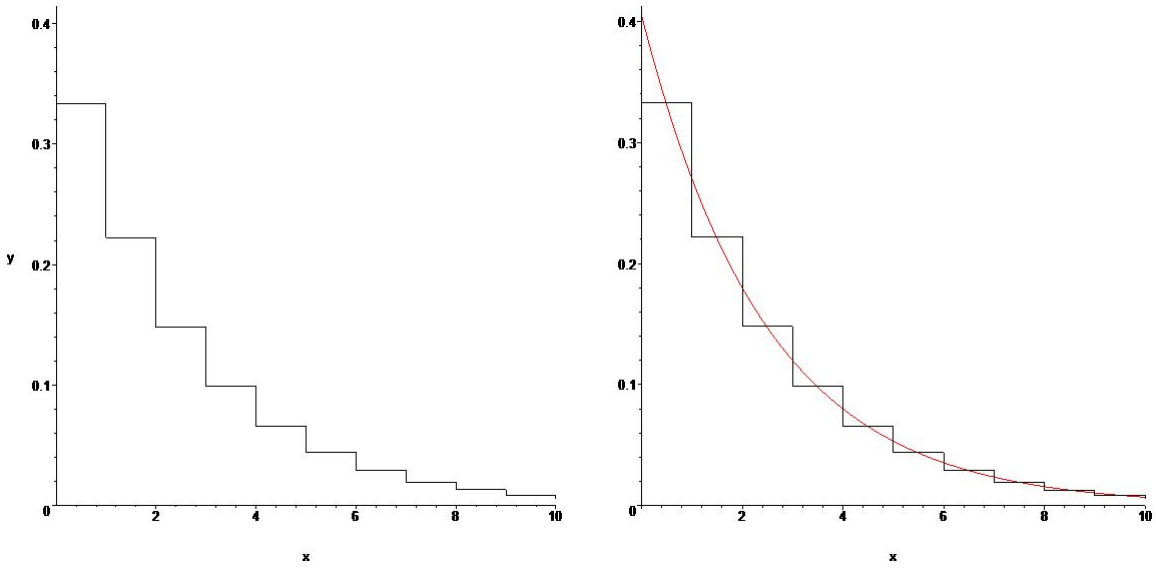
$G \leftarrow 1$
`while` $\text{Unif}[0,1] > q$,
    $G \leftarrow G + 1$

Question: How long does this take, on average?

Answer: The running time is proportional to the final value of G. Hence the expected running time is $O(\mathbf{E}[G]) = O(1/q)$.

This is pretty bad if p is small! Another option is to try the Inverse Transform method — i.e., see if we can find an easy formula for the inverse of the CDF of a Geometric(p). This will work, but there is a slightly slicker way to do it.

On the left here is a histogram-style plot of the PMF of a Geometric(1/3) random variable. In general, of course, the PMF values go down by a factor of $(1-q)$: q, then $q(1-q)$, then $q(1-q)^2$, then $q(1-q)^3$, etc.



On the right is a plot of the PDF of $Y \sim \text{Exponential}(\lambda)$, where $\lambda = -\ln(1-1/3))$. In general, the PDF of $Y \sim \text{Exponential}(\lambda)$ with

$$\lambda = -\ln(1-q)$$

$$p_Y(u) = \lambda e^{-\lambda u} = -\ln(1-q)e^{\ln(1-q)u} = -ln(1-q)(1-q)^u.$$

This function has the exact same sort of decay: as you increase u by $+1$ it goes down by a factor of $(1-q)$.

Question: Could we be so lucky as to just generate $Y \sim \text{Exponential}(\lambda)$ and then set $G = \lceil Y \rceil$?!

Answer: Yes! Because if we do this,

$$\begin{aligned}\mathbf{Pr}[G=k] = \mathbf{Pr}[k-1 < Y \le k] &= F_Y(k) - F_Y(k-1) = (1-e^{-\lambda k}) - (1-e^{-\lambda(k-1)}) \\ &= e^{\ln(1-q)(k-1)} - e^{\ln(1-q)k} = (1-q)^{k-1} - (1-q)^k = q(1-q)^{k-1} = p_G(k),\end{aligned}$$

as needed.
So to generate a Geometric(p), it's just:
$G \leftarrow \lceil \text{Exponential}(-\ln(1-q)) \rceil$.

Actually, if you remember how we generate Exponentials from Uniforms, this is just:
$G \leftarrow \lceil \ln(\text{Unif}[0,1])/\ln(1-q) \rceil$.

Exercise: *Show that the above line of code is basically doing the Inverse Transform method for a Geometric, except with* $1 - \text{Unif}[0,1]$ *in place of* $\text{Unif}[0,1]$.

Exercise: *Suppose* $q = 1/2$. *Give a much easier way for generating a* Geometric$(1/2)$, *using* RandInt(2^{32}) *and bit-tricks.*

9 From Exponentials to Poissons

What about generating $N \sim Poisson(\lambda)$? This was our prototypical random variable for which we didn't even *have* a natural experiment. (Balls and bins with m bins and $n = \lambda m$ balls approximates it, but does not give it exactly.)

The Inverse Transform method for Poissons is really not suitable. For a modest λ like 25, you need to computer quantities like

$$p_N(30) = \frac{e^{-25} \cdot 25^{30}}{30!} \approx .0454.$$

Here you have an insanely huge number, 25^{30} (which is 140 bits long), roughly canceling out with two insanely tiny numbers e^{-25} and $1/30!$. Doing these calculations with enough precision is going to kill you. Is there any easier way?

Idea: Exploit the Poisson Process!

Imagine a Poisson process with rate λ. Here are two things we know about it:

1. Its generated by interarrival times which are Exponential(λ).
2. The number of arrivals between time 0 and time 1 has distribution Poisson(λ).

Thus, we can get a Poisson(λ) as follows:

$N \leftarrow 0$, $t \leftarrow 0$
`while` $t < 1$,
    $t \leftarrow t + \text{Exponential}(\lambda)$, $N \leftarrow N + 1$
$N \leftarrow N - 1$.

Actually, you can make this slicker. Remember that we generate Exponential(λ) as $-\frac{1}{\lambda}\ln(U)$, where $U \sim \text{Unif}[0,1]$. The above code adds up such draws randomly until they are at least 1. But

$$(-\tfrac{1}{\lambda}\ln(U_1)) + (-\tfrac{1}{\lambda}\ln(U_2)) + \cdots + (-\tfrac{1}{\lambda}\ln(U_n)) = -\tfrac{1}{\lambda}\ln(U_1U_2\cdots U_n) \geq 1 \quad \Leftrightarrow \quad U_1U_2\cdots U_n \leq e^{-\lambda}.$$

So we can use the following simpler code instead:

$N \leftarrow 0$, $r \leftarrow 1$
`while` $r > e^{-\lambda}$,
    $r \leftarrow r \cdot \text{Unif}[0,1]$, $N \leftarrow N + 1$
$N \leftarrow N - 1$.

The running time: As with the naive method for generating Geometrics, this algorithm takes time proportional to N. Hence the expected time it takes to generate a Poisson(λ) is $O(\mathbf{E}[\text{Poisson}(\lambda)]) = O(\lambda)$. This is fine if λ is modest, but not very good if λ is large. There exist algorithms that generate Poissons much faster, but they are significantly more complicated.

10 Binomials?

What about Binomials? Of course, we can generate Binomial(n, p) by doing the natural experiment, adding up n independent Bernoulli(p)'s. Again, this takes time $O(n)$, which is not so great. Again, there are algorithms that generate Binomials much faster, from Unif$[0, 1]$'s, but they are much more complicated.

11 Bernoullis

As mentioned, most pseudorandom number generators give blocks of, say, 32 random bits. Of course, it would be good enough if you just had independent accesses to *one* random bit, Bernoulli(1/2); you could then draw from it 32 times. Since even getting *one* random bit is a bit of a physics/philosophy headscratcher, it's interesting to dial down to the question of simulating one Bernoulli random variable by another.

11.1 From Bernoulli(1/2) to Bernoulli(1/3)

On Homework 1 Problem 2 you investigated simulating Bernoulli(1/3) using calls to Bernoulli(1/2). As you showed there, it's impossible to do this precisely. However as you also showed, you can get a simulation that runs in time $O(n)$ and fails only with probability 2^{-n} — this is good enough!

11.2 From Bernoulli(p) to Bernoulli(1/2) — the unknown bias case

What if someone handed us a weighted coin with some *unknown bias* p, and we could flip it independently as many times as we like. Could we somehow use this to generate a Bernoulli(1/2)? One idea might be to flip it a whole bunch of times, estimate p, and then try to do some kind of strategy like the one in Homework 1 Problem 2. There is a simpler way that gives a perfect simulation of Bernoulli(1/2) which is quite surprising if you haven't seen it before:

Von Neumann's Trick: Flip the coin twice. If you get HT, output 0. If you get TH, output 1. If you get TT or HH, try again.

Obviously, no matter what p is, the probability of HT equals the probability of TH (namely, $p(1-p)$). Hence the probability of outputting 1 conditioned on outputting anything is exactly 1/2; i.e., we get a perfect simulation of Bernoulli(1/2).

Downside: This can be somewhat slow if p is close to 0 or 1. The expected number of flips needed to get 1 unbiased bit is

$$2\,\mathbf{E}[\text{Geometric}(2p(1-p))] = \frac{1}{p(1-p)}.$$

So if you are willing to make n flips, you can get

$$\frac{1}{p(1-p)}n$$

bits out on average. It follows from the last lecture that the best we could hope for would be to take n flips and extract

$$H(p)n := (p\log_2(1/p) + (1-p)\log_2(1/(1-p)))n$$

bits on average. In fact, in 1972 Peter Elias showed a scheme that indeed achieves this rate as $n \to \infty$!

For a taste of it, you might like to investigate the following improvement on von Neumann's scheme: group the flips into blocks of 4, and output bits according to the following rule:

HHHH	nothing
HHHT	11
HHTH	01
HTHH	10
THHH	00
HHTT	1
HTHT	11
HTTH	10
THHT	01
THTH	00
TTHH	0
HTTT	10
THTT	00
TTHT	11
TTTH	01
TTTT	nothing

11.3 From Bernoulli(p) to Bernoulli(1/2) — the *known* bias case

The following is also an interesting question: suppose you *know* the bias p of the coin exactly. What scheme for generating, say, a Bernoulli(1/2) random variable uses the fewest average flips? Although Don Knuth and Andy Yao have a pretty good general scheme for this, determining the optimal scheme **still an open research question**. Here is a puzzle along these lines:

Puzzle 1: Suppose I give you independent draws from Bernoulli(1/3). You must produce a single draw from Bernoulli(1/2). Show a scheme for doing this where the expected number of draws from Bernoulli(1/3) is $2\frac{1}{7}$.

Puzzle 2 (Harder): Prove that $2\frac{1}{7}$ is optimal.

12 Pseudorandom bits

We've evaded the question long enough. Finally we have to ask, "How should we get pseudorandom bits? I.e., how should one implement RandInt(32)?"

You can't create something from nothing, so pseudorandom generators all assume that you have access to short short *seed* sequence of random bits.[3] They then take this seed and repeated perform some kind of *deterministic* transformation on it, each application yielding another 32 bits (say). The hope is that these bits "look like" independent uniform draws from RandInt(32). At this point, it is obligatory to repeat a quotation of John von Neumann's:

[3] You can get try getting this from the bits of the system clock, or from a geiger counter, or a lava lamp, or you can buy them off the Internet.

> Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

12.1 Linear congruential generators

An old-school pseudorandom number generation method is called the "linear congruential generator". (Most implementation of C's `rand()` function use these.) Here you let X_0 be the 32-bit seed, and then the generator defines

$$X_i = 1664525X_{i-1} + 1013904223 \bmod 2^{32}.$$

Here, 1664525 and 1013904223 are carefully chosen magic numbers. Linear congruential generators have two upsides:

1. They are very fast to compute.
2. The numbers they generate are indeed roughly uniformly distributed.

Other than that, they're kind of terrible :) There are gross dependencies between the X_i's. For example, if you naively generated a sequence of random bits by taking $X_i \bmod 2$, you'd get the sequence $\ldots, 0, 1, 0, 1, 0, 1, 0, 1, \ldots$. The values are also pretty bad for simulations. If you think of (X_1, X_2), (X_3, X_4), (X_5, X_6), (X_7, X_8) as points in the plane $\mathbb{R}^2$, then the sequence of points will exhibit clear "artifacts"; e.g., the points will coverable by an unusually small number of lines.

12.2 Mersenne Twister

The *choice du jour* for pseudorandom number generation is called the *Mersenne Twister* method. It was introduced by Makoto Matsumoto and Takuji Nishimura in 1997–98. It's a bit hard to define, but it's something like a souped-up version of a linear congruential generator; to get the next chunk of 32 bits, you do some bit-twiddling on some of the previously generated chunks, with some more magic numbers involved. The Mersenee Twister has better upsides:

1. It's still fast to compute.
2. The generated numbers pass a lot more kinds of "statistical tests".

You can find the code for Mersenne Twister easily on the web. It's the built-in pseudorandom number generator for Maple, Matlab, and Python, among others.

12.3 Truly pseudorandom numbers?

We saw that linear congruential generators aren't so great because they fail some pretty basic statistical tests. Mersenne Twister is better, but even for it, there are some relatively natural statistical tests it fails. Is the solution to just keep dreaming up more tests and then trying to come up with hacky pseudorandom generators that pass them? Or should we try to identify the "most important" statistcal tests? Neither of these options sounds too satisfying. But how *should* one go about it? Which statistical tests *should* you ensure your random bits pass?

12.4 Truly(?) pseudorandom bits

In many computer scientists' opinion, the ingenious answer to this, the real masterstroke, first appeared in a 1981 paper by CMU's own Professor Manuel Blum and his Ph.D. student Silvio Micali. Their answer: The bits should pass *all polynomial-time statistical tests*. In other words, *no polynomial-time algorithm should be able to "tell the difference" between the pseudorandom bits and truly random bits.*

Two questions spring to mind:

1. Why is this a good answer?
2. Is this an achievable answer?

As for (1), ask yourself: What is the thing you really want out of pseudorandom bits? It's that if you use them in your favorite randomized algorithm $\mathcal{A}$, you'll get the same kind of behavior as if you had run $\mathcal{A}$ with truly random bits. But any algorithm $\mathcal{A}$ you're running in real life is (presumably) *a polynomial-time algorithm.*

As for (2): Blum and Micali originally gave an efficient algorithm for generating such strong pseudorandom bits, based on a certain cryptographic assumption. Subsequent to that, CMU's Professor Lenore Blum, Manuel Blum, and Michael Shub developed an improved version called the Blum-Blum-Shub (BBS) pseudorandom generator. It has the property that no efficient algorithm can tell the difference between its output and truly random bits, *assuming that factoring integers is hard.* This is an even *weaker* cryptographic assumption that the assumption used in RSA encryption. The BBS generator is therefore pretty much the "gold standard" of pseudorandom generators.

There *is* one downside to the BBS generator, though: It's a somewhat slow. Well, actually, it's quite fast. But it's not *super-fast*, like, say, Mersenne Twister. If you need to generate, like, millions of random bits per second, then BBS is not the pseudorandom generator for you. Thus BBS is mainly used these days for cryptographic purposes, when you really really really want to make sure the pseudorandom bits you're generating are indistinguishable from truly random bits.