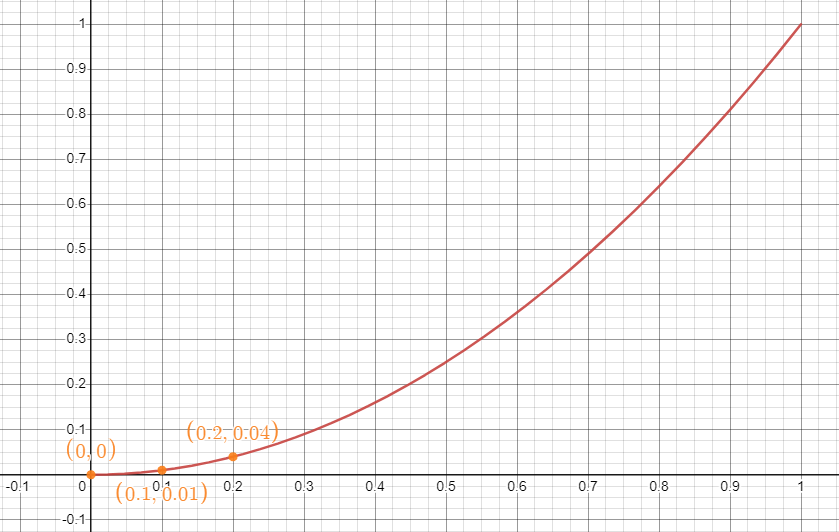
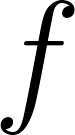
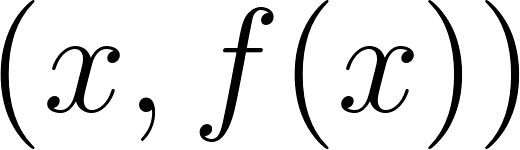
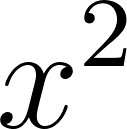
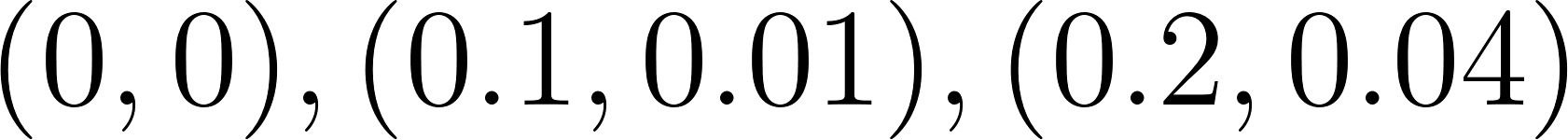
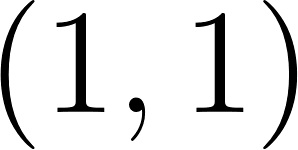
The [inverse transform method](https://en.wikipedia.org/wiki/Inverse_transform_sampling), first introduced in M1L8, will be encountered frequently throughout the course. It's important to note that fully understanding the concept requires time and practice. If you're unsure about what it is or why it works, don't worry—this is completely normal, and you're not alone.

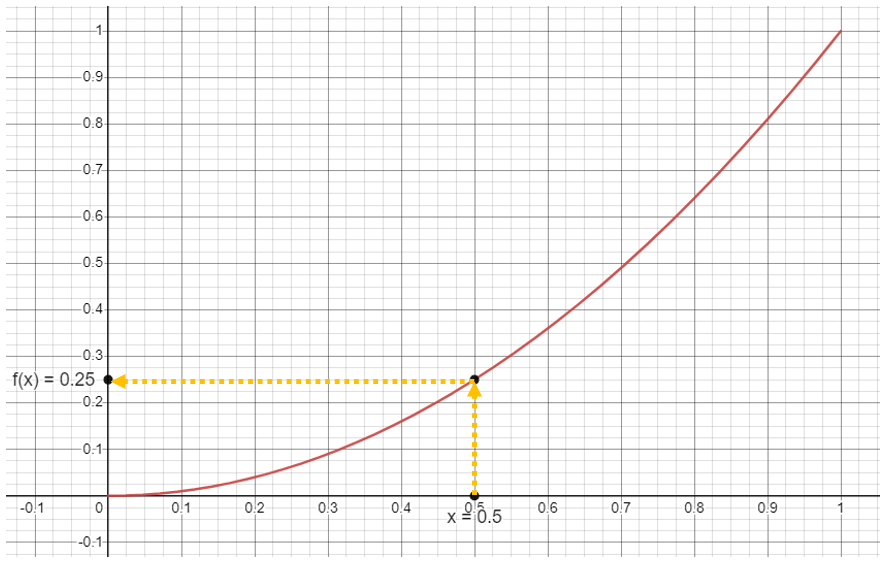
Below is an explanation of the inverse transform theorem/method provided by Angela Smiley, a teaching assistant for this course in previous terms which I have modified and expanded upon throughout my time as a teaching assistant. Angela is the best!

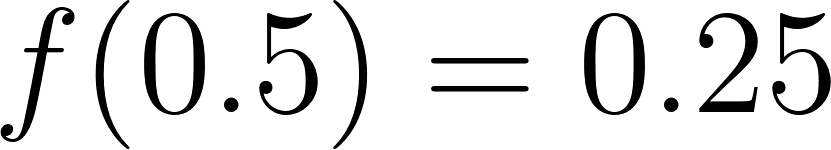
# Preliminaries

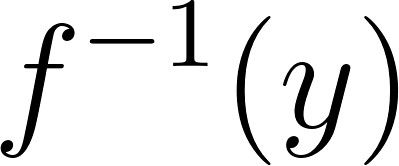


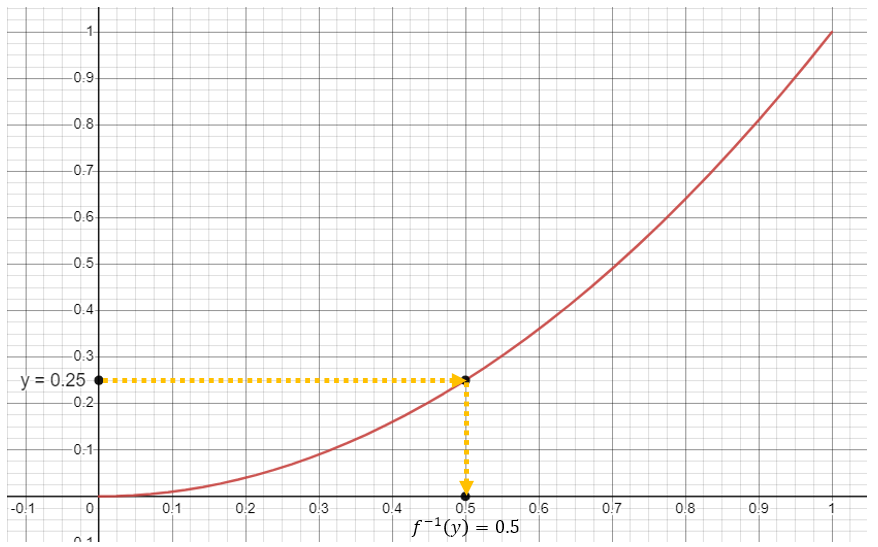
This... is a function! (It gets more interesting from here, promise.) Or more precisely, it's a graph of a function. You render a function [](https://www.codecogs.com/eqnedit.php?latex=f#0)'s graph by putting in a bunch of x values and drawing the point [](https://www.codecogs.com/eqnedit.php?latex=(x%2C%20f(x))#0) for each one. So in this case, for [](https://www.codecogs.com/eqnedit.php?latex=x%5E2#0), you get values like [](https://www.codecogs.com/eqnedit.php?latex=(0%2C0)%2C%20(0.1%2C%200.01)%2C%20(0.2%2C%200.04)#0), and so on, all the way up to [](https://www.codecogs.com/eqnedit.php?latex=(1%2C1)#0).

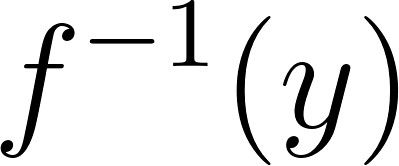
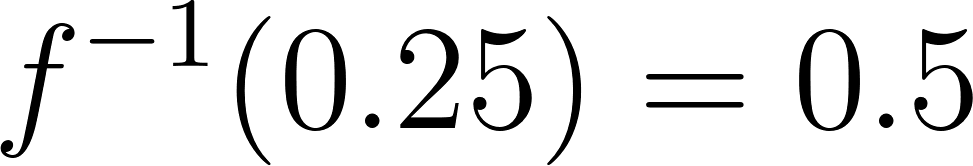
Among other things, this means that if we want to "apply" the function we've just graphed, we can pick a point on the x axis and follow it up to the curve and figure out where that point falls on the y axis:



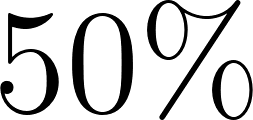
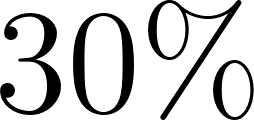
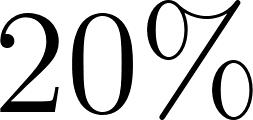
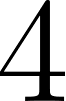
Voila! [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0). In this case, [](https://www.codecogs.com/eqnedit.php?latex=f(0.5)%20%3D%200.25#0).

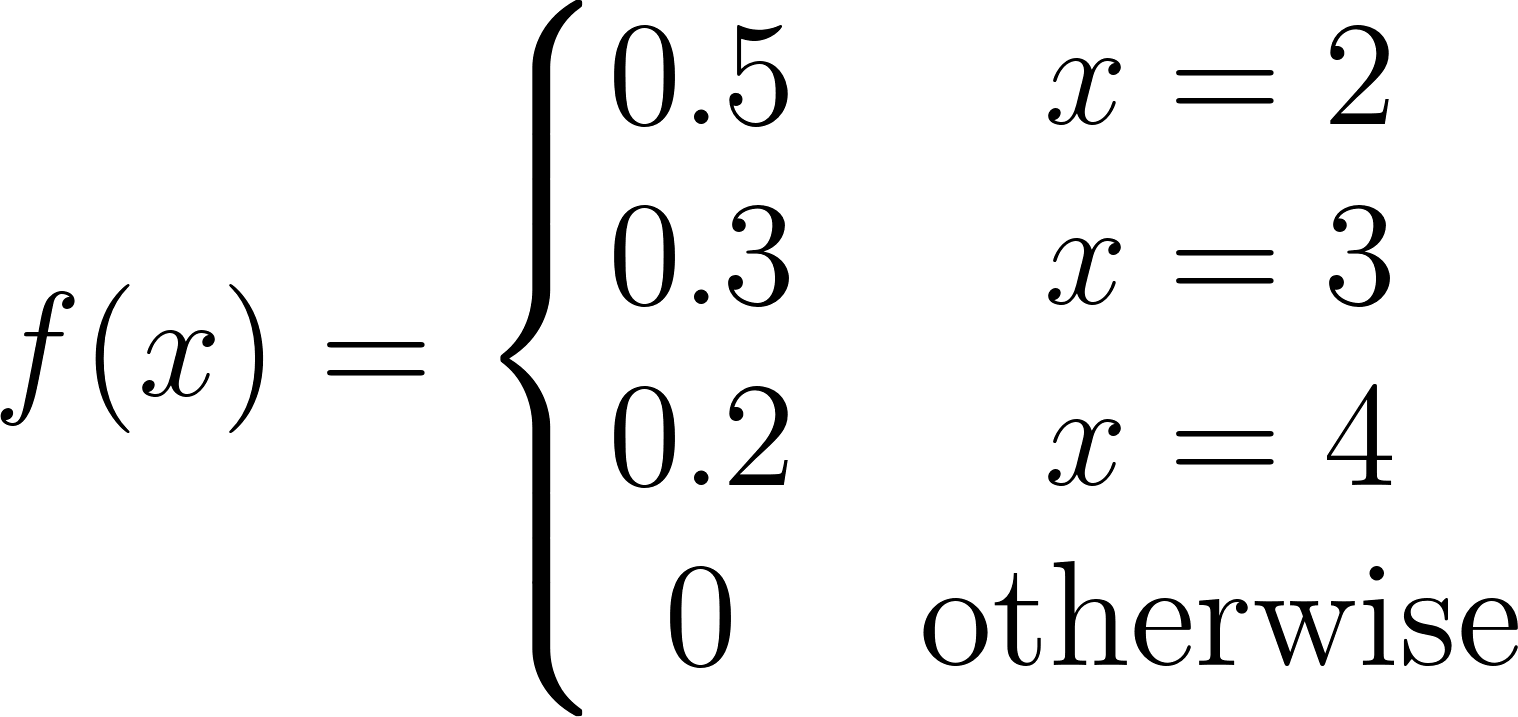
Now, here's the less-obvious corollary: if we want to find [](https://www.codecogs.com/eqnedit.php?latex=f%5E%7B-1%7D(y)#0), we can do something remarkably similar. Pick some point on the y axis and follow it over to reach the curve. Then figure out where that point falls on the x axis:



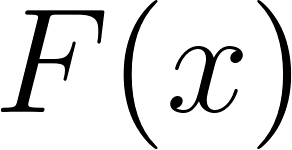
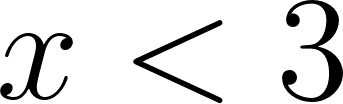
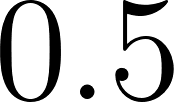
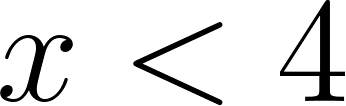
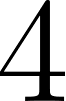
Voila! [](https://www.codecogs.com/eqnedit.php?latex=f%5E%7B-1%7D(y)#0). In this case, [](https://www.codecogs.com/eqnedit.php?latex=f%5E%7B-1%7D(0.25)%20%3D%200.5#0).

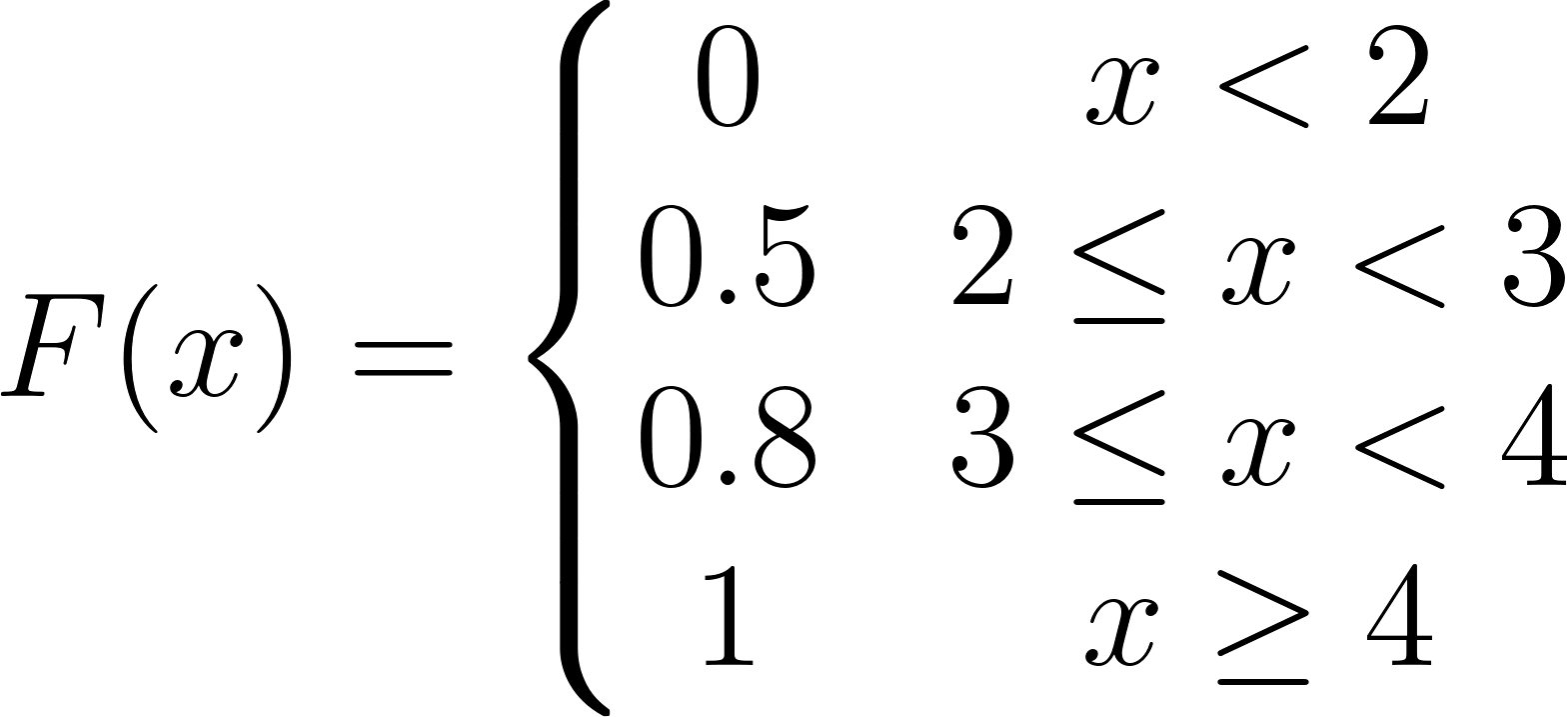
# Discrete distributions

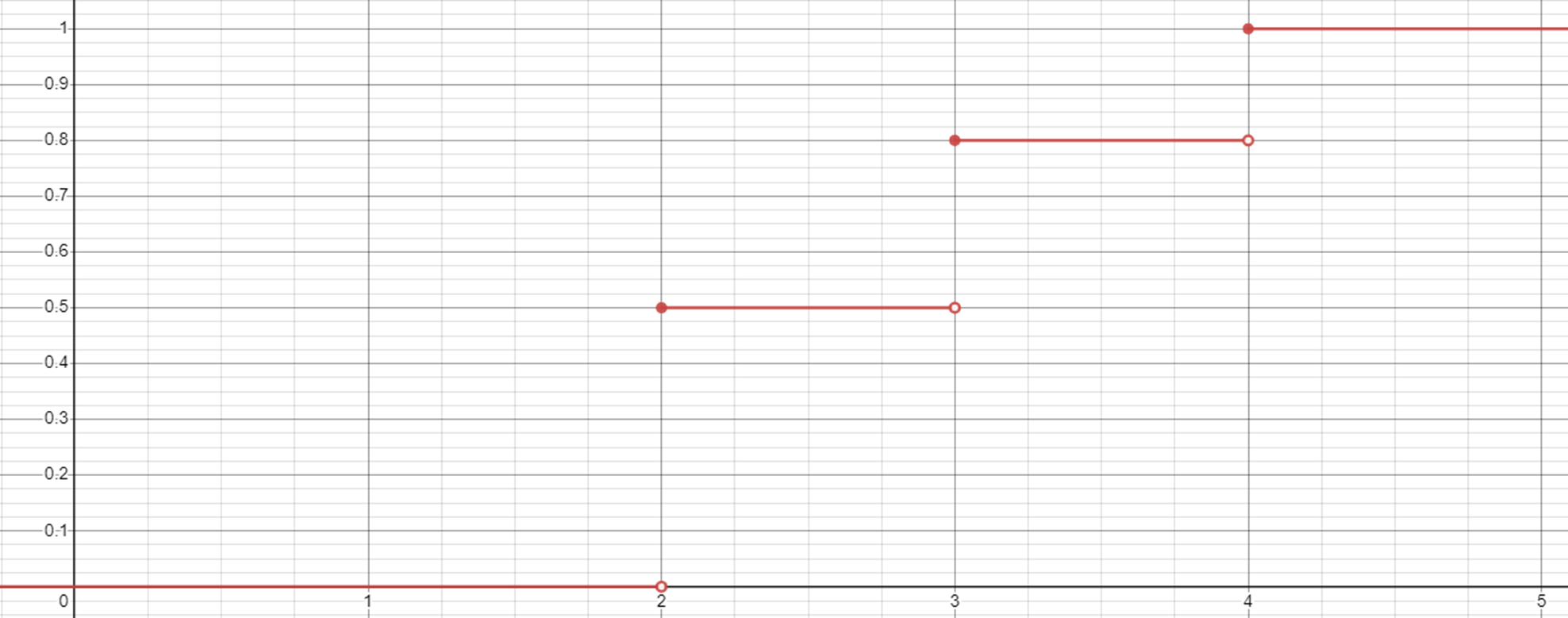
We like to describe [discrete probability distributions](https://en.wikipedia.org/wiki/Probability_distribution#Discrete_probability_distribution) with one of two functions, depending on how we're using them. One function, the probability mass function (PMF) [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0), captures the probability of sampling any given value from the distribution. So for instance, if I tell you "I'm thinking of a number. There's a [](https://www.codecogs.com/eqnedit.php?latex=50%5C%25#0) chance that it's [](https://www.codecogs.com/eqnedit.php?latex=2#0), a [](https://www.codecogs.com/eqnedit.php?latex=30%5C%25#0) chance that it's [](https://www.codecogs.com/eqnedit.php?latex=3#0), and a [](https://www.codecogs.com/eqnedit.php?latex=20%5C%25#0) chance that it's [](https://www.codecogs.com/eqnedit.php?latex=4#0).", I would describe that with this PMF:

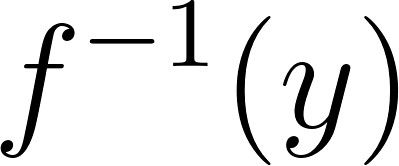
[](https://www.codecogs.com/eqnedit.php?latex=f(x)%3D%5Cleft%5C%7B%5Cbegin%7Bmatrix%7D%200.5%20%26%20x%3D2%5C%5C%200.3%20%26%20x%3D3%20%5C%5C0.2%20%26%20x%3D4%5C%5C0%20%20%26%20%5Ctext%7Botherwise%7D%20%5Cend%7Bmatrix%7D%5Cright.#0)

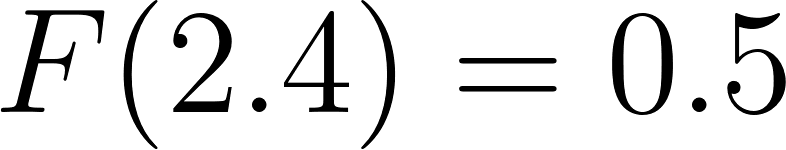
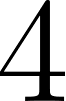
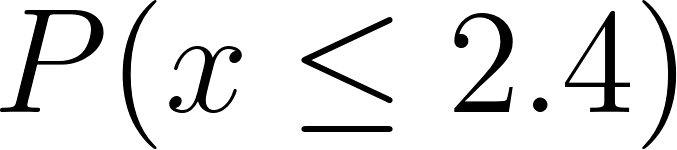
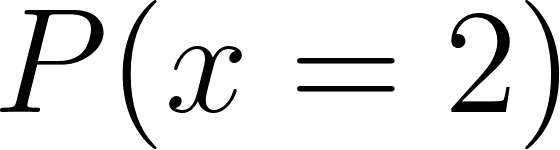


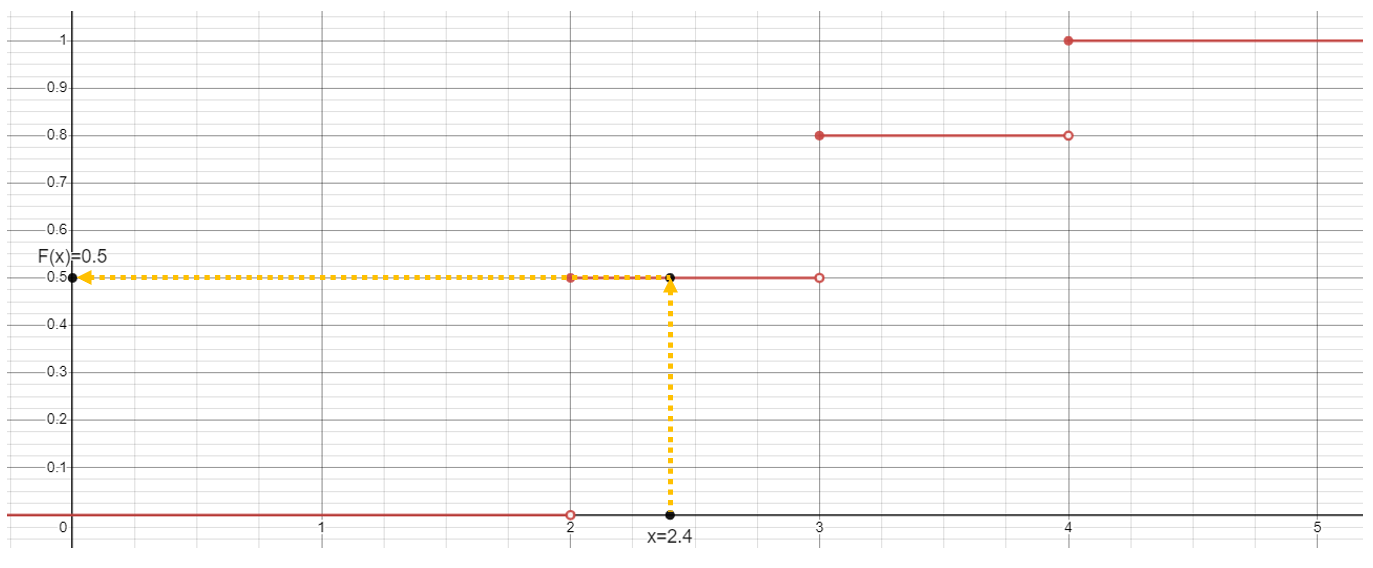
The other function, the CDF or cumulative distribution function [](https://www.codecogs.com/eqnedit.php?latex=F(x)#0), describes the probability that any given sample from this distribution will be less than some threshold, [](https://www.codecogs.com/eqnedit.php?latex=x#0). So for the distribution I described above, the probability of sampling a value less than [](https://www.codecogs.com/eqnedit.php?latex=2#0) is [](https://www.codecogs.com/eqnedit.php?latex=0#0). The probability of sampling [](https://www.codecogs.com/eqnedit.php?latex=x%20%3C%203#0) is [](https://www.codecogs.com/eqnedit.php?latex=0.5#0); the probability of sampling [](https://www.codecogs.com/eqnedit.php?latex=x%20%3C%204#0) is [](https://www.codecogs.com/eqnedit.php?latex=0.5%2B0.3%20%3D%200.8#0); the probability of sampling [](https://www.codecogs.com/eqnedit.php?latex=x#0) less than, say, [](https://www.codecogs.com/eqnedit.php?latex=5#0), is [](https://www.codecogs.com/eqnedit.php?latex=1#0). (This makes sense, because all of the possible x values [](https://www.codecogs.com/eqnedit.php?latex=-2#0), [](https://www.codecogs.com/eqnedit.php?latex=3#0), and [](https://www.codecogs.com/eqnedit.php?latex=4#0) - qualify, right?)

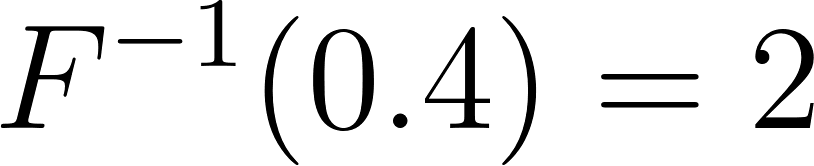
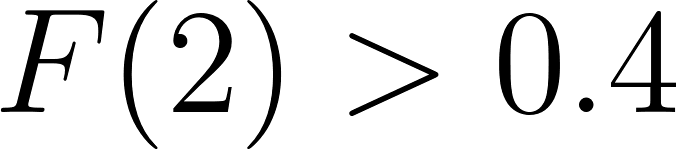
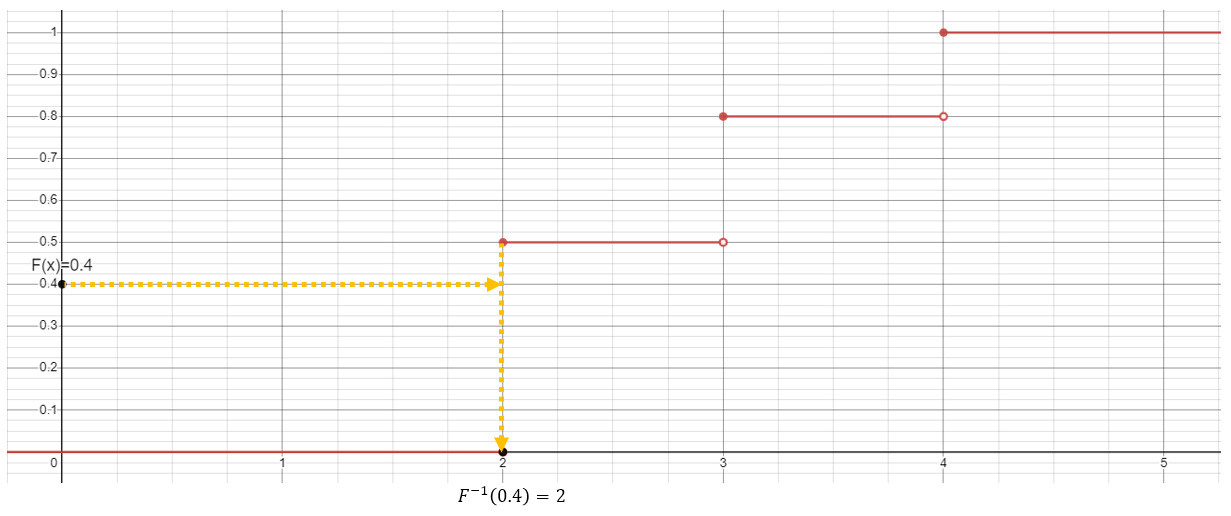
[](https://www.codecogs.com/eqnedit.php?latex=F(x)%3D%5Cleft%5C%7B%5Cbegin%7Bmatrix%7D%200%20%26%20x%3C2%5C%5C%5C%5C%200.5%20%26%202%20%5Cleq%20x%20%3C%203%20%5C%5C0.8%20%26%203%20%5Cleq%20x%20%3C%204%5C%5C1%20%20%20%26%20x%20%5Cgeq%204%20%5Cend%7Bmatrix%7D%5Cright.#0)

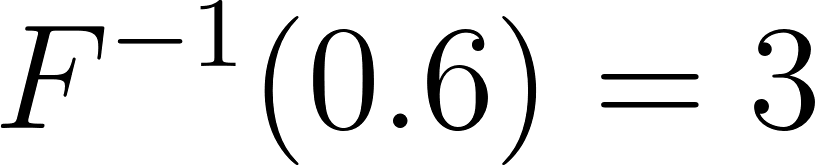
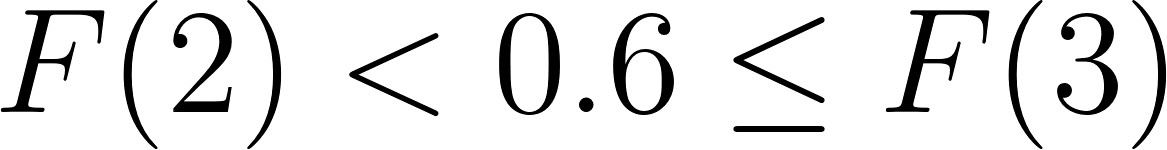


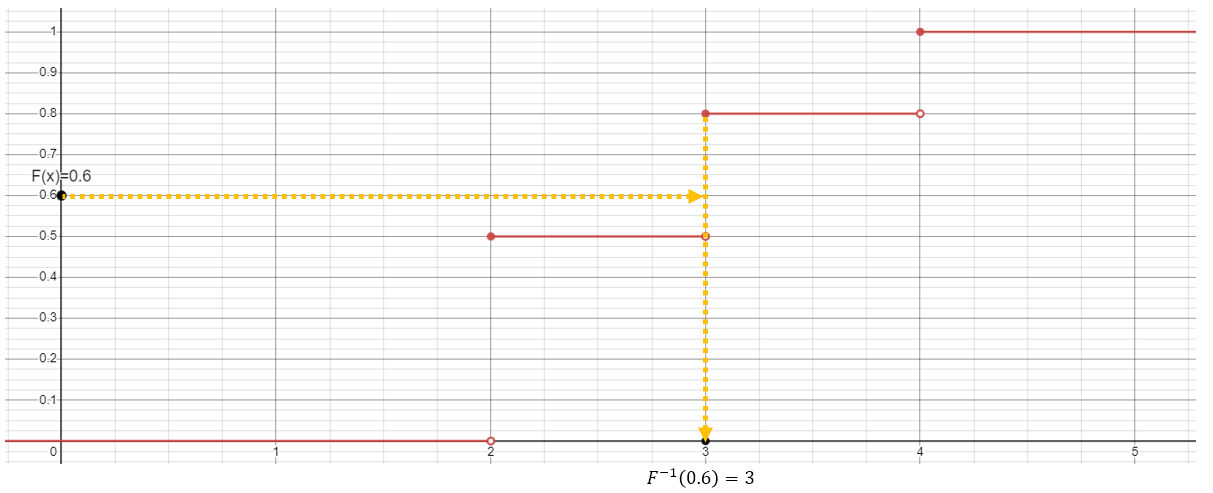
That's a plot of the CDF.. which is a function. In part 1 we were looking at [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0) and [](https://www.codecogs.com/eqnedit.php?latex=f%5E%7B-1%7D(y)#0) using a plot. No reason we can't do the same here!

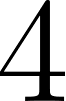
[](https://www.codecogs.com/eqnedit.php?latex=F(2.4)%20%3D%200.5#0), which checks out - for this distribution with possible values [](https://www.codecogs.com/eqnedit.php?latex=2#0), [](https://www.codecogs.com/eqnedit.php?latex=3#0) and [](https://www.codecogs.com/eqnedit.php?latex=4#0), the probability [](https://www.codecogs.com/eqnedit.php?latex=P(x%20%5Cleq%202.4)#0) is simply [](https://www.codecogs.com/eqnedit.php?latex=P(x%20%3D%202)#0).

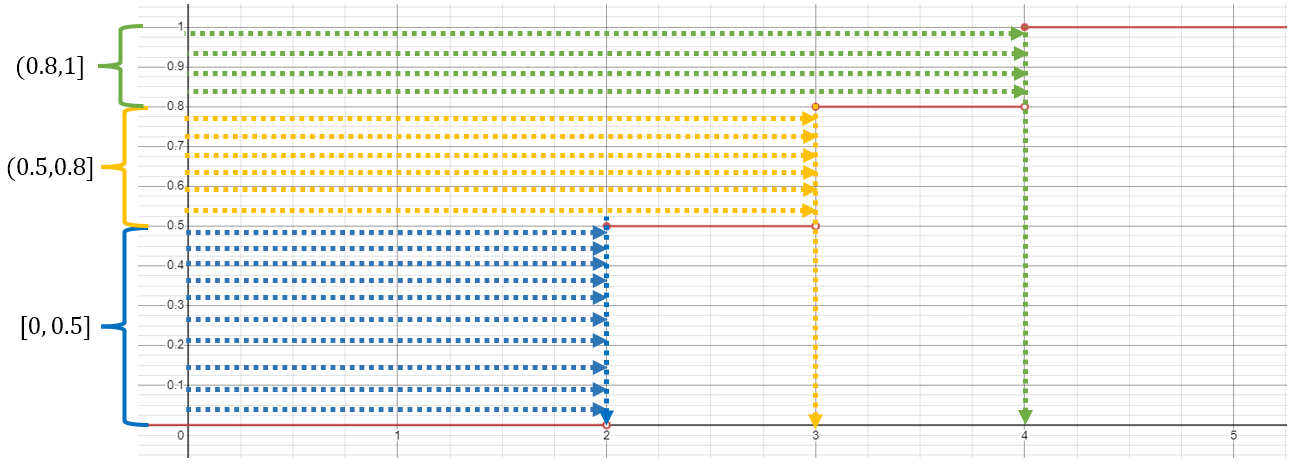


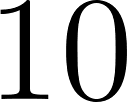
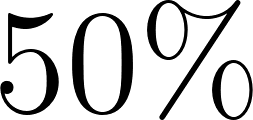
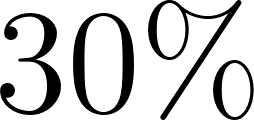
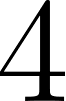
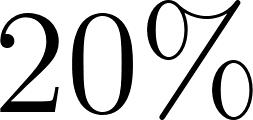
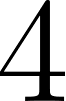
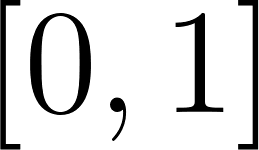
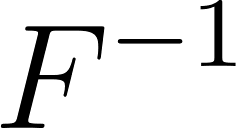
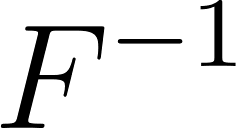
[](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(0.4)%20%3D%202#0), which also checks out - [](https://www.codecogs.com/eqnedit.php?latex=F(2)%20%3E%200.4#0). What happens if we do another one?

[](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(0.6)%20%3D%203#0). Again, makes sense since [](https://www.codecogs.com/eqnedit.php?latex=F(2)%20%3C%200.6%20%5Cleq%20F(3)#0).

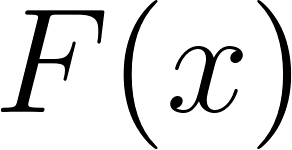


What happens if we do a whole bunch? Well, a lot of them "land" at [](https://www.codecogs.com/eqnedit.php?latex=2#0), some "land" at [](https://www.codecogs.com/eqnedit.php?latex=3#0), and a few "land" at [](https://www.codecogs.com/eqnedit.php?latex=4#0).

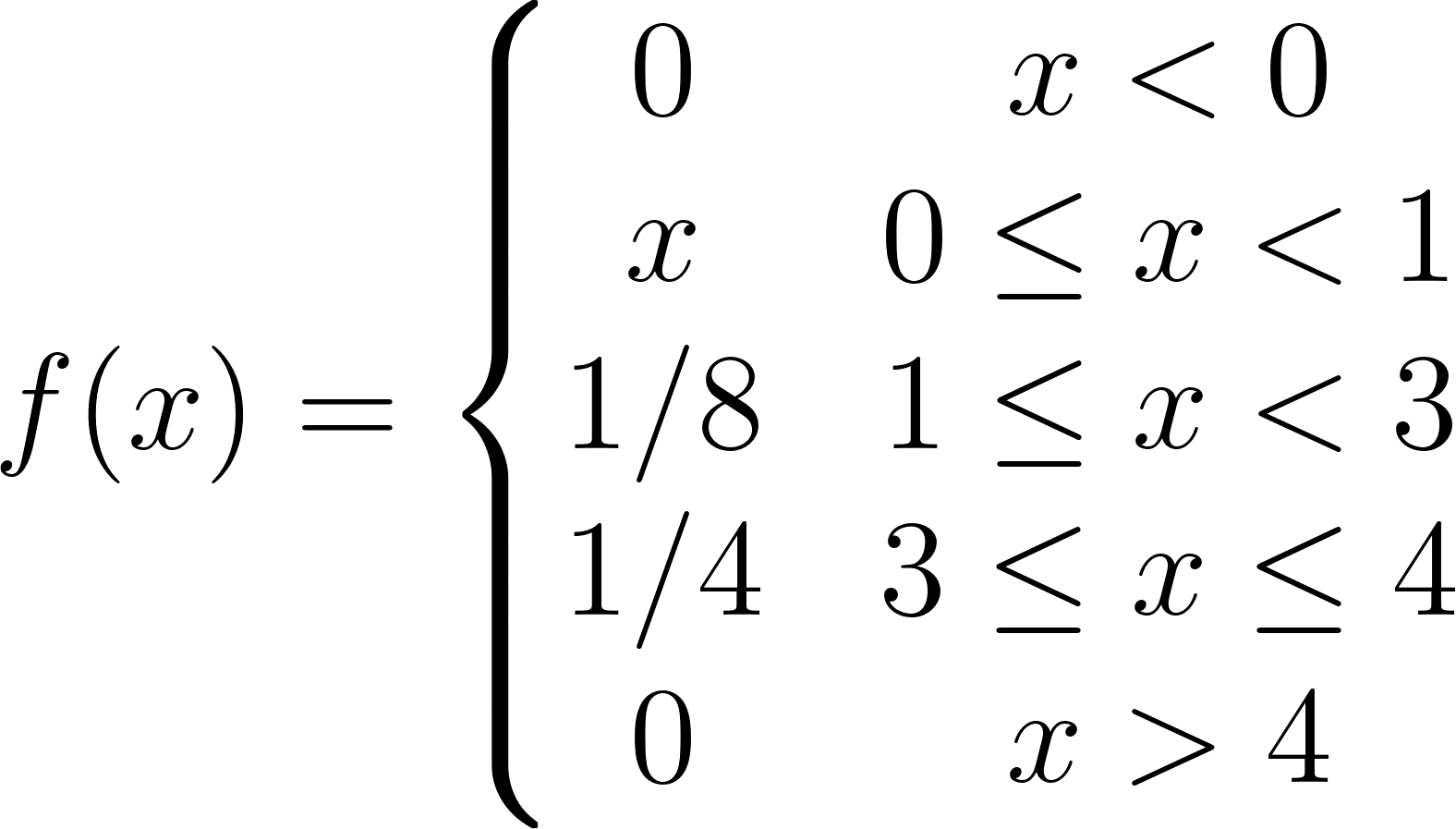


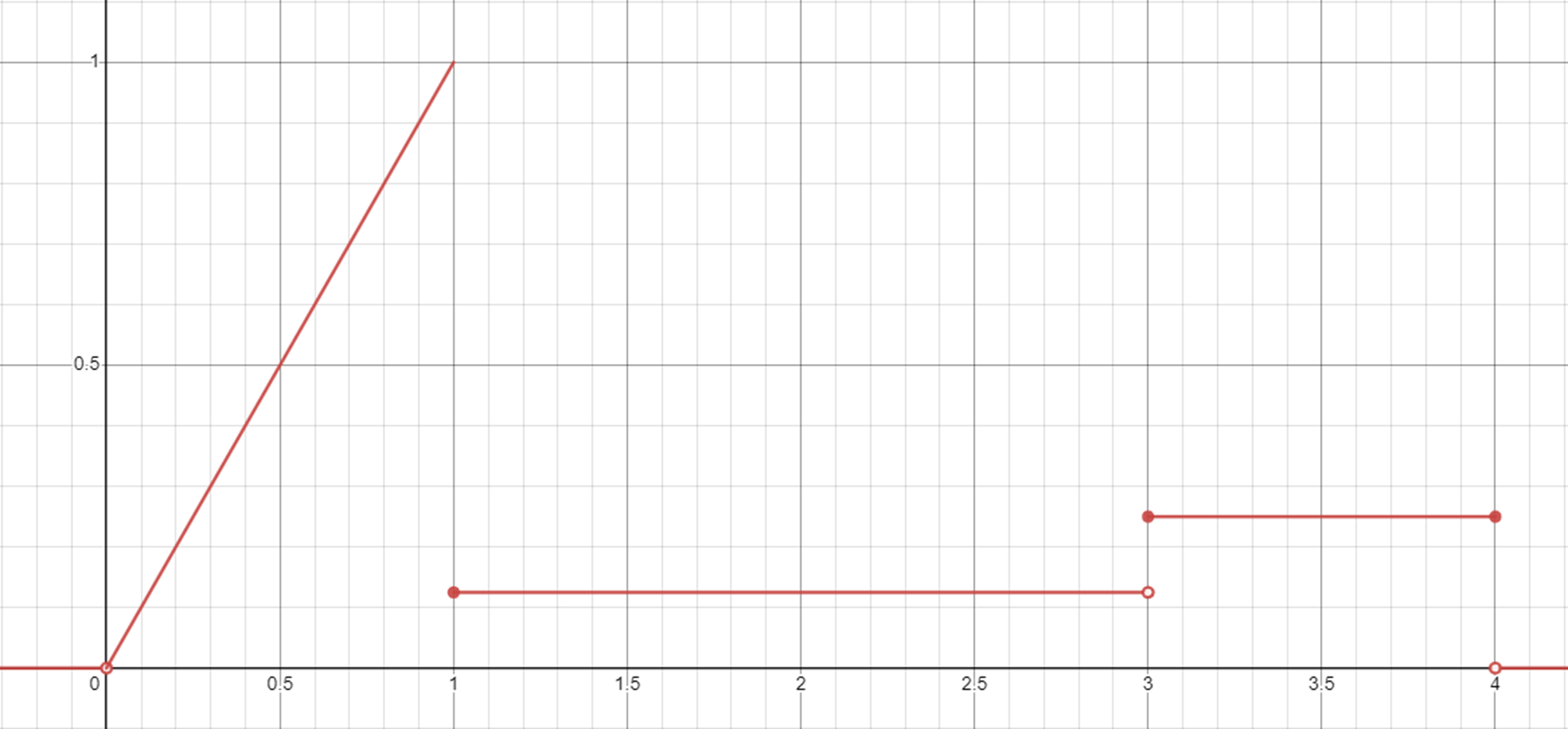
In fact, we can count exactly how many: [](https://www.codecogs.com/eqnedit.php?latex=10#0) of [](https://www.codecogs.com/eqnedit.php?latex=20#0) samples (or [](https://www.codecogs.com/eqnedit.php?latex=50%5C%25#0)), went to [](https://www.codecogs.com/eqnedit.php?latex=2#0); [](https://www.codecogs.com/eqnedit.php?latex=6#0) ([](https://www.codecogs.com/eqnedit.php?latex=30%5C%25#0)) went to [](https://www.codecogs.com/eqnedit.php?latex=3#0); and [](https://www.codecogs.com/eqnedit.php?latex=4#0) ([](https://www.codecogs.com/eqnedit.php?latex=20%5C%25#0)) went to [](https://www.codecogs.com/eqnedit.php?latex=4#0). For these uniformly distributed values in [](https://www.codecogs.com/eqnedit.php?latex=%5B0%2C%201%5D#0), the proportion that "lands" on each number (when we take it through [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D#0)) is exactly equal to the change in height of [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D#0) at that number - which is itself equal to the probability mass associated with that number in the original [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0)! Note that I placed the lines on the chart so it would follow the distribution exactly. When doing random simulations, we rely on the [Law of Large Numbers](https://en.wikipedia.org/wiki/Law_of_large_numbers) so that when we take a lot of samples, we will get the target distribution. If we take a random value between 0 and 1 uniformly, then 50% will fall in the interval that maps back to 2, 30% will fall in the interval that maps back to 3, and 20% will fall in the interval that maps back to 4.

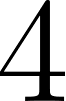
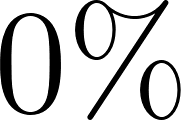
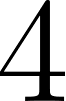
# Continuous distributions

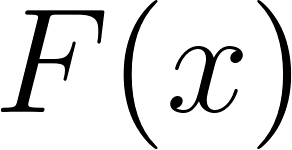
[Continuous distributions](https://en.wikipedia.org/wiki/Probability_distribution#Absolutely_continuous_probability_distribution) can also be described with two functions. One function, the pdf [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0), describes how relatively likely we are to get various numbers by sampling the distribution. The other function, the CDF [](https://www.codecogs.com/eqnedit.php?latex=F(x)#0), works exactly the same way as for a discrete distribution - this function answers the question, "What is the probability that a value from this distribution is less than [](https://www.codecogs.com/eqnedit.php?latex=x#0)?"

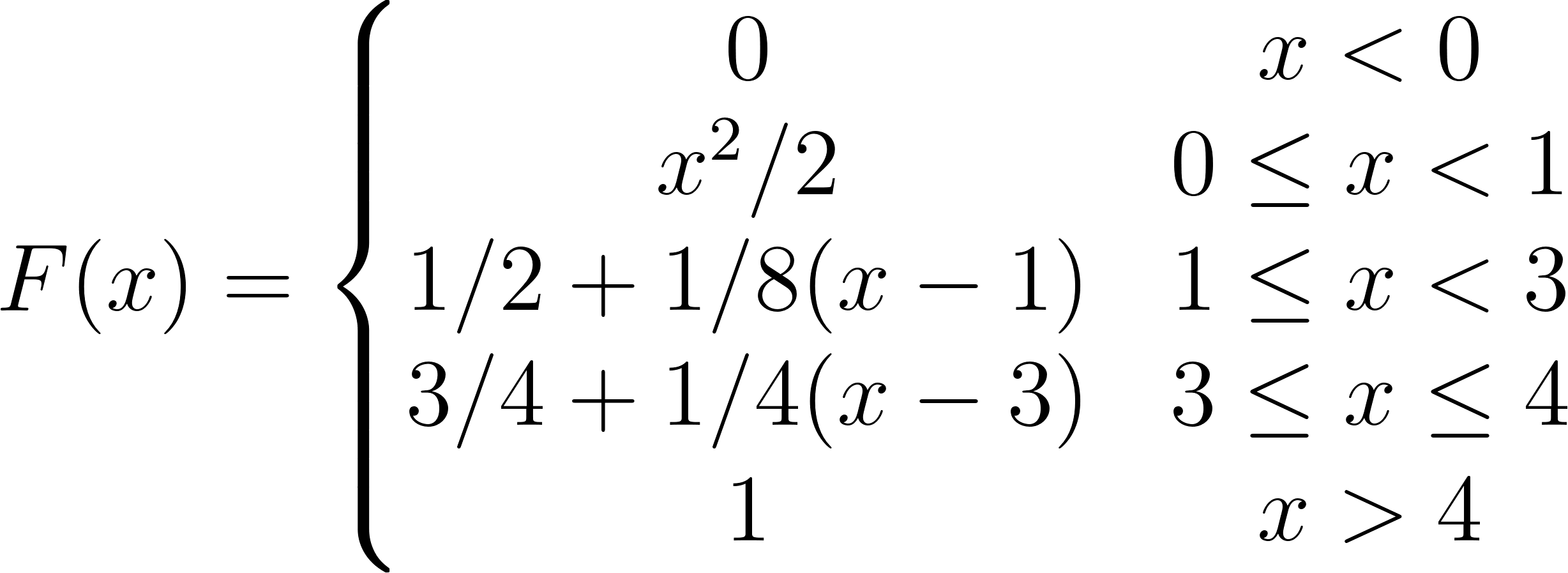
Let's consider a completely arbitrary continuous distribution with the following PDF:

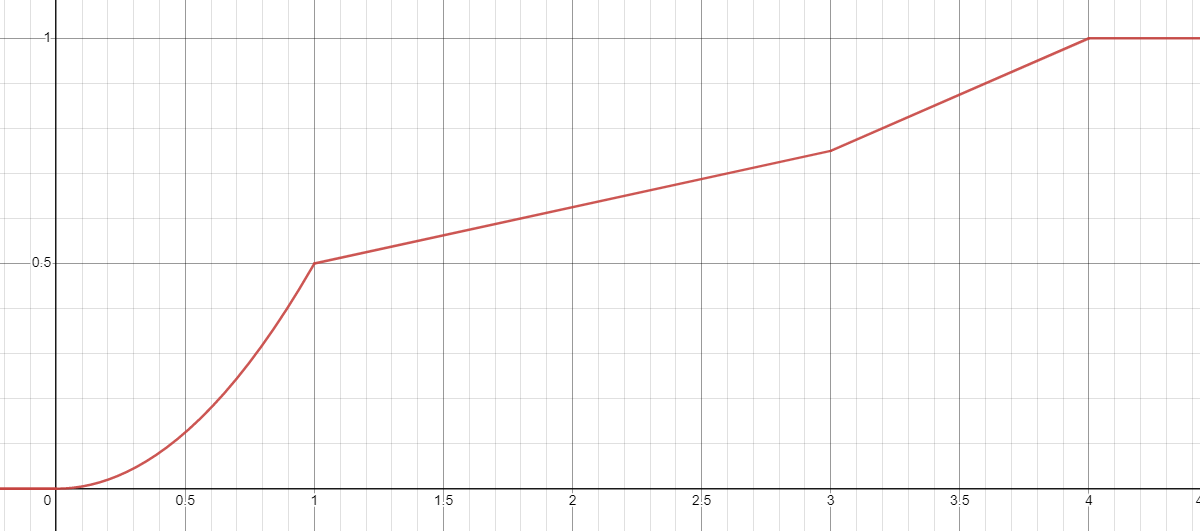
[](https://www.codecogs.com/eqnedit.php?latex=f(x)%3D%5Cleft%5C%7B%5Cbegin%7Bmatrix%7D%200%20%26%20x%3C%200%5C%5C%20x%20%26%200%20%5Cleq%20x%20%3C%201%20%5C%5C%201%2F8%20%26%201%20%5Cleq%20x%20%3C%203%5C%5C%201%2F4%20%20%26%203%20%5Cleq%20x%20%5Cleq%204%20%5C%5C%200%26%20x%3E4%20%5Cend%7Bmatrix%7D%5Cright.#0)



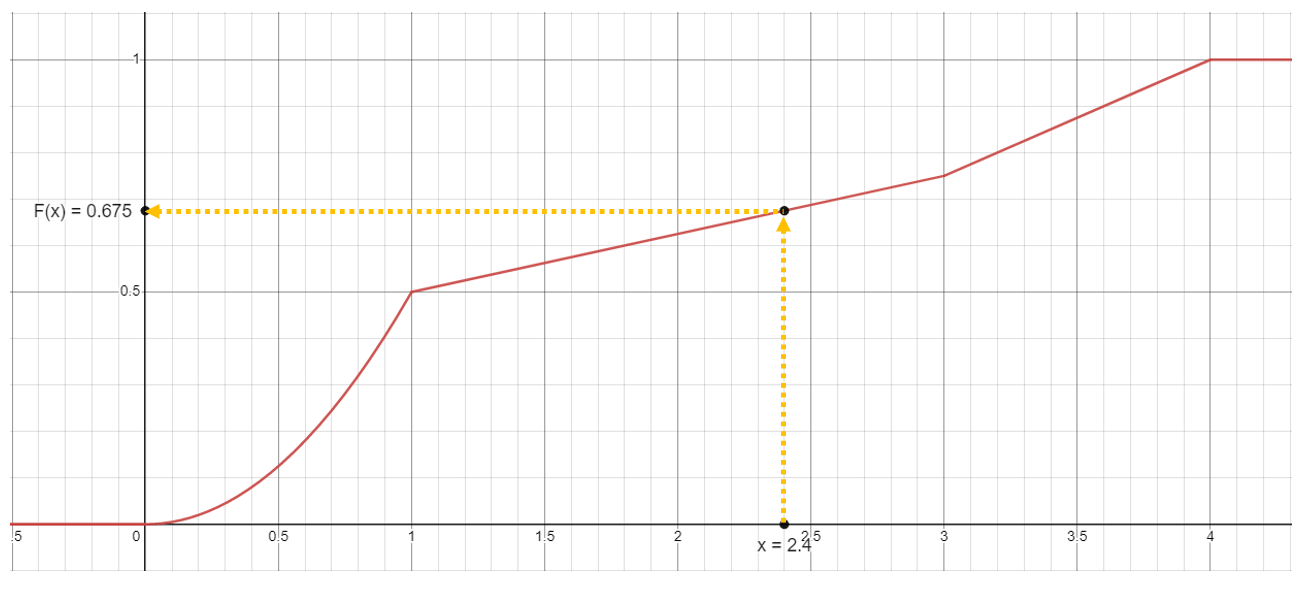
This distribution has some high-ish probability of sampling values between [](https://www.codecogs.com/eqnedit.php?latex=0#0) and [](https://www.codecogs.com/eqnedit.php?latex=1#0), which within that interval is very high near [](https://www.codecogs.com/eqnedit.php?latex=1#0) and drops precipitously towards [](https://www.codecogs.com/eqnedit.php?latex=0#0); a fairly low probability of sampling a given number between [](https://www.codecogs.com/eqnedit.php?latex=1#0) and [](https://www.codecogs.com/eqnedit.php?latex=3#0); a somewhat higher chance of sampling a given number between [](https://www.codecogs.com/eqnedit.php?latex=3#0) and [](https://www.codecogs.com/eqnedit.php?latex=4#0); and a [](https://www.codecogs.com/eqnedit.php?latex=0%5C%25#0) chance of sampling anything greater than [](https://www.codecogs.com/eqnedit.php?latex=4#0) or less than [](https://www.codecogs.com/eqnedit.php?latex=0#0)

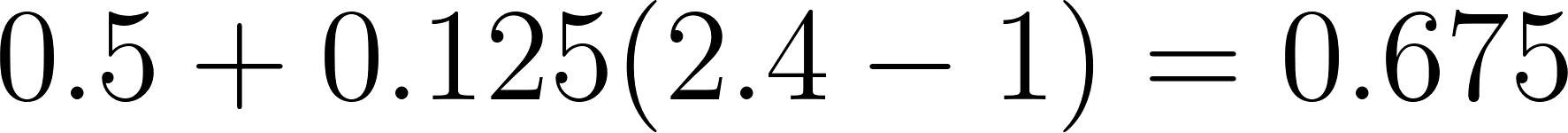
This distribution has the following CDF [](https://www.codecogs.com/eqnedit.php?latex=F(x)#0):

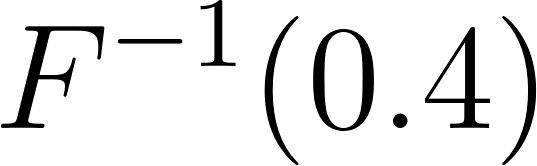
[](https://www.codecogs.com/eqnedit.php?latex=F(x)%3D%5Cleft%5C%7B%5Cbegin%7Bmatrix%7D%200%20%26%20x%3C0%5C%5C%20x%5E2%2F2%20%26%20%200%20%5Cleq%20x%20%3C%201%20%5C%5C1%2F2%20%2B%201%2F8(x-1)%20%26%201%20%5Cleq%20x%20%3C%203%5C%5C3%2F4%20%2B%201%2F4(x-3)%20%20%20%26%203%20%5Cleq%20x%20%5Cleq%204%20%5C%5C%201%20%26%20x%20%3E%204%5Cend%7Bmatrix%7D%5Cright.#0)

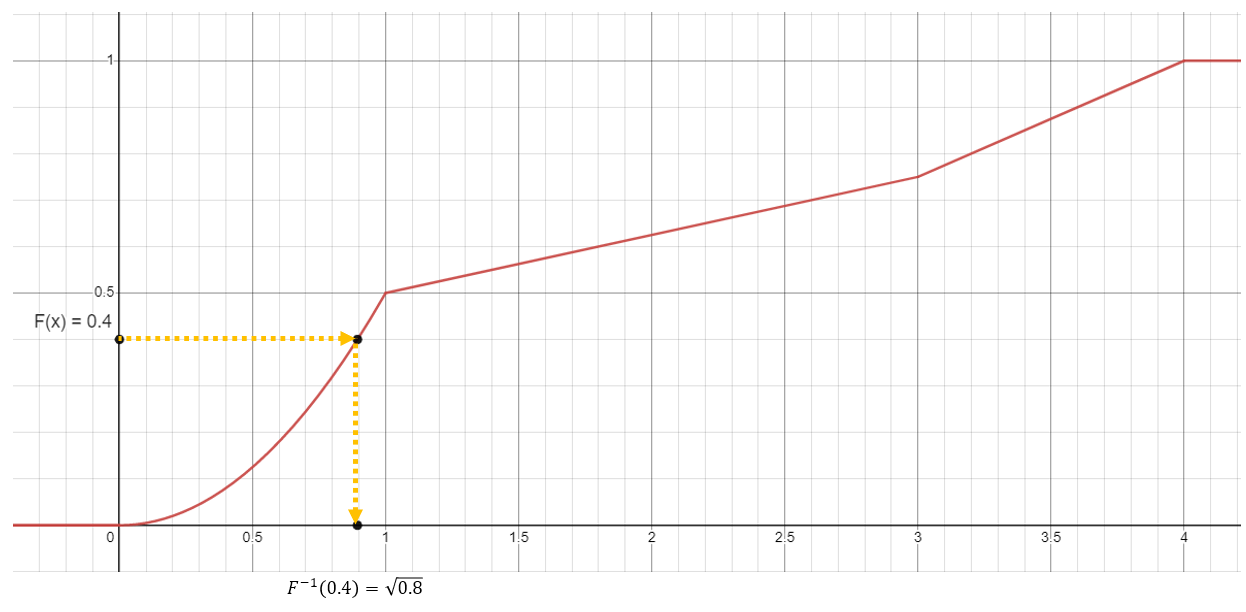


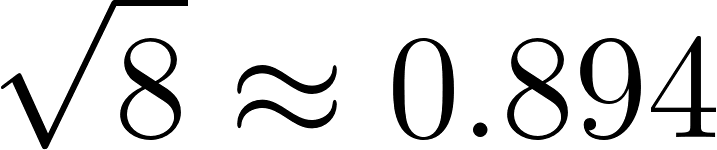
Now, we can do all the same things with this CDF plot that we did with the other one; after all, they are both CDFs. For example, let's approximate the value of [](https://www.codecogs.com/eqnedit.php?latex=F(2.4)#0):

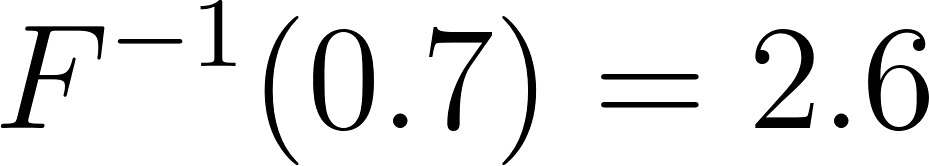


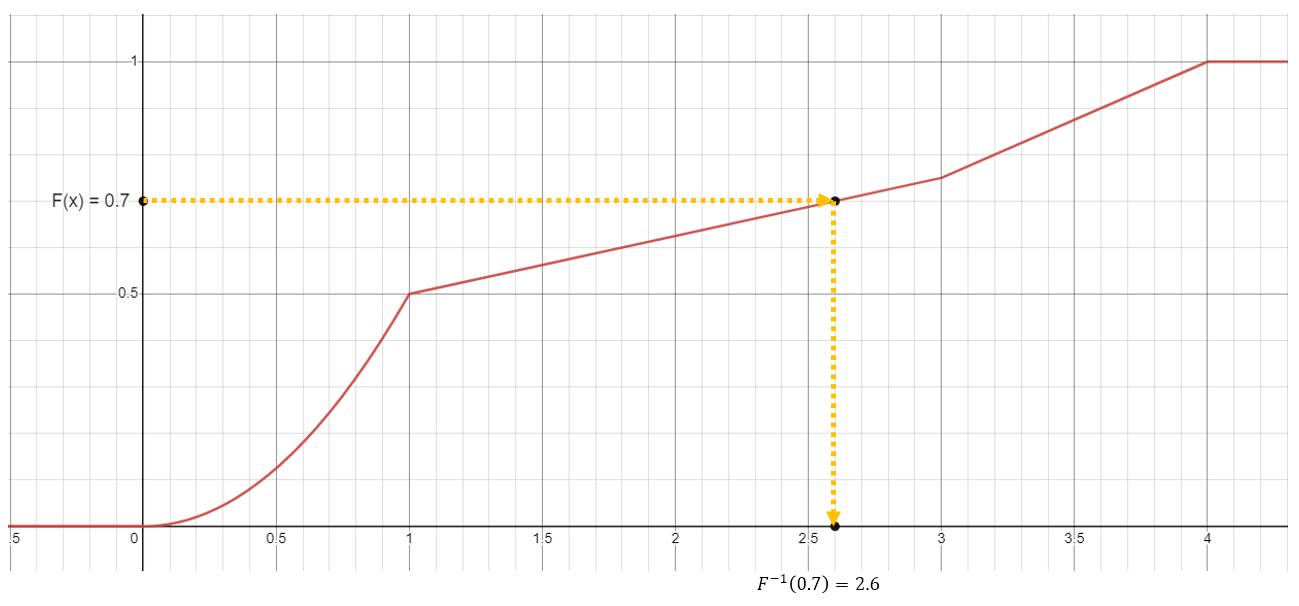
It is [](https://www.codecogs.com/eqnedit.php?latex=0.5%2B0.125(2.4%20-%201)%20%3D%200.675#0)

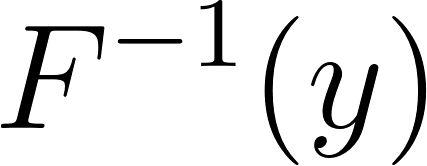
Or, we can find the value of [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(0.4)#0):

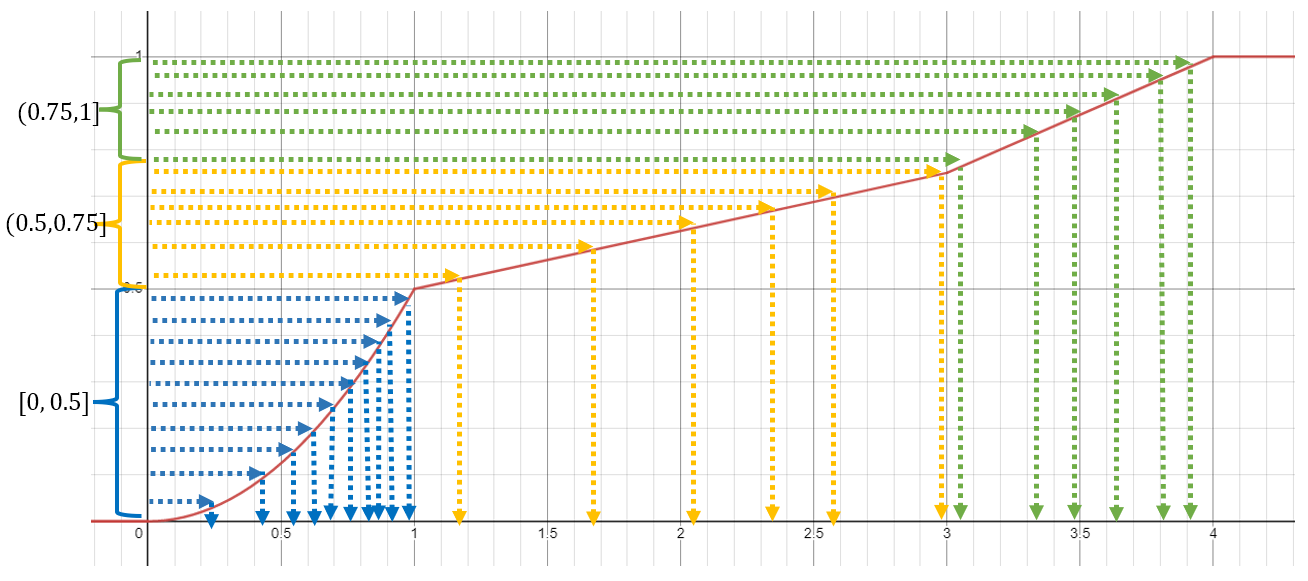


It’s [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B8%7D%20%5Capprox%200.894#0)

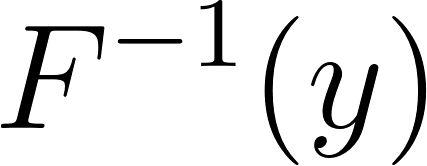
[](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(0.7)%20%3D2.6#0)

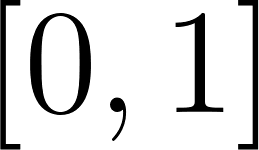
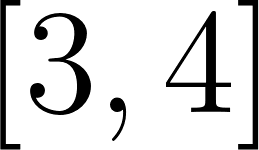
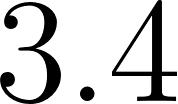
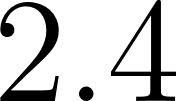


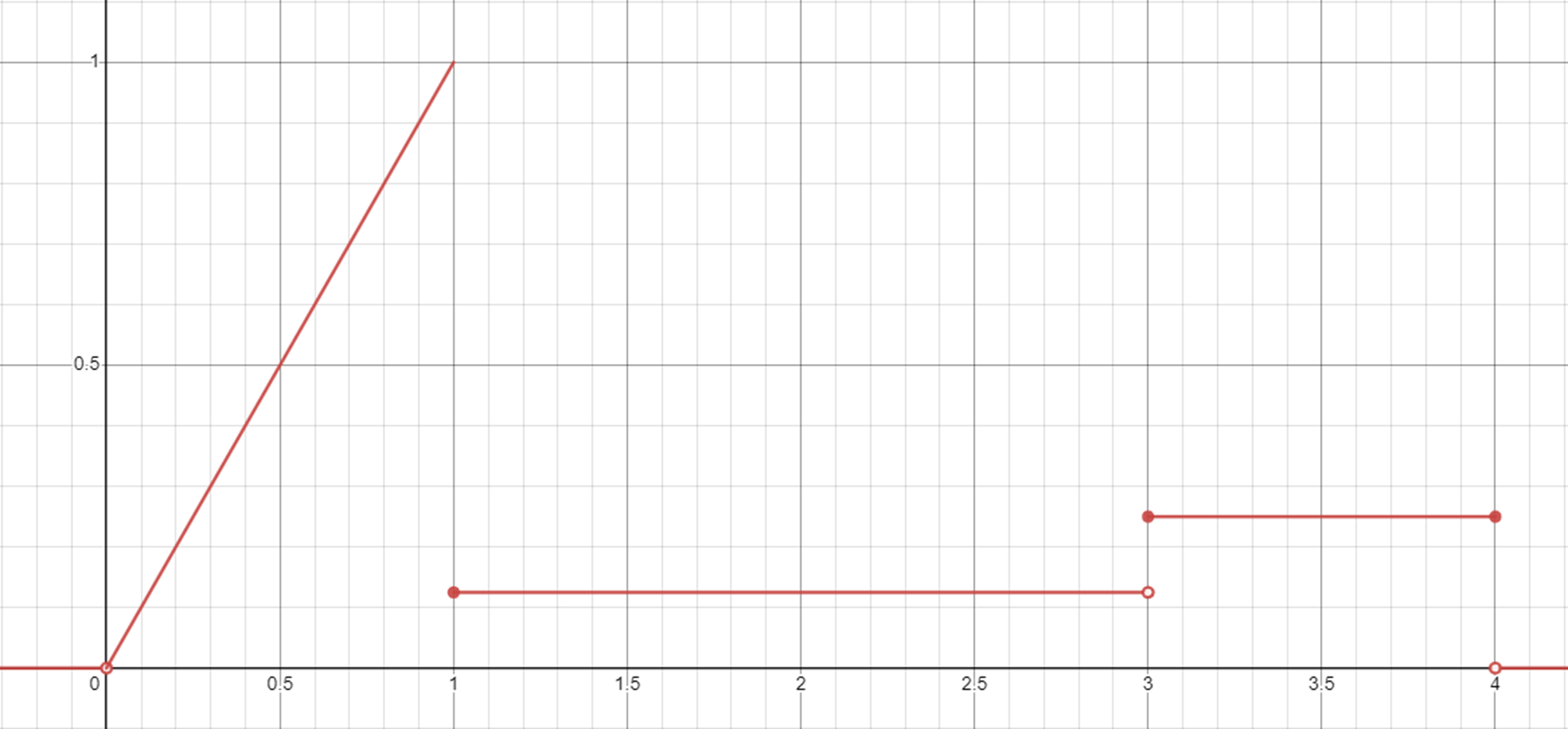
So what happens if we repeat that little exercise we did above, of taking a bunch of samples from the unit interval and mapping them back through [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(y)#0)?



But what does it mean?

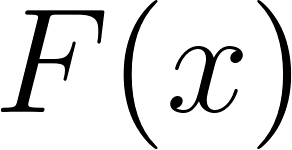
Well, you'll notice some patterns. A lot of [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(y)#0) values end up in the range from [](https://www.codecogs.com/eqnedit.php?latex=0#0) to [](https://www.codecogs.com/eqnedit.php?latex=1#0), with especially many ending up close to [](https://www.codecogs.com/eqnedit.php?latex=1#0). There are relatively fewer that land in the [](https://www.codecogs.com/eqnedit.php?latex=1-3#0) range, and the same absolute number land in the [](https://www.codecogs.com/eqnedit.php?latex=3-4#0) range even though it's half the length in the x direction.

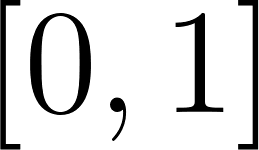
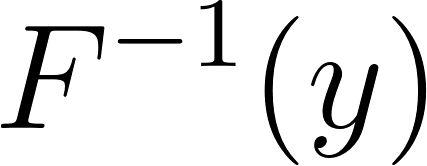
In fact, if we squint, the statement we made above when looking at the discrete CDF graph still seems to hold. It doesn't really make sense to talk about the change in height at a particular point on the graph, but we can talk about the change in height over an interval - and the number of samples that 'hit' it is proportional to that change. If taking a huge interval like [](https://www.codecogs.com/eqnedit.php?latex=%5B0%2C%201%5D#0) or [](https://www.codecogs.com/eqnedit.php?latex=%5B3%2C%204%5D#0) doesn't appeal, we can talk about the slope at a point on the graph - if we could somehow get the slope of every point, we could describe differences like "[](https://www.codecogs.com/eqnedit.php?latex=3.4#0) is twice as likely to be 'hit' as [](https://www.codecogs.com/eqnedit.php?latex=2.4#0)." And by slope I mean derivative. And by somehow, I mean by recycling a graph we already saw above, because...

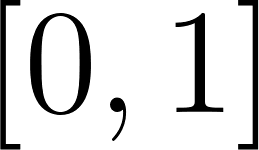
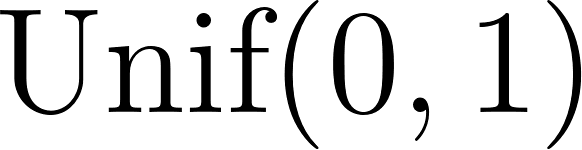
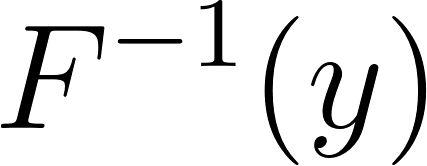


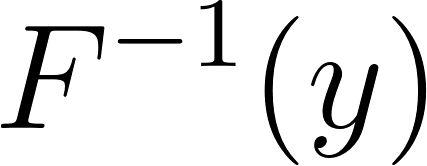
The derivative of the CDF is the PDF.

# Bringing it all together

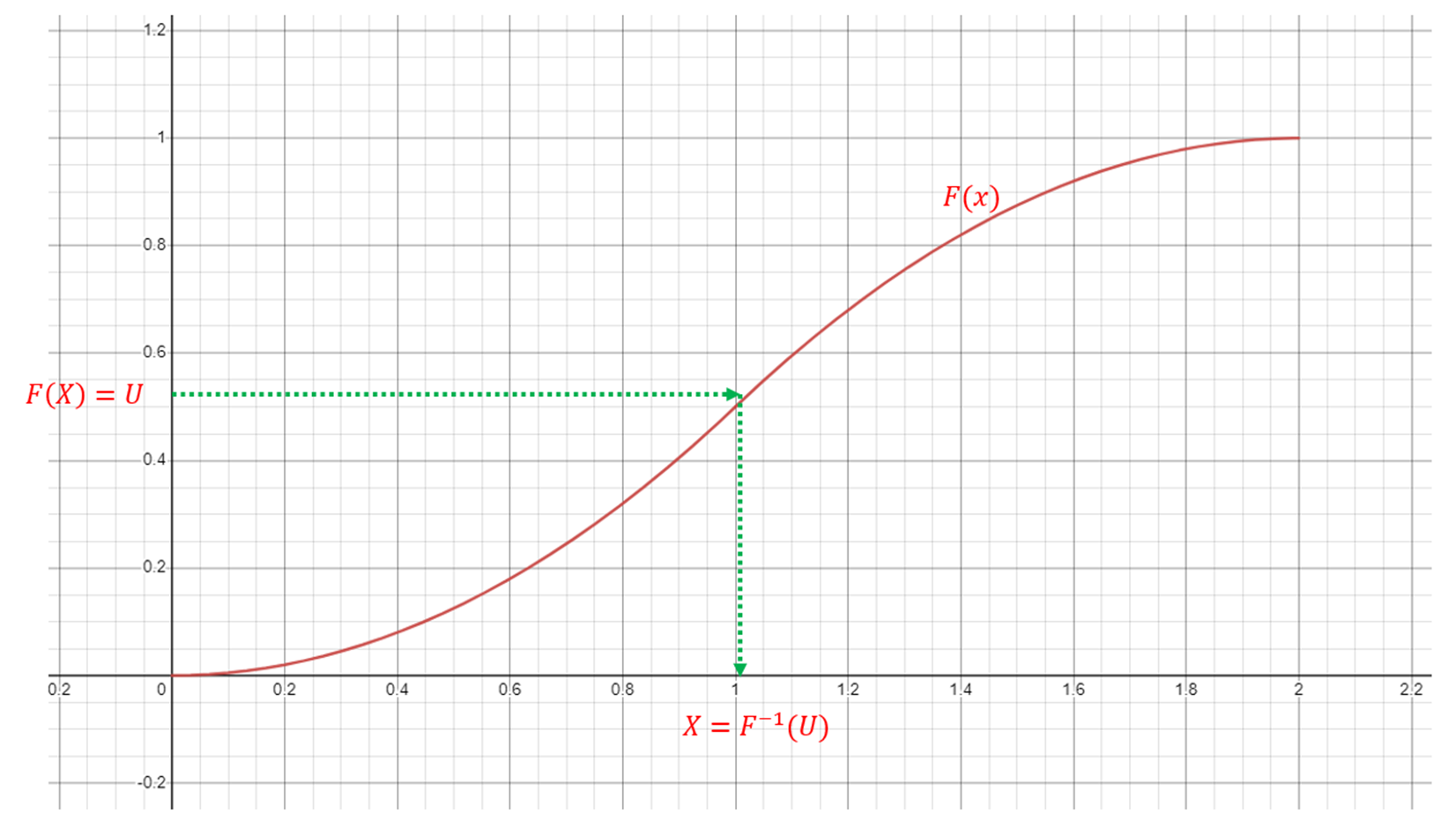
So both types of distributions have a CDF [](https://www.codecogs.com/eqnedit.php?latex=F(x)#0) and a PMF or PDF [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0). The PDF describes the relative likelihoods of obtaining different values when you sample this distribution - that is, the likelihood of sampling value x from the distribution is proportional to [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0).

If we uniformly take a bunch of samples from the interval [](https://www.codecogs.com/eqnedit.php?latex=%5B0%2C%201%5D#0) and send them through the [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(y)#0) looking glass, they will land on the [](https://www.codecogs.com/eqnedit.php?latex=x#0)-axis with a density described by [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0). The more samples we take, the closer the final distribution will be to [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0).

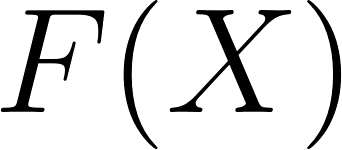
If we take samples uniformly at random from [](https://www.codecogs.com/eqnedit.php?latex=%5B0%2C%201%5D#0) - that is, if we sample [](https://www.codecogs.com/eqnedit.php?latex=%5Ctextrm%7BUnif%7D(0%2C%201)#0) - and dump them into [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(y)#0), what happens? Well, the likelihood of sampling value [](https://www.codecogs.com/eqnedit.php?latex=x#0) will be proportional to [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0). But this is exactly the definition of sampling a distribution with pdf or pmf [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0)!

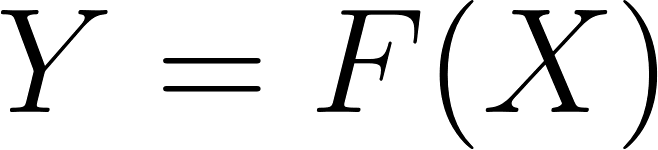
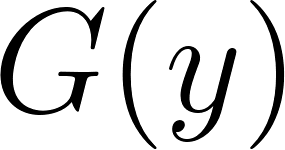
Hence, the inverse transform theorem, which is handy for any distribution where you can obtain [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D(y)#0) - regardless of its type (or ugliness). Maybe continuous and discrete distributions aren't so different after all.

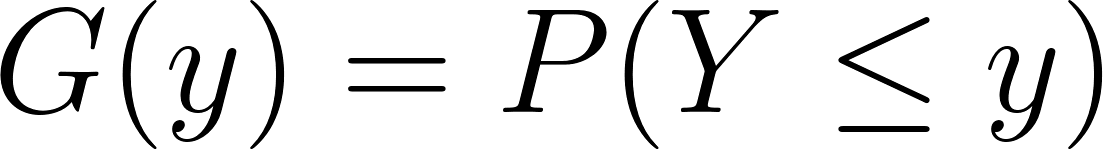
The whole concept revolves around this graphical depiction. We are starting with some value between 0 and 1 and then working it backwards (inverting it) to get a value from the target distribution we wish to generate from.

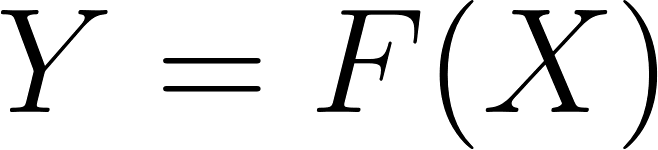


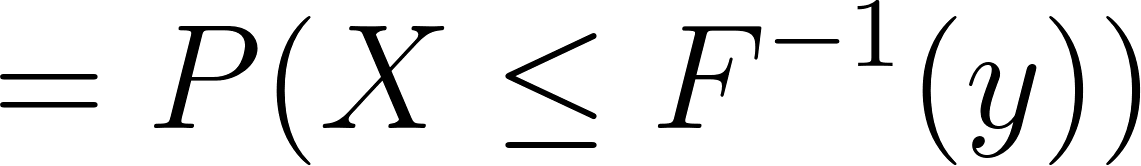
# Proof

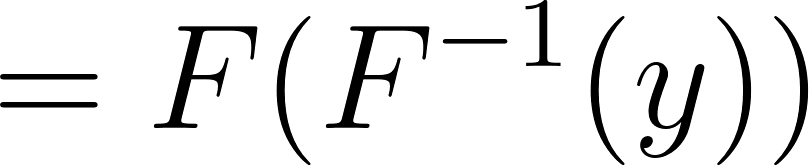
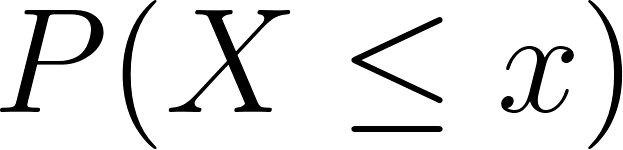
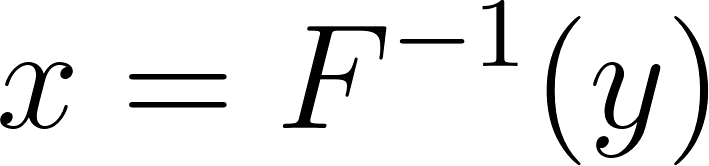
[](https://www.codecogs.com/eqnedit.php?latex=F(X)#0) represents a random variable that is a function of the random variable [](https://www.codecogs.com/eqnedit.php?latex=X#0). It represents picking some random value and finding the value of the CDF there. I like to think of it very informally as taking my finger and putting it somewhere on the [](https://www.codecogs.com/eqnedit.php?latex=y#0)-axis when graphing the CDF. Wherever your finger goes is some random place between [](https://www.codecogs.com/eqnedit.php?latex=0#0) and [](https://www.codecogs.com/eqnedit.php?latex=1#0) each with equal probability (as long as my finger isn't biased which I hope it isn't)

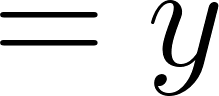
Let [](https://www.codecogs.com/eqnedit.php?latex=Y%3DF(X)#0) and suppose that [](https://www.codecogs.com/eqnedit.php?latex=Y#0) has CDF [](https://www.codecogs.com/eqnedit.php?latex=G(y)#0). (We now want to find the distribution of [](https://www.codecogs.com/eqnedit.php?latex=Y#0) (this is similar to what we've done in M2L8 Functions of a Random Variable)

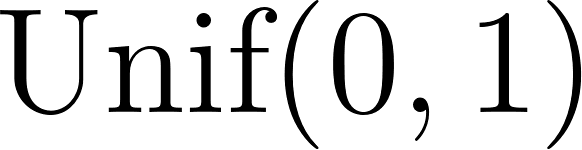
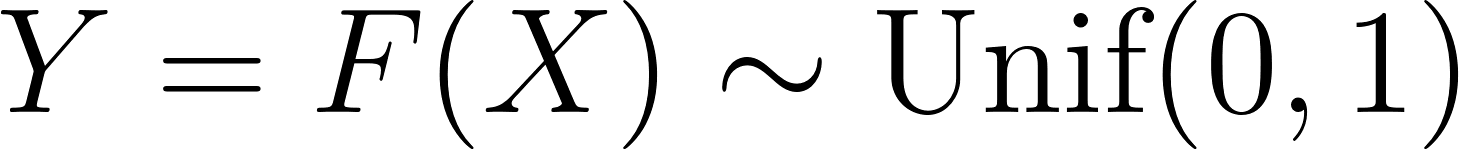
[](https://www.codecogs.com/eqnedit.php?latex=G(y)%3DP(Y%5Cleq%20y)#0) (Definition of a CDF)

[](https://www.codecogs.com/eqnedit.php?latex=%3DP(F(X)%5Cleq%20y)#0) ([](https://www.codecogs.com/eqnedit.php?latex=Y%3DF(X)#0))

[](https://www.codecogs.com/eqnedit.php?latex=%3DP(X%5Cleq%20F%5E%7B-1%7D(y))#0) (Take inverse of both sides)

[](https://www.codecogs.com/eqnedit.php?latex=%3DF(F%5E%7B-1%7D(y))#0) (Definition of a CDF; i.e. we had [](https://www.codecogs.com/eqnedit.php?latex=P(X%20%5Cleq%20x)#0) where [](https://www.codecogs.com/eqnedit.php?latex=x%3DF%5E%7B-1%7D(y)#0))

[](https://www.codecogs.com/eqnedit.php?latex=%3Dy#0) (Applying the CDFto the inverse of a CDF are inverse operations and "cancel" each other out)

[](https://www.codecogs.com/eqnedit.php?latex=y#0) is the CDF of the [](https://www.codecogs.com/eqnedit.php?latex=%5Ctextrm%7BUnif%7D(0%2C1)#0) distribution, so we have shown that [](https://www.codecogs.com/eqnedit.php?latex=Y%3DF(X)%5Csim%20%5Ctextrm%7BUnif%7D(0%2C1)#0)

# Example R Code

## Discrete Distribution

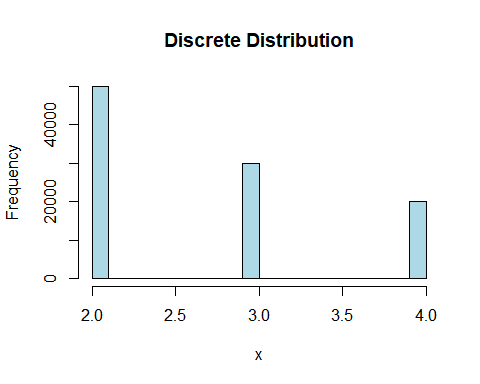
This code is demonstrating inverse transform sampling for a discrete distribution. It follows these steps:

* Define the inverse cumulative distribution function (inverse CDF) for the discrete distribution. This function takes an input value y and returns the corresponding value from the discrete distribution based on the cumulative probabilities: (scroll back up to the graph with the green, blue, and yellow lines to see why this works)
  + If y < 0.5, it returns 2.
  + If 0.5 ≤ y < 0.8, it returns 3.
  + If y ≥ 0.8, it returns 4.
* Set a seed for random number generation to ensure reproducible results (set.seed(1)).
* Define the number of samples to generate (n\_samples <- 1e5, which is equal to 100,000).
* Generate random samples from a uniform distribution between 0 and 1 (uniform\_samples <- runif(n\_samples)). These samples represent uniformly distributed probabilities.
* Apply the inverse CDF for the discrete distribution to the uniform samples. The sapply() function applies the inverse\_cdf\_discrete() function to each value in uniform\_samples and stores the results in a new vector called discrete\_samples.
* Create a histogram to visualize the resulting distribution of the discrete\_samples. This histogram shows the frequencies of the discrete values (2, 3, and 4) in the generated samples.

*# Inverse CDF for the discrete distribution*  
inverse\_cdf\_discrete <- **function**(y) {  
 ifelse(y < 0.5, 2,  
 ifelse(y < 0.8, 3, 4))  
}  
  
*# Generate random samples from a uniform distribution*  
set.seed(1)  
n\_samples <- 1e5  
uniform\_samples <- runif(n\_samples)  
  
*# Apply the inverse CDFs to the uniform samples*  
discrete\_samples <- sapply(uniform\_samples, inverse\_cdf\_discrete)  
  
*# Examine the proportions of each value*  
table(discrete\_samples) / length(discrete\_samples)

## discrete\_samples  
## 2 3 4   
## 0.50008 0.29951 0.20041

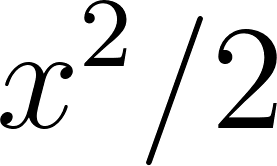
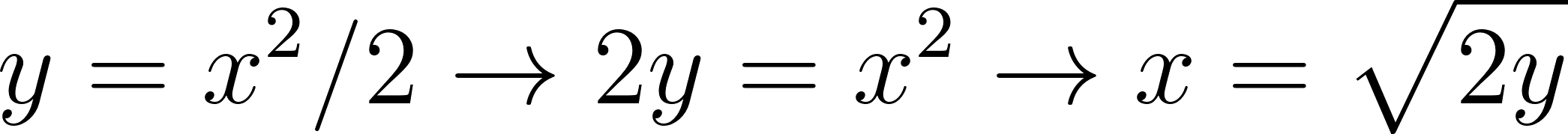
*# Create histograms to visualize the resulting distributions*  
hist(discrete\_samples, main = "Discrete Distribution", xlab = "x",   
 col = "lightblue", border = "black")



Compare this histogram with the graph of the PMF of this random variable.

## Continuous Distribution

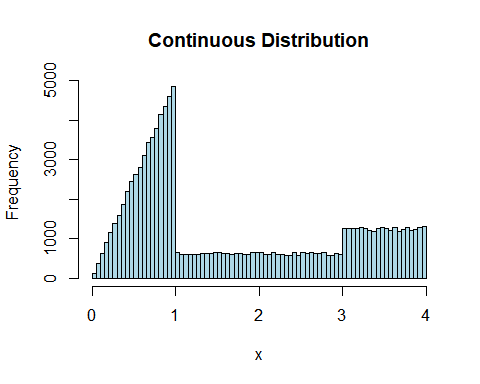
This code demonstrates inverse transform sampling for a continuous distribution and examines the proportions of samples in specified intervals. The code follows these steps:

* Define the inverse cumulative distribution function (inverse CDF) for the continuous distribution. This function takes an input value y and returns the corresponding value from the continuous distribution based on the cumulative probabilities (scroll back to the graph with the green, blue and yellow lines to see how this works)
  + If y < 0.5, it returns sqrt(2 \* y).
    - This is derived from finding the inverse of the piece of the CDF that is [](https://www.codecogs.com/eqnedit.php?latex=x%5E2%2F2#0)
    - [](https://www.codecogs.com/eqnedit.php?latex=y%3Dx%5E2%2F2%20%5Crightarrow%202y%3Dx%5E2%20%5Crightarrow%20x%3D%5Csqrt%7B2y%7D#0)
  + If 0.5 ≤ y < 0.75, it returns 8 \* y - 3.
    - This is derived from finding the inverse of the piece of the CDF that is [](https://www.codecogs.com/eqnedit.php?latex=1%2F2%20%2B%201%2F8(x-1)#0)
    - [](https://www.codecogs.com/eqnedit.php?latex=y%3D1%2F2%20%2B1%2F8(x-1)%20%5Crightarrow%208y%3D4%2Bx-1%20%5Crightarrow%20x%3D8y-3#0)
  + If y ≥ 0.75, it returns 4 \* y.
    - This is derived from finding the inverse of the piece of the CDF that is [](https://www.codecogs.com/eqnedit.php?latex=3%2F4%20%2B1%2F4(x-3)#0)
    - [](https://www.codecogs.com/eqnedit.php?latex=y%3D3%2F4%20%2B1%2F4(x-3)%20%5Crightarrow%204y%3D3%2Bx-3%20%5Crightarrow%20x%3D4y#0)
* Set a seed for random number generation to ensure reproducible results (set.seed(1)).
* Define the number of samples to generate (n\_samples <- 1e5, which is equal to 100,000).
* Generate random samples from a uniform distribution between 0 and 1 (uniform\_samples <- runif(n\_samples)). These samples represent uniformly distributed probabilities.
* Apply the inverse CDF for the continuous distribution to the uniform samples. The sapply() function applies the inverse\_cdf\_continuous() function to each value in uniform\_samples and stores the results in a new vector called continuous\_samples.
* Examine the proportions of each interval by creating a table with cut() and dividing by the total number of samples (length(continuous\_samples)). The breaks parameter defines the intervals to consider: [0, 1), [1, 3), and [3, 4).
* Create a histogram to visualize the resulting distribution of the continuous\_samples. This histogram shows the frequencies of the generated samples within the specified intervals.

*# Inverse CDF for the continuous distribution*  
inverse\_cdf\_continuous <- **function**(y) {  
 ifelse(y < 0.5, sqrt(2 \* y),  
 ifelse(y < 0.75, 8 \* y - 3, 4 \*y))  
}  
  
*# Generate random samples from a uniform distribution*  
set.seed(1)  
n\_samples <- 1e5  
uniform\_samples <- runif(n\_samples)  
  
*# Apply the inverse CDFs to the uniform samples*  
continuous\_samples <- sapply(uniform\_samples, inverse\_cdf\_continuous)  
  
*# Examine the proportions of each interval*  
table(cut(continuous\_samples,   
 breaks = c(0, 1, 3, 4))) / length(continuous\_samples)

##   
## (0,1] (1,3] (3,4]   
## 0.50008 0.24911 0.25081

*# Create histograms to visualize the resulting distributions*  
hist(continuous\_samples, 100, main = "Continuous Distribution", xlab = "x",   
 col = "lightblue", border = "black")



Compare this histogram with the graph of the PDF of this random variable.

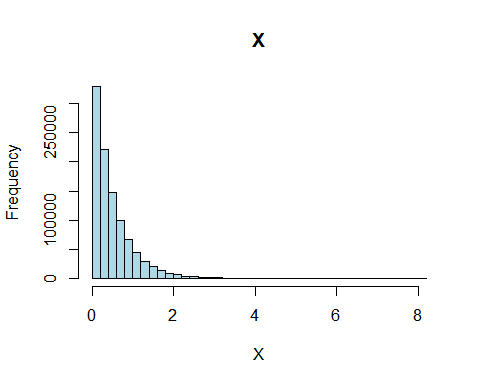
## F(X) is Uniform

This code demonstrates the transformation of a set of exponential random variates back to a uniform distribution using the cumulative distribution function (CDF) of the exponential distribution.

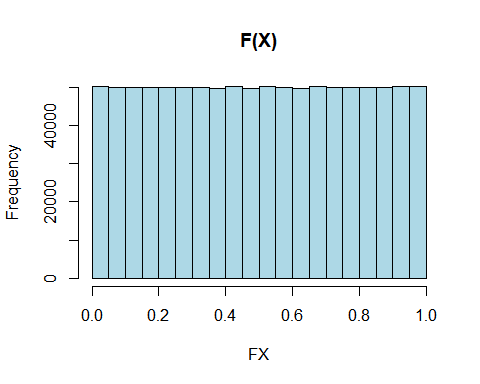
* Set a seed for random number generation to ensure reproducible results (set.seed(1)).
* Define the parameter lambda for the exponential distribution (lambda <- 2).
* Generate a large number of exponential random variates (1 million) using the rexp() function with the specified rate parameter lambda. The generated random variates are stored in the variable X.
* Create a histogram of the generated exponential random variates with 30 bins, using the hist() function. The title of the histogram is set to "X" (main = "X").
* Transform the exponential random variates X back to a uniform distribution using the CDF of the exponential distribution. The CDF is given by F(x) = 1 - exp(-lambda \* x). The transformed values are stored in the variable FX.
* Create a histogram of the transformed uniform distribution values FX, with 30 bins, using the hist() function. The title of the histogram is set to "F(X)" (main = "F(X)").

The two histograms display the distribution of the original exponential random variates (X) and the transformed uniform distribution values (F(X)). The second histogram should resemble a uniform distribution, demonstrating that applying the CDF to the exponential random variates results in a uniform distribution.

set.seed(1)  
  
*# Generate a bunch of exponential random variates*  
lambda <- 2  
X <- rexp(1e6, rate = lambda)  
hist(X, 30, main = "X", col = "lightblue", border = "black")



*# Plug them back into the CDF of the exponential*  
FX <- 1 - exp(-lambda \* X)  
hist(FX, 30, main = "F(X)", col = "lightblue", border = "black")



## U vs. 1-U

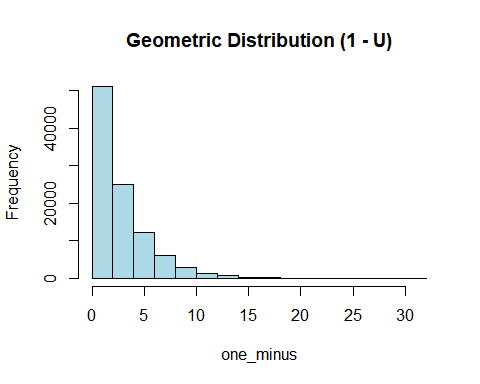
In the long run, [](https://www.codecogs.com/eqnedit.php?latex=U#0) or [](https://www.codecogs.com/eqnedit.php?latex=1-U#0) will generate the target distribution for \*most\* distributions but not all. Each individual result will be different, but you will still get the correct distribution.

### Geometric Distribution

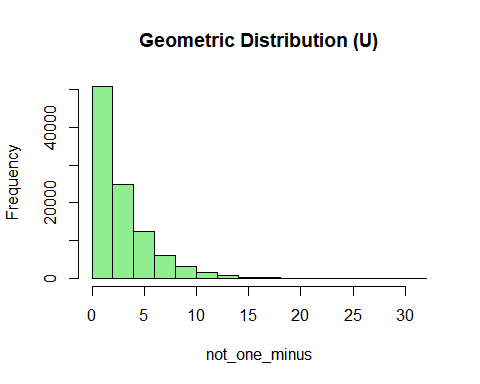
This code generates random samples from the geometric distribution with success probability p, using inverse transform sampling, and computes the mean and variance..

* Set a seed for random number generation to ensure reproducible results (set.seed(6644)).
* Define the success probability p for the geometric distribution (p <- 0.3).
* Define the number of random samples to generate (n <- 1e5).
* Generate n uniform random samples between 0 and 1 using the runif() function and store them in the variable U.
* Generate random samples from a geometric distribution using the formula ceil((log(1-U)) / (log(1-p))). Store the generated samples in the variable one\_minus.
* Create a histogram of the one\_minus samples.
* Generate random samples from a geometric distribution using the formula ceil((log(U)) / (log(1-p))). Store the generated samples in the variable not\_one\_minus.
* Create a histogram of the not\_one\_minus samples.
* Calculate the mean and variance of the one\_minus and not\_one\_minus samples using the mean() and var() functions, respectively.

set.seed(6644)  
p <- 0.3  
n <- 1e5  
  
U <- runif(n)  
  
one\_minus <- ceiling((log(1-U)) / (log(1-p)))  
hist(one\_minus, main = "Geometric Distribution (1 - U)",   
 col = "lightblue", border = "black")



not\_one\_minus <- ceiling((log(U)) / (log(1-p)))  
hist(not\_one\_minus, main = "Geometric Distribution (U)",   
 col = "lightgreen", border = "black")



mean(one\_minus)

## [1] 3.32418

var(one\_minus)

## [1] 7.774965

mean(not\_one\_minus)

## [1] 3.34249

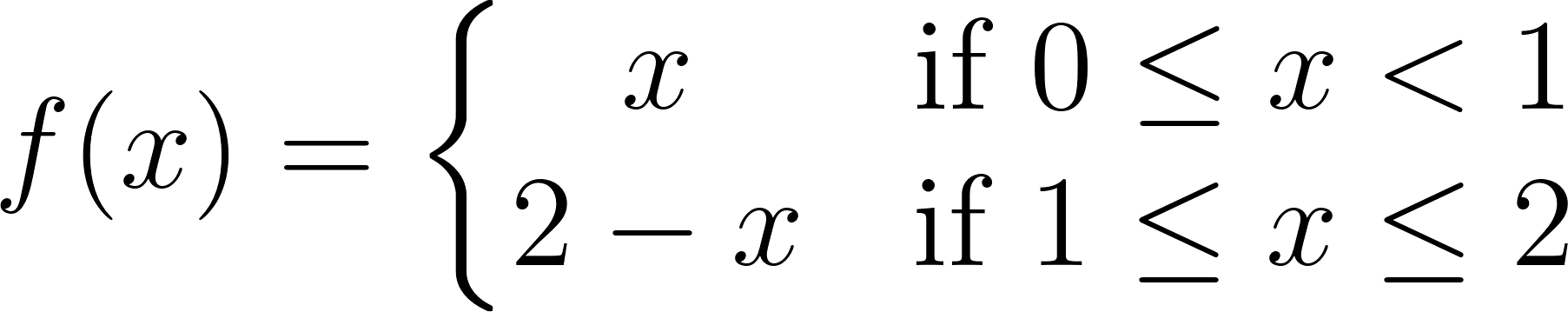
var(not\_one\_minus)

## [1] 7.711968

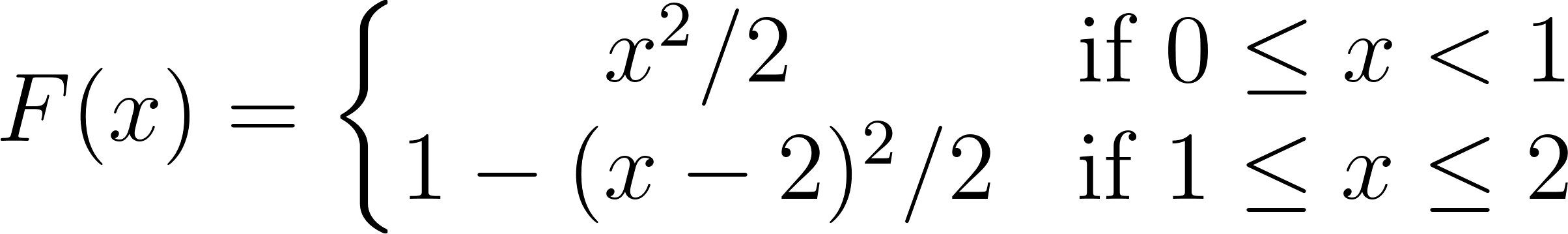
### Triangular Distribution

So when would using [](https://www.codecogs.com/eqnedit.php?latex=U#0) or [](https://www.codecogs.com/eqnedit.php?latex=1-U#0) matter? In M7L3 there is an example using the triangular distribution.

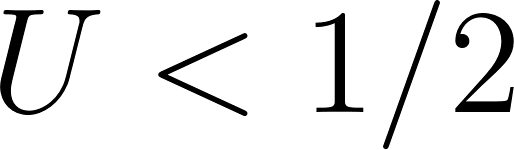
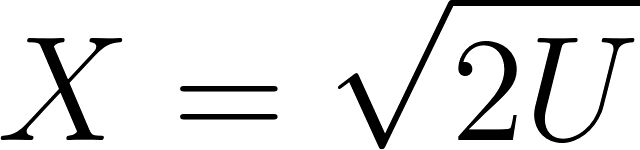
You can try using inverse transform with the following triangular distribution:

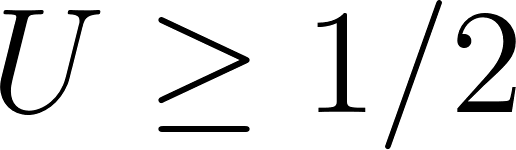
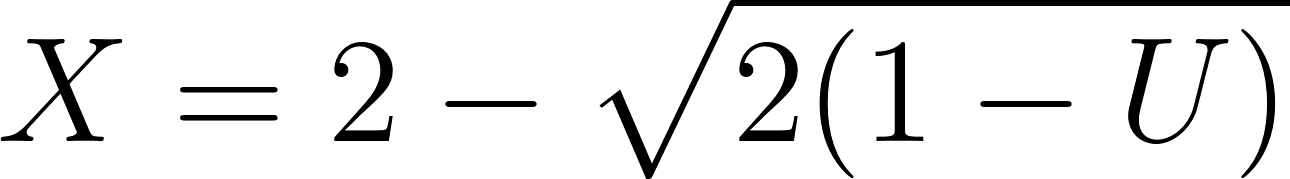
[](https://www.codecogs.com/eqnedit.php?latex=f(x)%3D%5Cleft%5C%7B%5Cbegin%7Bmatrix%7D%20x%20%26%20%5Ctextrm%7Bif%20%7D%200%5Cleq%20x%3C1%20%5C%5C%202-x%20%26%20%5Ctextrm%7Bif%20%7D%201%20%5Cleq%20x%20%5Cleq%202%20%5Cend%7Bmatrix%7D%5Cright.#0)

The cumulative distribution function is

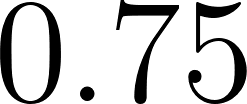
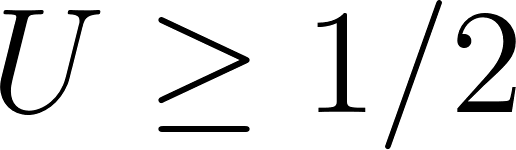
[](https://www.codecogs.com/eqnedit.php?latex=F(x)%3D%5Cleft%5C%7B%5Cbegin%7Bmatrix%7D%20x%5E2%2F2%20%26%20%5Ctextrm%7Bif%20%7D%200%5Cleq%20x%3C1%20%5C%5C%201-(x-2)%5E2%2F2%20%26%20%5Ctextrm%7Bif%20%7D%201%20%5Cleq%20x%20%5Cleq%202%20%5Cend%7Bmatrix%7D%5Cright.#0)

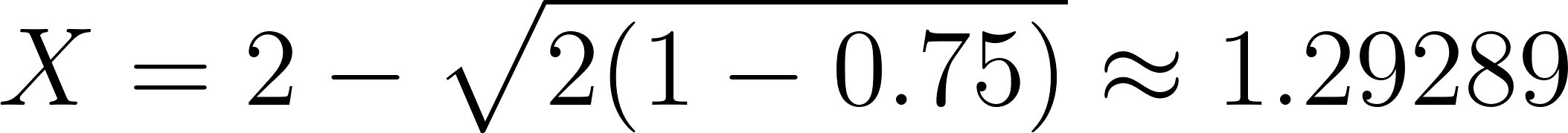
An inverse transform method would involve two cases:

For [](https://www.codecogs.com/eqnedit.php?latex=U%3C1%2F2#0), use [](https://www.codecogs.com/eqnedit.php?latex=X%3D%5Csqrt%7B2U%7D#0)

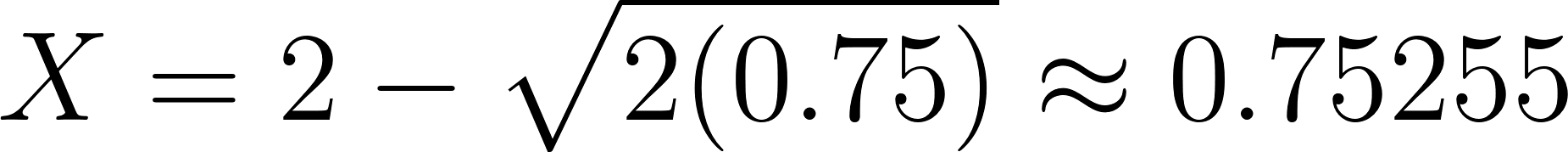
For [](https://www.codecogs.com/eqnedit.php?latex=U%5Cgeq%201%2F2#0), use [](https://www.codecogs.com/eqnedit.php?latex=X%3D2-%5Csqrt%7B2(1-U)%7D#0)

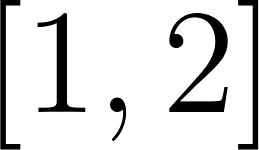
If you try to change that second [](https://www.codecogs.com/eqnedit.php?latex=1-U#0) with [](https://www.codecogs.com/eqnedit.php?latex=U#0) you're going to be using the "wrong" [](https://www.codecogs.com/eqnedit.php?latex=U#0) for that piece of the cumulative distribution function.

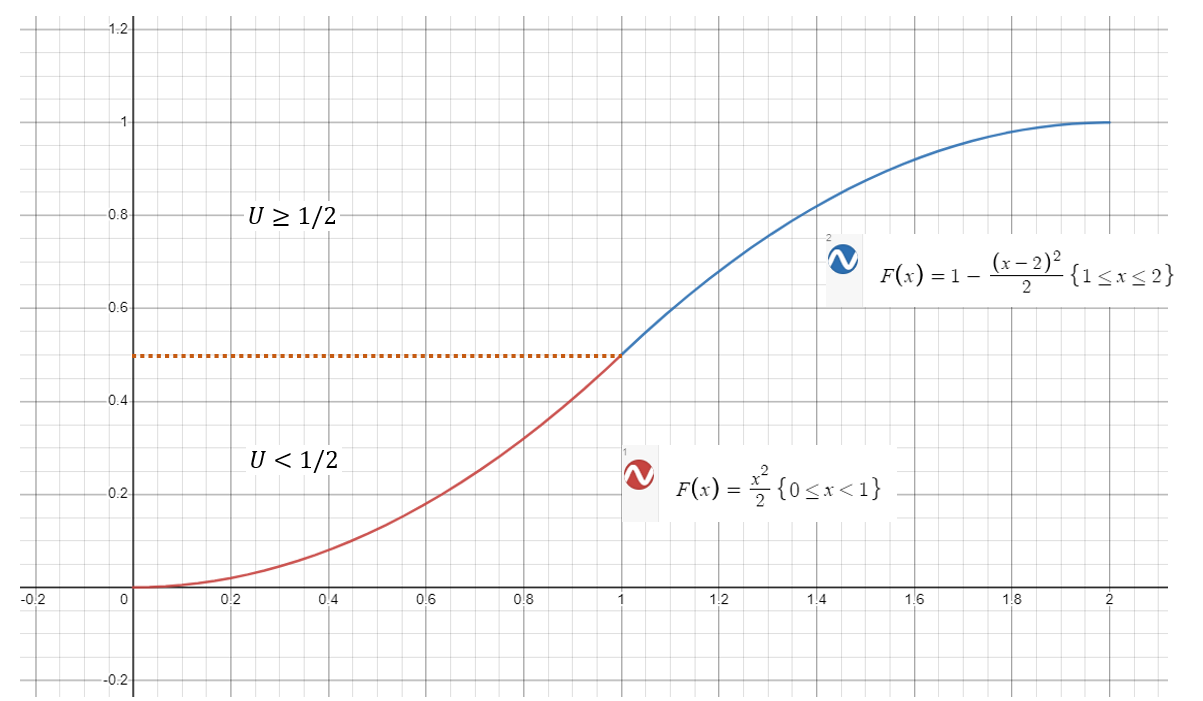
So, let's say that you generated a [](https://www.codecogs.com/eqnedit.php?latex=U#0) of [](https://www.codecogs.com/eqnedit.php?latex=0.75#0) which puts us into the second piece where [](https://www.codecogs.com/eqnedit.php?latex=U%5Cgeq%201%2F2#0).

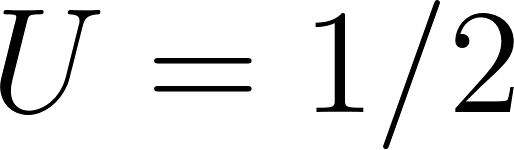
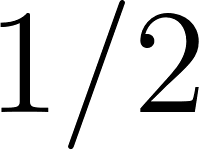
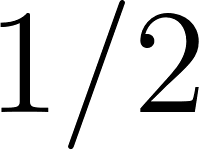
[](https://www.codecogs.com/eqnedit.php?latex=X%3D2-%5Csqrt%7B2(1-0.75)%7D%5Capprox%201.29289#0)

If we were instead replacing [](https://www.codecogs.com/eqnedit.php?latex=1-U#0) with [](https://www.codecogs.com/eqnedit.php?latex=U#0), we would get

[](https://www.codecogs.com/eqnedit.php?latex=X%3D2-%5Csqrt%7B2(0.75)%7D%5Capprox%200.75255#0)

This is problematic because that piece of the function should be returning values in [](https://www.codecogs.com/eqnedit.php?latex=%5B1%2C2%5D#0) so you won't generate the correct distribution by arbitrarily replacing [](https://www.codecogs.com/eqnedit.php?latex=1-U#0) with [](https://www.codecogs.com/eqnedit.php?latex=U#0) here.



Also notice that the "change over" point is at [](https://www.codecogs.com/eqnedit.php?latex=U%3D1%2F2#0). Any value less than [](https://www.codecogs.com/eqnedit.php?latex=1%2F2#0) will "hit" the red piece of the function and any value greater than [](https://www.codecogs.com/eqnedit.php?latex=1%2F2#0) will "hit" the blue portion.

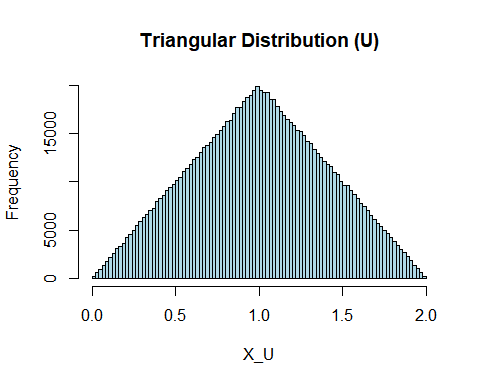
If you look at something like the exponential distribution, it all maps back to the same function, and since [](https://www.codecogs.com/eqnedit.php?latex=U#0) and [](https://www.codecogs.com/eqnedit.php?latex=1-U#0) are both uniform random variables, it doesn't matter. The same situation applies for the geometric distribution as well as many other distributions. You only have to worry about it if you have some "exotic" cumulative distribution function where it matters where it maps back to.

Here you can see it doesn't work.

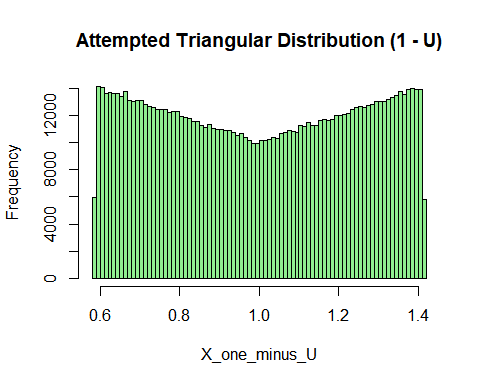
This code generates random samples from a custom continuous distribution using inverse transform sampling with two different approaches for the inverse CDF function. It then plots histograms of the generated samples with titles and colors added.

* Set a seed for random number generation to ensure reproducible results (set.seed(6644)).
* Define the number of random samples to generate (n <- 1e6).
* Generate n uniform random samples between 0 and 1 using the runif() function and store them in the variable U.
* Define the inverse CDF functions for the lower and upper parts of the distribution (lower\_cdf, upper\_cdf).
* Generate random samples from the custom continuous distribution using the inverse CDF functions and store them in the variable X\_U.
* Create a histogram of the X\_U samples.
* Define the inverse CDF functions for the lower and upper parts of the distribution replacing U with 1-U (lower\_cdf\_one\_minus, upper\_cdf\_one\_minus).
* Generate random samples from the custom continuous distribution using the inverse CDF functions and store them in the variable X\_one\_minus\_U.
* Create a histogram of the X\_one\_minus\_U samples.

set.seed(6644)  
n <- 1e6  
U <- runif(n)  
  
lower\_cdf <- **function**(u) sqrt(2\*u)  
upper\_cdf <- **function**(u) 2 - sqrt(2 \* (1 - u))  
  
X\_U <- ifelse(U < 0.5, lower\_cdf(U), upper\_cdf(U))  
hist(X\_U, 100, main = "Triangular Distribution (U)",   
 col = "lightblue", border = "black")



lower\_cdf\_one\_minus <- **function**(u) sqrt(2\*(1-u))  
upper\_cdf\_one\_minus <- **function**(u) 2 - sqrt(2 \* (1 - (1 - u)))  
  
X\_one\_minus\_U <- ifelse(U < 0.5, lower\_cdf\_one\_minus(U), upper\_cdf\_one\_minus(U))  
hist(X\_one\_minus\_U, 100, main = "Attempted Triangular Distribution (1 - U)",   
 col = "lightgreen", border = "black")



# Why (or Why Not) Inverse Transform?

As the person interested in running a simulation, you technically have a choice on how you would generate the random numbers for your simulation.

In the chapter on generating random variates, Law (2015) summarizes some considerations one might have when choosing.

Exactness. If possible, one should use an algorithm that results in random variates with exactly the desired distribution, within the unavoidable external limitations of machine accuracy and exactness of the U(0,1) random-number generator.

Efficient. Prefer algorithms that are efficient in terms of both storage space and execution time. Some algorithms require larger amounts of storage for constants or tables.

Complexity. Can be somewhat subjective; includes both conceptual as well as implementational factors. A potential gain in efficiency could dictate the use of a more complex method.

Other considerations.

Some algorithms rely on random variates from distributions other than U(0,1) which is undesirable for all other things being equal.

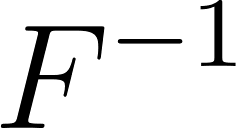
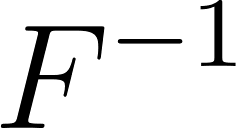
A given algorithm may be efficient for some parameter values but costly for others.

If we want to use variance-reduction techniques such as common random numbers or antithetic random numbers, then these require synchronization of U(0,1) input random variates; this is most easily accomplished for certain types of random-variate generation algorithms (such as inverse transform).

## Advantages of inverse transform

* Is valid for distributions that are mixed (have both continuous and discrete components) as well as for continuous distribution functions with flat spots.
* As mentioned above, facilitates variance-reduction techniques that rely on inducing correlation between random variates (more on this in a later Module)
* Can generate from truncated distributions (in some situations a fitted distribution might provide a good model for observed data generally but other information says that no value can be larger than some finite constant)
* Can be useful for generating order statistics

## Disadvantages of inverse transform

* If we are not able to find a formula for [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D#0) in closed form, then the method might not be possible (although numerical methods that approximate [](https://www.codecogs.com/eqnedit.php?latex=F%5E%7B-1%7D#0) could potentially be used).
* For a given distribution, it may not be the fastest way to generate random variates.

## It Depends

A method such as acceptance-rejection can be useful when more direct methods (e.g., inverse transform) fail or are inefficient. The efficiency of this approach will depend on some choices made (and the ability to effectively employ those choices). This will be covered more in Module 7. Ultimately, we would hope to minimize the probability of a rejection here and accept as many as possible so we are not generating too many random variates that we ultimately do not get to use whereas inverse transform lets us use all of them.

As with many things in the sphere of analytics, there isn't always just one "right answer" of doing things. It will depend on your particular use case and constraints.