Quantitative Understanding in Biology Module I: Statistics
Lecture II: Probability Density Functions and the Normal Distribution

The Binomial Distribution

Consider a series of N repeated, independent yes/no experiments (these are known as Bernoulli trials), each of which has a probability p of being 'successful'. The binomial distribution gives the probability of observing exactly k successes.

The `dbinom` function in R will compute this probability for you: `dbinom(k, n, p)`

Note that the binomial distribution is a discrete distribution. That is, it only makes sense for integer values of k. You can't ask: what is the probability of observing 4.3 heads in ten coin tosses. Also note that the binomial distribution has two parameters: n and p.

We can plot the probabilities of observing k heads in ten flips of a fair coin:

```
> x <- 0:10
> barplot(dbinom(x,size=10,prob=0.5), names.arg = x)
```

Note that we have chosen to represent the probability density function (PDF) with a bar plot, and not a line plot. This emphasizes the discrete nature of the probability density function.

In the literature, you will often see reference to 'successful' and 'unsuccessful' Bernoulli trials. This implies that one outcome is worse than the other, which is not always the case. In the case of heads vs. tails, or blue vs. brown eyes, it is not clear which outcome is considered a 'success'. While most would consider death to be the 'unsuccessful' outcome, in the case of, say, cancer treatment, cell death (of cancer cells) would be considered successful. The bottom line is to be sure you convey (or understand) what you mean by a successful outcome.

We can also compute and plot cumulative probabilities. This is useful for answering questions such as: What is the probability of observing more than seven heads in ten tosses of a fair coin?

The above question is answered by the evaluation…

```
> 1 - pbinom(7,size=10,prob=0.5)
[1] 0.0546875
```

To see why the above command gives the answer we are looking for, consider that `pbinom(7,10,0.5)` yields the probability of observing 0…7 heads. Subtracting that from one thus gives the probability of all other cases; i.e., 8…10 heads.

A plot of the cumulative density function is generated by:

```
> barplot(pbinom(x,10,0.5), names.arg = x)
```

Note that, by default, the `pbinom` function reports the probability for k ≤ x (the first argument). When working with discrete density functions, be careful about off-by-one errors. You can verify your answer for the above problem by summing the PDF:

```
> sum(dbinom(8:10,10,0.5))
[1] 0.0546875
```

We can also use R to generate random values that follow a distribution:

```
> rbinom(50,10,0.5)
 [1] 4 6 5 3 4 5 5 6 5 8 4 5 6 4 4 5 4 6 5 2 6 5 6 6 6
[26] 7 5 6 5 7 7 4 6 6 7 5 6 4 7 7 7 5 5 5 6 5 6 6 5 5
> hist(rbinom(1000000, 10, 0.5), breaks=seq(-0.5, 10.5, by=1))
```

Note how the binning was specified.

A histogram of a large quantity of random data from a distribution will look a lot like a plot of the density function from that distribution, except that the density function is normalized.

The above distribution roughly resembles a normal distribution. This is not always the case for a binomial distribution:

```
> x <- 0:10; barplot(dbinom(x, 10, 0.2), names.arg = x)
```

The peak for this distribution is not surprising; there are 10 trials with each having a 20% chance of success, so we expect a peak at around 2.

Consider 100 trials, with each having a probability of success of 2%. We still expect a peak at around 2.

```
> x <- 0:10; barplot(dbinom(x, 100, 0.02), names.arg = x)
```

Note that we choose to only plot the first eleven bars (k = 0…10). We could plot out to 100, but, in this case, the probably of observing more than 10 successes is quite small. In fact, we know how to compute this:

```
> 1-pbinom(10,100,0.02)
[1] 5.646028e-06
```


The Poisson Distribution

Let's plot the binomial distribution where N increases while N*p is constant. We'll learn a little more about plotting in R as we go. If you are using R Studio, this is a great time to open up an R Script window…

Attempt #1:

```
> x <- 0:10
> par(mfrow=c(1,4))
> barplot(dbinom(x, 10, 0.2), names.arg = x)
> barplot(dbinom(x, 100, 0.02), names.arg = x)
> barplot(dbinom(x, 1000, 0.002), names.arg = x)
> barplot(dbinom(x, 10000, 0.0002), names.arg = x)
```

Nice! We can make a panel of plots with R; this is a useful way to compare plots. But in this case the axes are not uniform, and the plots should be labeled.

Attempt #2 (the `par` settings are still in effect):

```
> n <- 10; barplot(dbinom(x, n, 2/n), names.arg=x, ylim=c(0,0.35), main=paste("n = ", n))
> n <- 100; barplot(dbinom(x, n, 2/n), names.arg=x, ylim=c(0,0.35), main=paste("n = ", n))
> n <- 1000; barplot(dbinom(x, n, 2/n), names.arg=x, ylim=c(0,0.35), main=paste("n = ", n))
> n <- 10000; barplot(dbinom(x, n, 2/n), names.arg=x, ylim=c(0,0.35), main=paste("n = ", n))
```

Computers are good at repetitive tasks; let's get R to do the repetitive work (again, `par` is in effect).

Attempt #3

```
> for (n in c(10,100,1000,10000)) {
+    barplot(dbinom(x, n, 2/n), names.arg=x, ylim=c(0,0.35), main=paste("n = ", n))
+ }
```

Finally, we reset our plotting parameter so we get single plots instead of panels.

```
> par(mfrow=c(1,1))
```

We can observe that as N increases, the distribution converges. This is, in fact, the Poisson distribution. It has only one parameter, often denoted in the literature as λ, which is the average number of successes observed in a particular interval of interest.

The Poisson distribution is very useful. It is used to model count data per unit time (or length, or area, or whatever). Once you have measured λ, you can use the Poisson distribution to compute the probability of observing a specific number of events in a given interval. You don't ever have to know what the underlying probability of your event is, or how many times your event of interest did not happen.

The Poisson distribution is sometimes referred to as the distribution of "rare events". This is done because p (the probability of a single Bernoulli trial being successful) is usually very, very small. This is

counterbalanced by the fact that there are many, many opportunities for that event to happen, so, overall, the event is not that rare.

Some examples of processes modeled by the Poisson distribution are: ticks of a radiation counter, mutations of a DNA sequence, lightning strikes, and neuronal spike trains.

Example: An average of four babies are born each night on a particular maternity ward. (A) What is the probability that the intern on call will be able to sleep through the night? (B) What is the probability that the intern will be swamped (defined as more than six births to be attended to)?

For part (A), we simply evaluate the Poisson distribution for the case where k = 0, and λ = 4:

```
> dpois(0,4)
[1] 0.01831564
```

For part (B), we will use the cumulative distribution function:

```
> 1 - ppois(6,4)
[1] 0.1106740
```

With a 1.8% chance of getting a good night's sleep and an 11% chance of getting swamped, things don't look too good for our intern.

Exercise: Add one more plot to the panel you made above, showing the Poisson distribution for λ=2.

Note that, like the binomial distribution, the Poission distribution is also discrete. It does not make sense to ask what the probability of 3.7 births on the ward is. However, λ does not need to be an integer; it is fine if the average number of births on the ward is 4.1 per night (our intern's life is just a little bit tougher then).

As you have probably figured out by now, R has a set of functions for each standard probability distribution. Functions prefixed by 'd' report the probability density function; those prefixed with a 'p' report the cumulative distribution function, and those with an 'r' generate random samples from the underlying distribution:

```
> rpois(50,0.25)
 [1] 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 2 0 1 0
[26] 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

The Geometric Distribution

The geometric distribution models the waiting time between successes of Bernoulli trials, each with probability p. They tend to have long tails.

```
> x <- 0:20
```



```
> barplot(dgeom(x,0.25), names.arg=x)
```

Again, we also have a cumulative density function…

```
> barplot(pgeom(x,0.25), names.arg=x)
```

…and a means of generating random values from the distribution:

```
> rgeom(10, 0.25)
 [1] 6 4 8 0 1 0 2 3 1 3
```

Note that the distribution is defined as the number of (discrete) trials **between** successful outcomes. This implies that the distribution is defined for x=0. In fact, the value of the distribution at x=0 should be obvious…

In addition to the 'd', 'p', and 'r' functions, R has a quantile function for each distribution. We saw this in an earlier section for the normal distribution.

Example: Assuming two heterozygotic parents and classical Mendelian inheritance, how many offspring would you have to breed to be 95% sure that at least one will express a recessive phenotype?

The answer is (to me, at least) surprisingly high…

```
> qgeom(0.95, 0.25) + 1
[1] 11
```

We need to add one in the computation above because waiting for zero breeding periods implies producing one offspring. The relatively high answer is not that unexpected when you recall that this distribution does have long tails.

There is another way to reason this problem, and verify the result. We know that the probably of each offspring not expressing the recessive phenotype is 0.75. So, for example, the probability of not have any offspring that express the desired recessive phenotype if two are bred is $0.75^2 = 0.5625$. For three offspring, the result is $0.75^3 = 0.4218$. To solve our problem, we need to increase our exponent until the resulting probability is less than 1-95%=0.05.

```
> x <- 0:15; cbind(x, y=0.75^x)
       x          y
 [1,]  0 1.00000000
 [2,]  1 0.75000000
 [3,]  2 0.56250000
 [4,]  3 0.42187500
 [5,]  4 0.31640625
 [6,]  5 0.23730469
 [7,]  6 0.17797852
 [8,]  7 0.13348389
 [9,]  8 0.10011292
```



```
[10,]  9 0.07508469
[11,] 10 0.05631351
[12,] 11 0.04223514
[13,] 12 0.03167635
[14,] 13 0.02375726
[15,] 14 0.01781795
[16,] 15 0.01336346
```

The Uniform Distribution

This is the first continuous distribution we consider in this section. It has two parameters, a minimum and a maximum of a range. The idea behind the uniform distribution is trivial: any value (not just integers) between the minimum and maximum is equally likely to be found in a sample from this population.

```
> curve(dunif(x, 0, 10), -0.3, 10.3, ylim=c(-0.005, 0.105))
```

A plot of the probability density function for the uniform distribution looks like a flat plateau. Note that the PDF is normalized, so that the area under the curve is unity. Note that the plot that R produces is an approximation; the edges of the plateau should be perfectly vertical.

Observe that for continuous distribution we plot curves, not bars.

Also note that the probability of observing a particular value from a continuous distribution is essentially zero. The chances of observing, say, exactly 5.000000000000000000000000 (with infinitely many zeros) from the above distribution is nil. However, you can compute the probability of observing a number within a range, say between 4.99 and 5.01. This is the area under the density curve between these limits, or, alternatively, the integral of the probably density function from the lower limit to the upper limit.

The Exponential Distribution

The exponential distribution is a continuous form of the geometric distribution. It has one parameter, which is the reciprocal of the expected value (i.e., the average rate of the event). This is usually written as $1/\lambda$. Consider a neuron that spikes, on average, at 10Hz.

```
> par(mfrow=c(2,1))
> curve(dexp(x, 10), 0, 0.5)
> curve(pexp(x, 10), 0, 0.5)
```

Like its discrete counterpart, the exponential distribution is used to model waiting times. While the geometric distribution models time as discrete steps, or attempts of a Bernoulli trial, the exponential distribution models waiting times as a continuous variable.

Note that 'time' in this context can be any interval between events, and is sometimes actually a length or other variable (for example, your event could be passing a red house along a road). You will

occasionally see the distance between mutated nucleotides along a chromosome modeled with an exponential distribution. Very strictly speaking, this should be modeled with a geometric distribution, since the positions are discrete. The exponential distribution is not a bad approximation for this.

As with all of the distributions we have surveyed that are derived from Bernoulli trials, there is an important assumption that the trials are independent. From the description above, one might naïvely imagine that you could model wake/sleep intervals as an exponential distribution (we are waiting a continuous amount of time for an event, falling asleep, to take place). However, this is probably not a good model, because we tend not to fall asleep just after getting up; the events are not independent. (This might, however, be a good model for wake/sleep intervals for cats!)

The Normal Distribution

Almost everyone is familiar with the normal distribution; it is the famous bell curve. Formally, it has two parameters, the mean and the standard deviation.

```
> curve(dnorm(x, 0, 1), -4, 4)
```

One very important result, and one of the few numerical facts that should be memorized, is that 95% of the area under the normal probability distribution function lies within the interval $\bar{x}$ ± 1.96 SD.

Using R, we can add lines to our plot above to demarcate this region:

```
> lines(c(-1.96, -1.96), c(0, dnorm(-1.96)))
> lines(c(+1.96, +1.96), c(0, dnorm(+1.96)))
```

In the lines functions above, the first vector is a list of x-coordinates; the second is a list of y-coordinates.

We can also produce plots with shaded regions:

```
> x <- seq(-4, 4, 0.01)
> y <- dnorm(x)
> plot(x,y,type="l")
> polygon(c(-1.96, x[x >= -1.96 & x <= 1.96], 1.96),
+         c(0,     y[x >= -1.96 & x <= 1.96], 0),
+         col="red")
```

We can also recover the 95% interval using the cumulative distribution function:

```
> pnorm(1.96) - pnorm(-1.96)
[1] 0.9500042
```

And, of course, we have our quantile function. You should be able to trivially predict the results of these:

```
> qnorm(0.5); qnorm(0.025); qnorm(0.975)
```


Statistical Tests, the Normal Distribution, and the Central Limit Theorem

Many statistical tests are derived under the assumption that the data follow a normal distribution.

Most biological data does not strictly follow a normal distribution. In particular, to be strictly true, you must be able to admit, albeit at very low probabilities, any real value, including arbitrarily large positive and negative numbers. Also, any discrete distribution cannot, by definition, be normal.

The central limit theorem, a cornerstone of statistical theory, resolves this apparent conflict. It states that the distribution of estimated means of a population based on repeated, finite samples, will approximate a normal distribution. The amazing thing about the central limit theorem is that this works even if the underlying distribution is not normal. In fact, even if it is pathologically not normal, the central limit theorem still holds. Since much statistical testing involves comparing means (we will spend the next two sessions doing this), we have some hope of our statistics being valid.

Another way of thinking about the central limit theorem is that it tells us that the SEM is a valid indication of how well we know the true mean of the underlying population, even if the sample mean and SD are not a good representation of the distribution of the population.

It also turns out that a measurement subject to noise from several random sources (of approximately the same magnitude) will approximate a normal distribution, even if the sources of the underlying noise are not normal.

If your data are not normally distributed, you have a few choices:

- Use statistical tests anyway; many are quite robust and work well, even for moderately non-normal distributions of data. However, some are not, and you may want to research the robustness of a particular method before using it.
- Transform your data so they are normally distributed. Lograthmic transformations are often helpful (especially when dealing with sizes of organisms or colonies). You may find some justification for doing this by considering the underlying physical process. Do be careful to take due account of units of measure if you do this, as it is mathematically incorrect to take a logarithm of a quantity that is not dimensionless.
- Use a non-parametric test. These are usually based on rank-order of data, rather than the actual numerical values themselves. Non-parametric tests are usually substantially less powerful than their parametric counterparts. This means that you have a higher chance of your analysis being inconclusive, and may need to increase N in your experimental design to get good results.
- Use tests specifically designed for a particular non-normal distribution. These can be difficult to find and apply correctly. We'll see one common example in the next session, but, in general, it may be wise to work with a professional statistician if you need to go down this road.

Assessing Normality

Since most statistical tests can be applied if the data are 'close to normal', we need a way of assessing this. The waters are somewhat murky, but there are a few options:

- You can plot histograms and density functions and visually inspect them.

We have seen some problems with histograms in an earlier session. The arbitrariness in the binning makes them more than occasionally misleading. Adding a density plot can help, but there is still another underlying problem: our brains are not that good at distinguishing a normal distribution (one that follows a Gaussian curve) from other generally bell shaped distributions.

- You can perform formal statistical tests for normality.

This is less useful than you would initially expect. They are often inconclusive for small N, as there is simply not enough information to make any kind of determination. Furthermore, for large N, the test usually concludes that the distribution of the data is not normal (because most biological data actually are not), even when it is close enough to normal for other tests to work.

These tests can be a useful tool, however, in automated pipelines when you can't practically inspect all your data and you need to find a few needles in a haystack.

- You can evaluate normality using QQ plots.

This is a favorite of statisticians. QQ plots (or quantile-quantile plots) transform your data in such a way that if your data are normally distributed, the points will fall close to a straight line. Since our visual system is very good at assessing linearity, with a little training, we can often perform better at distinguishing normal from non-normal distributions using QQ plots than with histograms or density functions.

Formal Tests for Normality

R provides two functions that can be used to test for normality. `shapiro.test` and `ks.test` perform the Shapiro test and the Kolmogorov-Smirnov tests, respectively. Neither one is considered to be that good any more, although the Kolmogorov-Smirnov test used to be common, and many people are familiar with it. You may want to download and install the '`nortest`' package for some more modern options.

For our purposes, QQ plots (described below) are a better means of assessing normality, but we will take this opportunity to have a first look at formal statistical hypothesis testing. We'll cover the principles in more detail in later sessions, so just consider this a light introduction.

Formal statistical testing begins by formulating a null hypothesis; this is often written as H_0. In our case:

> *H_0: The data being tested were sampled from a normally distributed population.*

Next, we compute a p-value. The p-value answers this question (and only this question):

> *Assuming that the null hypothesis is true, what is the probability of seeing a distribution as far or further from normality as your sampled test data?*

If the p-value is small (say < 0.05 for 95% certainty), then we reason that the null hypothesis is unlikely to be true. This is a qualified form of *reductio ad absurdum*.

If the p-value is not low we can't conclude anything about the null hypothesis. The test is inconclusive. The data could have come from a normal distribution, or the null hypothesis could have been false. We have no idea. In other words, these tests can only rule out normality, but can never confirm it.

To perform the Kolmogorov-Smirnov test in R…

```
> x <- rnorm(100)
> ks.test(x, "pnorm", mean=mean(x), sd=sd(x))

        One-sample Kolmogorov-Smirnov test

data:  rnorm(100)
D = 0.062, p-value = 0.8367
alternative hypothesis: two-sided

> x <- rexp(100)
> ks.test(x, "pnorm", mean=mean(x), sd=sd(x))

        One-sample Kolmogorov-Smirnov test

data:  rexp(100)
D = 0.1711, p-value = 0.005741
alternative hypothesis: two-sided
```

For now, just look at the p-values. Note that `ks.test` can be used to test against other distributions.

When we run the KS test in these examples, it is important to understand that we are testing against a specific distribution with a specified mean and SD. You can, for example, generate data from a normal distribution with a mean of 0.1, and then test is against the default, standard distribution where the mean is zero and the SD is unity. As we may expect, for small N it is difficult to rule out the null hypothesis, whereas for large N, it is possible:

```
> ks.test(rnorm(20, mean=0.1), "pnorm")
p-value = 0.384

> ks.test(rnorm(100000, mean=0.1), "pnorm")
p-value < 2.2e-16
```


QQ Plots

Consider a sample of 100 values. We begin by computing and plotting a quantile for each value in the distribution.

```
> x <- rnorm(100)
> p <- ((1:100) - 0.5)/100
> p
  [1] 0.005 0.015 0.025 0.035 0.045 0.055 0.065 0.075 0.085 0.095 0.105 0.115
 [13] 0.125 0.135 0.145 0.155 0.165 0.175 0.185 0.195 0.205 0.215 0.225 0.235
 [25] 0.245 0.255 0.265 0.275 0.285 0.295 0.305 0.315 0.325 0.335 0.345 0.355
 [37] 0.365 0.375 0.385 0.395 0.405 0.415 0.425 0.435 0.445 0.455 0.465 0.475
 [49] 0.485 0.495 0.505 0.515 0.525 0.535 0.545 0.555 0.565 0.575 0.585 0.595
 [61] 0.605 0.615 0.625 0.635 0.645 0.655 0.665 0.675 0.685 0.695 0.705 0.715
 [73] 0.725 0.735 0.745 0.755 0.765 0.775 0.785 0.795 0.805 0.815 0.825 0.835
 [85] 0.845 0.855 0.865 0.875 0.885 0.895 0.905 0.915 0.925 0.935 0.945 0.955
 [97] 0.965 0.975 0.985 0.995
> q <- qnorm(p)
> plot(q)
```

This sigmoid shape is characteristic of the Gaussian distribution. Reading from the plot, this says that, if the data we have is indeed sampled from a normal distribution, we expect the smallest value to be about -2.6. We can see this numerically as well.

```
> q[1]
[1] -2.575829
```

Similarly, the 20th value (in rank order) should be around -0.85. Etc.

Now, if we plot the expected values against the actual values, we expect a more or less straight line.

```
> plot(q, sort(x))
```

If your data are from a normal distribution, but with a different mean and/or SD, you will get a straight line with a different slope or intercept. If your data are not from a normal distribution, you will observe systematic departures from a straight line.

Especially for non-large N, due to sampling variability, you will see some squiggle in the data, even if it is sampled from a normal distribution. A common technique for assessing normality is to prepare a panel of eight or ten QQ plots. One is plotted with your data, and the others with random samples from the normal distribtion of the same size as your data set. If the panel with your data stands out, you can conclude that it is probably not normal.

Note that QQ plots are not limited to testing for normality. You can test against any postulated distribution, and can even plot two distributions against each other.

If you are testing for normality, R has two built-in functions that streamline the process:


```
> qqnorm(x); qqline(x)
```

