Lab 2: probability distributions, averaging, and Jensen's inequality

September 26, 2010

1 Random variables/probability distributions in R

R knows about lots of probability distributions. For each, it can generate random numbers drawn from the distribution ("deviates"); compute the cumulative distribution function and the probability distribution function; and compute the quantile function, which gives the x value such that $\int_0^x P(x)\,dx$ (area under the curve from 0 to x) is a specified value, such as 0.95 (think about "tail areas" from standard statistics). For example, you can obtain the critical values of the standard normal distribution, ± 1.96, with `qnorm(c(0.025,0.975))`.

Let's take the binomial distribution (yet again) as an example.

First, use `set.seed()` for consistency:

```
> set.seed(1001)
```

- `rbinom(n,size,prob)` gives `n` random draws from the binomial distribution with parameters `size` (trials per draw) and `prob` (probability of success on each draw). You can give different parameters for each draw. For example:

```
> rbinom(10,size=8,prob=0.5)
 [1] 7 4 4 4 4 6 1 2 3 5
> ## specify probabilities of 0.2, 0.4, 0.6 for successive draws
> rbinom(3,size=8,prob=c(0.2,0.4,0.6))
[1] 1 2 3
```

Now plot the result of drawing 200 values from a binomial distribution with $N = 12$ and $p = 0.1$ and plotting the results as a `factor` (with 200 draws we don't have to worry about any of the 13 possible outcomes getting missed and excluded from the plot):

```
> z  <-  rbinom(200,size=12,prob=0.1)
> plot(factor(z),xlab="# of successes",ylab="# of trials out of 200")
```

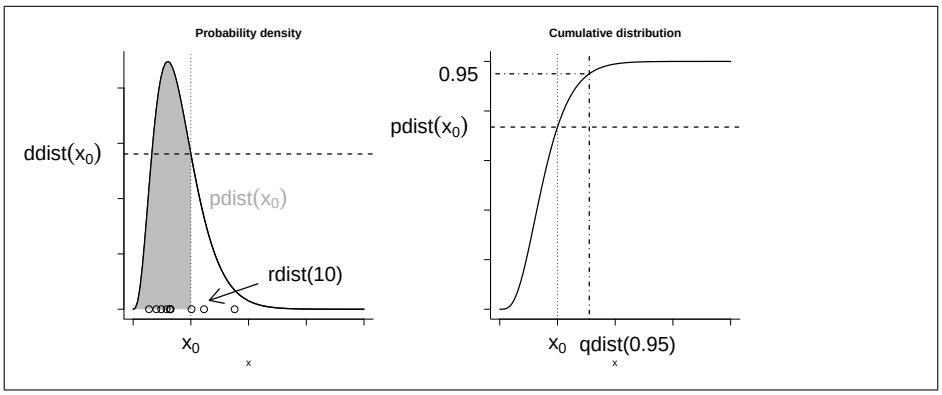


Figure 1: R functions for an arbitrary distribution `dist`, showing density function (`ddist`), cumulative distribution function (`pdist`), quantile function (`qdist`), and random-deviate function (`rdist`).

(Saying `plot(factor(z,...))` gives a nice barplot. What do you get if you just say `plot(z)`?)

- `dbinom(x,size,prob)` gives the value of the probability distribution function (pdf) at `x` (for a continous distribution, the analogous function would compute the probability density function). Since the binomial is discrete, `x` has to be an integer, and the pdf is just the probability of getting that many successes; if you try `dbinom` with a non-integer `x`, you'll get zero and a warning.

- `pbinom(q,size,prob)` gives the value of the cumulative distribution function (cdf) at `q` (e.g. `pbinom(7,size=10,prob=0.4)`);

- `qbinom(p,size,prob)` gives the quantile function $x = q(p)$, where `p` is a number between 0 and 1 (an area under the pdf, or value of the cdf) and x is the value such that $P(X \leq x) = p$. The *quantile function* Q is the inverse of the cumulative distribution function C: if $Q(p) = q$ then $C(q) = p$. Example: `qbinom(0.95,size=10,prob=0.4)`.

These four functions exist for each of the distributions R has built in: e.g. for the normal distribution they're `rnorm()`, `pnorm()`, `dnorm()`, `qnorm()`. Each distribution has its own set of parameters (so e.g. `pnorm(x)` is `pnorm(x,mean=0,sd=1)`).

***Exercise 1**: For the binomial distribution with 10 trials and a success probability of 0.2:

- Pick 8 random values and sort them into increasing order (if you `set.seed(1001)` beforehand, you should get $X = 0$ (twice), $X = 2$ (4 times), and $X = 4$ and $X = 5$ (once each)).
- Calculate the probabilities of getting exactly 3, exactly 4, or exactly 5 successes.
- Calculate the probability of getting 5 or more successes. (Note that `pbinom(x,...)` gives the probability. of getting *less than or equal to* `x` successes. Using `lower.tail=FALSE` gives the probability of getting *greater than* `x` successes.)

You can use the R functions to test your understanding of a distribution and make sure that random draws match up with the theoretical distributions as they should. This procedure is particularly valuable when you're developing new probability distributions by combining simpler ones, e.g. by zero-inflating or compounding distributions.

The results of a large number of random draws should have approximately the correct moments (mean and variance), and a histogram of those random draws (with `freq=FALSE` or `prob=TRUE`) should match up with the theoretical distribution. For example, draws from a binomial distribution with $p = 0.2$ and $N = 20$ should have a mean of approximately $Np = 4$ and a variance of $Np(1-p) = 3.2$:

```
> set.seed(1001)
> N <- 20; p <- 0.2
> x  <-  rbinom(10000,prob=p,size=N)
> c(mean(x),var(x))
```

```
[1] 4.001200 3.144913
```

The mean is very close, the variance is a little bit farther off. We can use the `replicate()` function to re-do this command many times and see how close we get:

```
> var_dist  <-  replicate(1000,var(rbinom(10000,prob=p,size=N)))
```

(this may take a little while; if it takes too long, lower the number of replicates to 100).

Looking at the summary statistics and at the 2.5% and 97.5% quantiles of the distribution of variances:

```
> summary(var_dist)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.052   3.169   3.199   3.199   3.229   3.340
```

```
> quantile(var_dist,c(0.025,0.975))

    2.5%    97.5%
3.114357 3.285333
```

(Try plotting a histogram (`hist(var_dist)`) or a kernel density estimate (`plot(density(var_dist))`) of these replicates.) Even though there's some variation (of the variance) around the theoretical value, we seem to be doing the right thing since the theoretical value ($Np(1-p)$) falls within the central 95% of the empirical distribution.

Finally, we can compare the observed and theoretical quantiles of our original sample:

```
> quantile(x,c(0.025,0.975))

 2.5% 97.5%
    1     8

> qbinom(c(0.025,0.975),prob=p,size=N)

[1] 1 8
```

Figure 2 shows the entire simulated frequency distribution along with the theoretical values. The steps in R are:

1. pick 10,000 random deviates:

   ```
   > x  <-  rbinom(10000,prob=p,size=N)
   ```

2. Tabulate the values, and divide by the number of samples to get a probability distribution:

   ```
   > tx <- table(factor(x,levels=0:12))/10000
   ```

   (The `levels` command is necessary in this case because the probability of $x = 12$ with $p = 0.2$ and $N = 12$ is actually so low ($\approx 4 \times 10^{-9}$) that it's likely that a sample of 10,000 won't include any samples with 12 successes.)

3. Draw a barplot of the values, extending the y-limits a bit to make room for the theoretical values and saving the x locations at which the bars are drawn:

   ```
   > b1  <-  barplot(tx,ylim=c(0,0.23),ylab="Probability")
   ```

4. Add the theoretical values, plotting them at the same x-locations as the centers of the bars:

   ```
   > points(b1,dbinom(0:12,prob=p,size=N),pch=16)
   ```

   (`barplot()` doesn't put the bars at x locations corresponding to their numerical values, so you have to save those values as `b1` and re-use them to make sure the theoretical values end up in the right place.)

Here are a few alternative ways to do this plot: try at least one for yourself.

1. `> plot(factor(x)); points(b1,10000*dbinom(0:12,prob=p,size=N))`

   (plots the number of observations without rescaling and scales the probability distribution to match);

2. `> plot(table(x)/10000); points(0:12,dbinom(0:12,prob=p,size=N))`

   Plotting a table does a plot with `type="h"` (high density), which plots a vertical line for each value. I think it's not quite as pretty as the barplot, but it works. (Don't forget to use 0:12 for the x axis values — if you don't specify x the points will end up plotted at x locations 1:13.) Unlike factors, tables can be scaled numerically, and the lines end up at the right numerical locations, so we can just use `0:12` as the x locations for the theoretical values.

3. You could also draw a histogram: since histograms were really designed continuous data you have to make sure the breaks occur in the right place (halfway between counts):

   ```
   > h <-  hist(x,breaks=seq(-0.5,12.5,by=1),col="gray",
         prob=TRUE)
   > points(0:12,dbinom(0:12,prob=p,size=N))
   ```

1.1 Continuous distribution: lognormal

Doing the equivalent plot for continuous distributions is actually somewhat easier, since you don't have to deal with the complications of a discrete distribution: just use `hist(...,prob=TRUE)` to show the sampled distribution (possibly with `ylim` adjusted for the maximum of the theoretical density distribution) and `ddist(x,[parameters],add=TRUE)` to add the theoretical curve.

Let's go through the same exercise for the lognormal, a continuous distribution:

```
> z <- rlnorm(1000,meanlog=2,sdlog=1)
```

Plot the results:

```
> hist(z,breaks=100,freq=FALSE,col="gray",
      ylim=c(0,0.09))
> lines(density(z,from=0),lwd=2,col=2)
```

Add a parametric estimate of the curve, based on the observed mean and standard deviation of the log-values:

```
> obsmeanlog <- mean(log(z))
> obssdlog <- sd(log(z))
> curve(dlnorm(x,meanlog=obsmeanlog,sdlog=obssdlog),add=TRUE,lwd=2,from=0,
       col="blue",n=200)
```

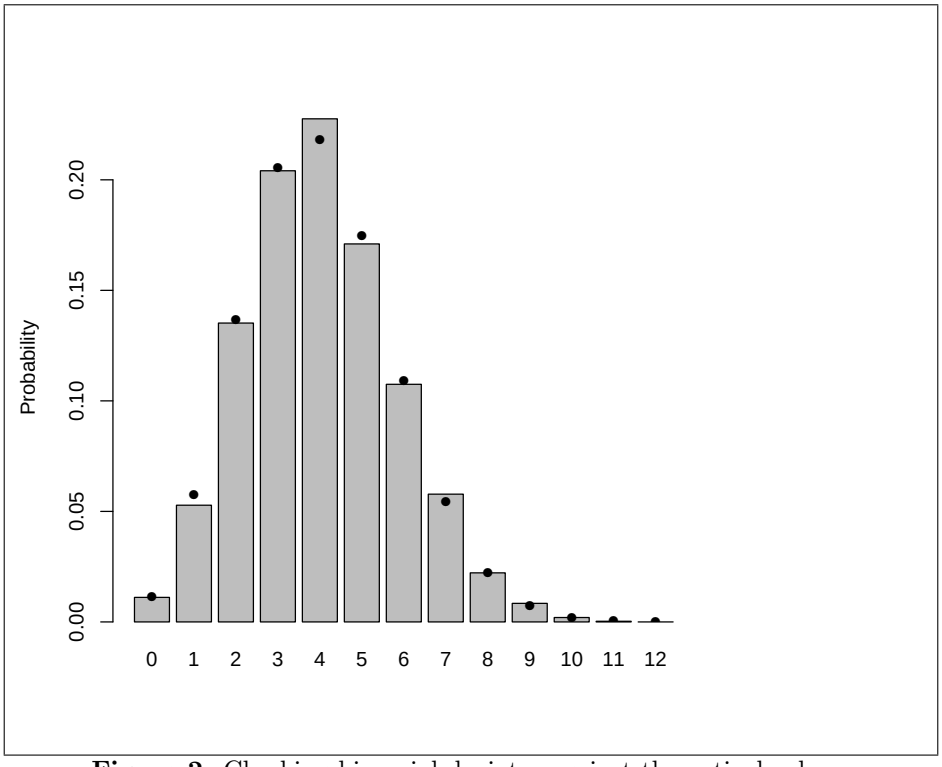


Figure 2: Checking binomial deviates against theoretical values.

Note that the maximum height of the parametric curve is higher than the maximum height of the density estimator: this is generally true, since the density estimator "smooths" (smears out) the data.

The probability of $x > 20$, and the 95% confidence limits:

```
> plnorm(20,meanlog=2,sdlog=1,lower.tail=FALSE)

[1] 0.1596901

> qlnorm(c(0.025,0.975),meanlog=2,sdlog=1)

[1]  1.040848 52.455437
```

Comparing the theoretical values of the mean and variance with the observed values for this random sample (see `?rlnorm` for theoretical values):

```
> meanlog <- 2
> sdlog <- 1
> ## expected/observed means
> c(exp(meanlog+sdlog^2/2),mean(z))

[1] 12.18249 12.22854

> ## expected/observed variances
> c(exp(2*meanlog+sdlog^2)*(exp(sdlog^2)-1),var(z))

[1] 255.0156 266.1457

> ## expected/observed 95th percentile
> c(qlnorm(0.95,meanlog=meanlog,sdlog=sdlog),quantile(z,0.95))

              95%
38.27717 39.89901
```

There is a fairly large difference between the expected and observed variance. This is typical: variances of random samples have larger variances, or absolute differences from their theoretical expected values, than means of random samples.

Sometimes it's easier to deal with log-normal data by taking the logarithm of the data and comparing them to the normal distribution:

```
> hist(log(z),freq=FALSE,breaks=100,col="gray")
> curve(dnorm(x,mean=meanlog,sd=sdlog),add=TRUE,lwd=2)
> lines(density(log(z)),col="red",lwd=2)
```

***Exercise 2**: Use `rnbinom()` to pick 10,000 negative binomial deviates with $\mu = 2$, $k = 0.5$ Draw a plot of the distribution. Check that the mean, variance, and quantiles agree reasonably well with the theoretical values. Add points representing the theoretical distribution to the plot. Because (unlike the binomial) the negative binomial has no theoretical maximum value, you will have to pick an maximum to plot (e.g. by finding the maximum value in the observed data set). When you plot, don't forget to fill in zeros for values that don't occur in the sample by using `factor(x,levels=0:max(x))`.
Extra credit: Now translate the μ and k parameters into their p and n equivalents (the coin-flipping derivation of the negative binomial: see `?rnbinom`). Use these parameters as `size` and `prob` (rather than `size` and `mu`), compute the probabilities of each value, and add those points to the plot [use a different plotting symbol to make sure you can see that they are exactly the same as the theoretical values based on the $\{\mu, k\}$ parameterization].

2 The method of moments: reparameterizing distributions

In class, I showed how to use the *method of moments* to estimate the parameters of a distribution by setting the sample mean and variance ($\bar{x}$, s^2) equal to the theoretical mean and variance of a distribution and solving for the parameters. For the negative binomial, in particular, we start by setting the sample moments (sample mean and variance) equal to their theoretical values:

$$\bar{x} = \mu$$
$$s^2 = \mu(1 + \mu/k)$$

We solve the second equation for k (the first is already "solved" for μ) to get method-of-moments estimates $\mu = \bar{x}$ and $k = (\bar{x})/(s^2/\bar{x} - 1)$.

You can also define your own functions that use your own parameterizations: call them `my_function` rather than just replacing the standard R functions (which will lead to insanity in the long run).

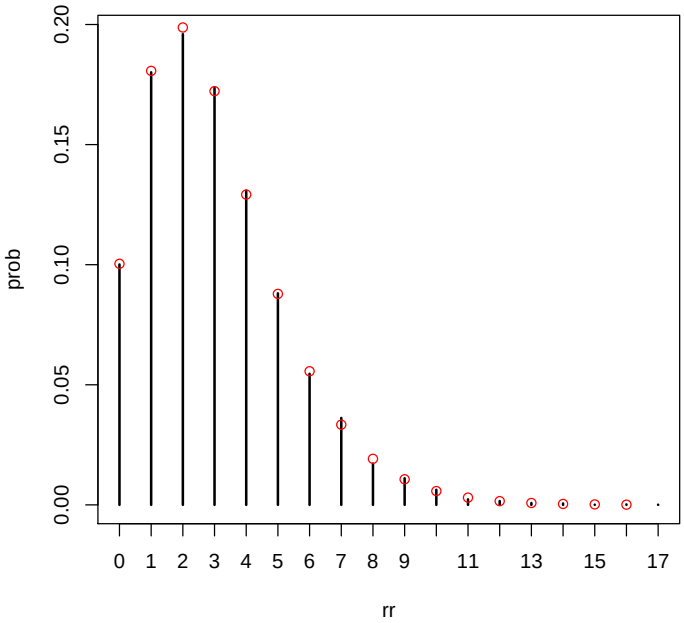
For example, defining

```
> my_dnbinom = function(x,mean,var,...) {
   ## take mean and var, use them to get
   ##   original parameters ...
   mu = mean
   k = mean/(var/mean-1)
   ## feed original parameters to the original function
   dnbinom(x,mu=mu,size=k,...)
 }
> my_rnbinom = function(n,mean,var,...) {
   mu = mean
   k = mean/(var/mean-1)
   rnbinom(n,mu=mu,size=k,...)
 }
```

(the ... in the function takes any other arguments you give to `my_dnbinom` and just passes them through, unchanged, to `dnbinom`).

Now we test that this really worked:

```
> rr = my_rnbinom(10000,mean=3,var=5)
> ## correct values
> mean(rr); var(rr)
[1] 3.0152
[1] 5.093478
> plot(table(rr)/10000,ylab="prob")
> points(0:16,my_dnbinom(0:16,mean=3,var=5),col=2)
```



Defining your own functions can be handy if you need to work on a regular basis with a distribution that uses a different parameterization than the one built into the standard R function.

***Exercise 3**: Morris (1997) gives a definition of the beta distribution that differs from the standard statistical parameterization. The standard parameterization is

$$\text{Beta}(x|a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}$$

whereas Morris uses

$$\text{Beta}(x|P,\theta) = \frac{\Gamma(\theta)}{\Gamma(\theta P)\Gamma(\theta(1-P))} x^{\theta P-1}(1-x)^{\theta(1-P)-1}.$$

Find expressions for P and θ in terms of a and b and vice versa. Explain why you might prefer Morris's parameterization. Define a new set of functions that generate random deviates from the beta distribution (`my_rbeta`) and calculate the density function (`my_dbeta`) in terms of P and θ. Generate a histogram from this distribution and draw a vertical line showing the mean of the distribution.

3 Averaging across discrete and continuous distributions

Suppose we have a (tiny) data set; we can organize it in two different ways, in standard long format or in tabular form:

```
> (dat <- c(5,6,5,7,5,8))

[1] 5 6 5 7 5 8

> (tabdat=table(dat))

dat
5 6 7 8
3 1 1 1
```

To get the (sample) probability distribution of the data, just scale by the total sample size:

```
> prob=tabdat/length(dat); prob

dat
        5         6         7         8
0.5000000 0.1666667 0.1666667 0.1666667
```

(dividing by `sum(tabdat)` would be equivalent).

In the long format, we can take the mean with `mean(dat)` or, replicating the formula $\sum x_i/N$ exactly, `sum(dat)/length(dat)`.

In the tabular format, we can calculate the mean with the formula $\sum P(x)x$, which in R would be `sum(prob*5:8)` or more generally

```
> vals <- as.numeric(names(prob))
> sum(prob*vals)

[1] 6
```

(you could also get the values by `as.numeric(levels(prob))`, or by `sort(unique(dat))`).

However, `mean(prob)` or `mean(tabdat)` is just plain wrong (at least, I can't think of a situation where you would want to calculate this value).

Going back the other way, from a table to raw values, we can use the `rep()` function to repeat values an appropriate number of times. In its simplest form, `rep(x,n)` just creates a vector repeats `x` (which may be either a single value or a vector) `n` times but **if `n` is a vector as well** then each element of `x` is repeated the corresponding number of times: for example,

```
> rep(c(1,2,3),c(2,1,5))
```

```
[1] 1 1 2 3 3 3 3 3
```

gives two copies of 1, one copy of 2, and five copies of 3.

Therefore,

```
> rep(vals,tabdat)
```

```
[1] 5 5 5 6 7 8
```

will recover our original data (although not in the original order) by repeating each element of `vals` the correct number of times.

3.1 Jensen's inequality

Looking at the data from Schmitt et al. (1999) on damselfish recruitment, the density of settlers nearly fits an exponential distribution with a mean of 24.2 (so λ, the rate parameter, $\approx 1/24.2$: Figure 3). Schmitt et al. also say that recruitment (R) as a function of settlement (S) is $R = aS/(1 + (a/b)S)$, with $a = 0.696$ (initial slope, recruits per 0.1 m^2 patch reef per settler) and $b = 9.79$ (asymptote, recruits per 0.1 m^2 patch reef at high settlement density).

Let's see how strong Jensen's inequality is for this population and recruitment function. We'll figure out the average by approximating an integral by a sum: $E[R] = \int_0^\infty f(S)P(S)\,dS \approx \sum f(S_i)P(S_i)\Delta S$, where $f(S)$ is the density of recruits as a function of settlers and $P(S)\,dS$ is the probability of a particular number of settlers. We need to set the range big enough to get most of the probability of the distribution, and the ΔS small enough to get most of the variation in the distribution; we'll try 0–200 in steps of 0.1. (If I set the range too small or the ΔS too big, I'll miss a piece of the distribution or the function. If I try to be too precise, I'll waste time computing.)

In R:

```
> a <- 0.696; b <- 9.79
> dS <- 0.1
> Smax <- 200
> S <- seq(0,Smax,by=dS)
> pS <-  dexp(S,rate=1/24.5)
> fS <- a*S/(1+(a/b)*S)
> sum(pS*fS)*dS
```

```
[1] 5.008049
```

If you have time, try a few different values for $S_{\max}$ and ΔS to see how the results vary.

R also knows how to integrate functions numerically: it can even approximate an integral from 0 to ∞ (`Inf` denotes $+\infty$ in R, in contexts where it makes sense). First we have to define a (vectorizable) function:

```
> tmpf <- function(S) {
   dexp(S,rate=1/24.5)*a*S/(1+(a/b)*S)
 }
```

```
> library(emdbook)
> data(DamselSettlement)
> par(mar=c(5,4,2,4)+0.1) ## leave room for right axis
> with(DamselSettlement,hist(density,breaks=100,freq=FALSE,las=1,
                             xlab="Settlement density",
                             ylab="Probability density",
                             main=""))
> m <- with(DamselSettlement,mean(density))
> curve(dexp(x,1/m),lty=2,lwd=2,add=TRUE)
> par(new=TRUE)
> a <- 0.696; b <- 9.79
> curve(a*x/(1+(a/b)*x),from=0,to=463.4,axes=FALSE,ann=FALSE,
       lty=3,lwd=2)
> axis(side=4,las=1)
> mtext(side=4,"Recruit density",line=2.5)
```

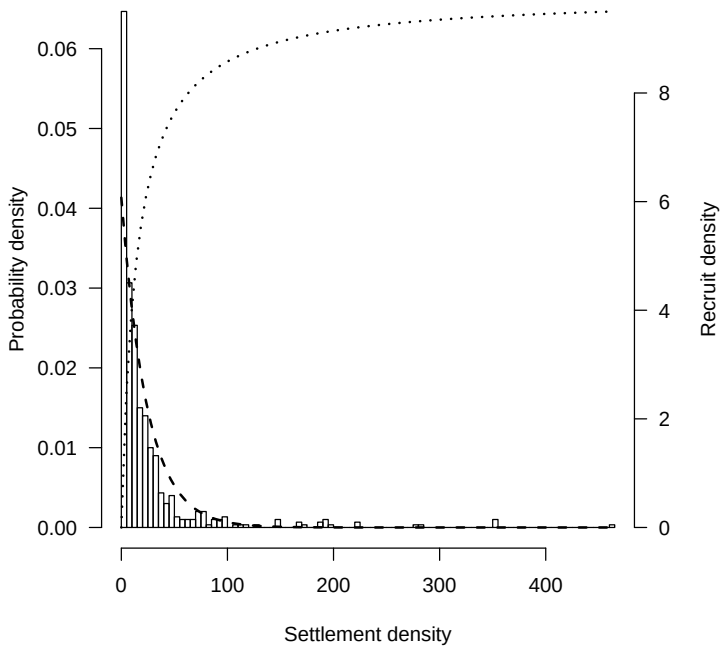


Figure 3: Settlement densities of larval *Dascyllus trimaculatus* on anemones: Schmitt et al. (1999)

Then we can just ask R to integrate it:

```
> (i1 <- integrate(tmpf,lower=0,upper=Inf))
```

```
5.010691 with absolute error < 5.5e-05
```

(Use `adaptIntegrate()`, in the `cubature` package, if you need to do multidimensional integrals, and note that if you want to retrieve the numerical value of the integral for use in code you need to say `i1$value`; use `str(i1)` to see the other information encoded in `i1`.)

This integral shows that we were pretty close with our first approximation. However, numerical integration will always imply some level of approximation; be careful with functions with sharp spikes, because it can be easy to miss important parts of the function.

Now to try out the delta function approximation, which is that $(E[f(x)] \approx f(\bar{x}) + (\sigma^2 f''(\bar{x})/2))$.

The first derivative is $a(1 + ax/b)^{-2}$; the second derivative is $-2a^2/b(1 + ax/b)^{-3}$ (confirm this for yourself).

You can calculate the derivatives with the `D` function in R, but R doesn't do any simplification, so the results are very ugly:

```
> ## first derivative:
> (d2 <- D(D(expression(a*x/(1+(a/b)*x)),"x"),"x"))
```

```
-(a * (a/b)/(1 + (a/b) * x)^2 + (a * (a/b)/(1 + (a/b) * x)^2 -
    a * x * (a/b) * (2 * ((a/b) * (1 + (a/b) * x)))/((1 + (a/b) *
        x)^2)^2))
```

It's best, whenever possible, to compute derivatives by hand and only use R to check your results.

As stated above, the mean value of the distribution is about 24.5. The variance of the exponential distribution is equal to the mean squared, or 600.25.

```
> Smean <- 24.5
> Svar <- Smean^2
> a <- 0.696
> b <- 9.79
> ## first derivative, will use this later for pictures:
> d1_num <- a*(1+a*Smean/b)^(-2)
> ## second derivative
> (d2_num <- -2*a^2/b*(1+a*Smean/b)^(-3))
```

```
[1] -0.004801415
```

```
> eval(d2,list(x=Smean)) ## check with R
```

```
[1] -0.004801415
```

```
> mval = a*Smean/(1+(a/b)*Smean)
> (dapprox = mval + 1/2*Svar*d2_num)
```

```
[1] 4.778299
```

Calculate relative errors of different approaches:

```
> (merr = (mval-i1$value)/i1$value)
```

```
[1] 0.2412107
```

```
> (err = (dapprox-i1$value)/i1$value)
```

```
[1] -0.04637931
```

The answer from the delta method is only about -24% below the true value, as opposed to the naive answer ($f(\bar{x})$) which is about -5% high.

OPTIONAL section:
I also tried this problem in Mathematica, which was able to give me a closed-form solution to

$$\int_0^\infty \lambda e^{-\lambda S} \left(\frac{aS}{1+(a/b)S} \right) = b e^{b\lambda/a} \cdot \texttt{ExpIntegralE}(2, b\lambda/a)$$

where `ExpIntegralE` is an "exponential integral" function (don't worry about it too much). R can compute the type 2 exponential integral (as specified by the first 2 in the parentheses) via the `expint_E2` function in the `gsl` package (if you want to run this code you have to install the GSL libraries on your computer, too, outside of R):

```
> library(gsl)
> fun2 = function(a,b,lambda) {
   b*exp(b*lambda/a)*expint_E2(b*lambda/a)
 }
> f2 = fun2(0.696,9.79,1/24.5)
> f2-i1$value
```

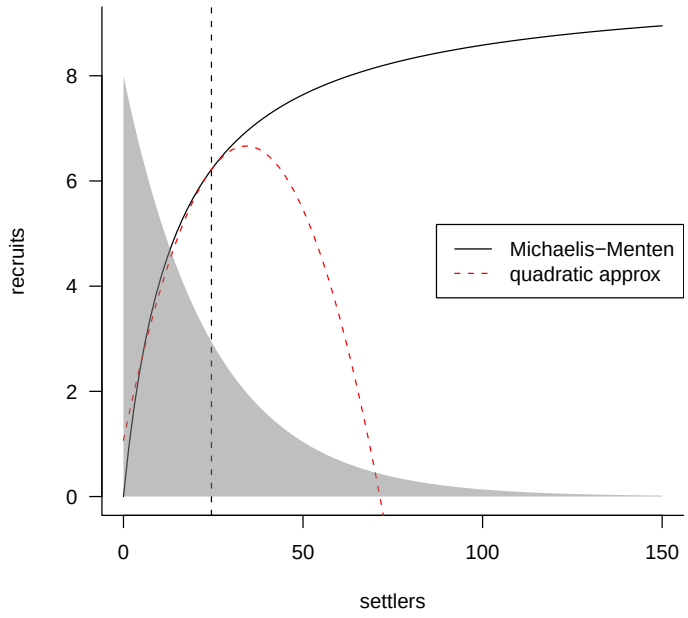
```
[1] 6.494716e-11
```

There is very little difference between the numerical integral and the exact solution . . .)
end OPTIONAL section

A picture . . . (I'm getting a bit fancy here, using `rgb()` to define a transparent gray color and then using `polygon()` to fill an area with this color: I use an x vector that starts at 0, runs to 150, and then runs from 150 to 0, while the y vector first traces out the top of the probability distribution and then runs along zero (the bottom). The transparent gray may not show up in HTML versions of the page, or other places where the graphics system doesn't support transparency.)

```
> a = 0.696
> b = 9.79
> Smean = 24.5
```

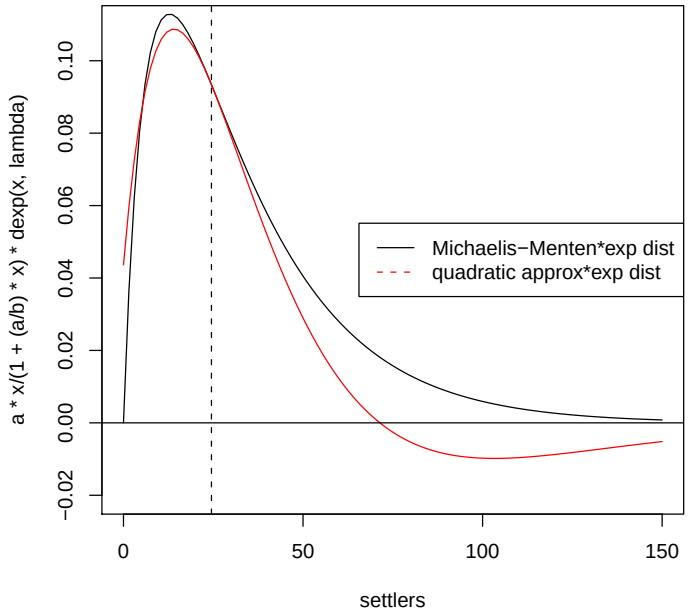
```
> lambda = 1/Smean
> curve(a*x/(1+(a/b)*x),from=0,to=150,xlab="settlers",
      ylab="recruits",las=1,bty="l")
> abline(v=Smean,lty=2)
> curve(mval+(x-Smean)*d1_num+(x-Smean)^2*d2_num,add=TRUE,lty=2,col=2)
> trgray = rgb(0.5,0.5,0.5,alpha=0.5)
> xvec = 0:150
> polygon(c(xvec,rev(xvec)),
        c(dexp(lambda*xvec)*8,rep(0,length(xvec))),
        col=trgray,border=NA)
> legend("right",c("Michaelis-Menten","quadratic approx"),
       col=c("black","red"),
       lty=1:2)
```



Drawing the thing we're actually trying to integrate, and its quadratic approximation:

```
> curve(a*x/(1+(a/b)*x)*dexp(x,lambda),from=0,to=150,xlab="settlers",
      ylim=c(-0.02,0.11))
> abline(v=Smean,lty=2)
> curve((mval+(x-Smean)*d1_num+(x-Smean)^2*d2_num)*dexp(x,lambda),add=TRUE,col=2,
      from=0)
> legend("right",c("Michaelis-Menten*exp dist","quadratic approx*exp dist"),
```

```
        col=c("black","red"),
        lty=1:2)
> abline(h=0)
```



***Exercise 4**: go through the above process again, but this time use a Gamma distribution instead of an exponential. Keep the mean equal to 24.5 and change the variance to 100, 25, and 1, respectively (use the information that the mean of the gamma distribution is `shape*scale` and the variance is `shape*scale^2`; use the method of moments). Including the results for the exponential (which is a gamma with shape=1), make a table showing how the (1) true value of mean recruitment [calculated by numerical integration in R either using `integrate()` or summing over small ΔS] (2) value of recruitment at the mean settlement (3) delta-method approximation (4,5) proportional error in #2 and #3 change with the variance.

Optional:

4 Creating new distributions

4.0.1 Finite mixture distributions

The general recipe for generating samples from finite mixtures is to use a uniform distribution to sample which of the components of the mixture to sample, then use `ifelse` to pick values from one distribution or the other. To pick 1000 values from a mixture of normal distributions with the parameters shown in the chapter ($p = 0.3$, $\mu_1 = 1$, $\sigma_1 = 2$, $\mu_2 = 5$, $\sigma_2 = 1$):

```
> u1 <- runif(1000)
> z <- ifelse(u1<0.3,rnorm(1000,mean=1,sd=2),
            rnorm(1000,mean=5,sd=1))
> hist(z,breaks=100,freq=FALSE)
```

The probability density of a finite mixture composed of two distributions D_1 and D_2 in proportions p_1 and $1-p_1$ is $p_1 D_1 + p_2 D_2$. We can superimpose the theoretical probability density for the finite mixture above on the histogram:

```
> curve(0.3*dnorm(x,mean=1,sd=2)+0.7*dnorm(x,mean=5,sd=1),
      add=TRUE,lwd=2)
```

4.1 Zero-inflated distributions

The general formula for the probability distribution of a zero-inflated distribution, with an underlying distribution $P(x)$ and a zero-inflation probability of p_z, is:

$$\begin{aligned} \text{Prob}(0) &= p_z + (1-p_z)P(0) \\ \text{Prob}(x>0) &= (1-p_z)P(x) \end{aligned}$$

So, for example, we could define a probability distribution for a zero-inflated negative binomial as follows:

```
> dzinbinom = function(x,mu,size,zprob) {
   ifelse(x==0,
          zprob+(1-zprob)*dnbinom(0,mu=mu,size=size),
          (1-zprob)*dnbinom(x,mu=mu,size=size))
 }
```

(the name, `dzinbinom`, follows the R convention for a probability distribution function: a `d` followed by the abbreviated name of the distribution, in this case `zinbinom` for "**z**ero-**i**nflated **n**egative **binom**ial").

The `ifelse()` command checks every element of `x` to see whether it is zero or not and fills in the appropriate value depending on the answer.

A random deviate generator would look like this:

```
> rzinbinom = function(n,mu,size,zprob) {
   ifelse(runif(n)<zprob,
          0,
          rnbinom(n,mu=mu,size=size))
 }
```

The command `runif(n)` picks `n` random values between 0 and 1; the `ifelse` command compares them with the value of `zprob`. If an individual value is less than `zprob` (which happens with probability `zprob`$=p_z$), then the corresponding random number is zero; otherwise it is a value picked out of the appropriate negative binomial distribution.

optional exercise: Check graphically that these functions actually work. For an extra challenge, calculate the mean and variance of the zero-inflated negative binomial and compare it to the results of `rzinbinom(10000,mu=4,size=0.5,zprob=0.2)`.

5 Compounding distributions

The key to compounding distributions in R is that the functions that generate random deviates can all take a vector of different parameters rather than a single parameter. For example, if you were simulating the number of hatchlings surviving (with individual probability 0.8) from a series of 8 clutches, all of size 10, you would say

```
> rbinom(8,size=10,prob=0.8)

[1]  9  8  8  8  8  7  9 10
```

but if you had a series of clutches of different sizes, you could still pick all the random values at the same time:

```
> clutch_size = c(10,9,9,12,10,10,8,11)
> rbinom(8,size=clutch_size,prob=0.8)

[1]  8  8  8 11  8  8  6  7
```

Taking this a step farther, the clutch size itself could be a random variable:

```
> clutch_size = rpois(8,lambda=10)
> rbinom(8,size=clutch_size,prob=0.8)

[1]  9  6  8  7  2  6 13  8
```

We've just generated a Poisson-binomial random deviate ...

As a second example, I'll follow Clark *et al.* in constructing a distribution that is a compounding of normal distributions, with 1/variance of each sample drawn from a gamma distribution.

First pick the variances as the reciprocals of 10,000 values from a gamma distribution with shape 5 (setting the scale equal to 1/5 so the mean will be 1):

```
> var_vals=1/rgamma(10000,shape=5,scale=1/5)
```

Take the square root, since `dnorm` uses the standard deviation and not the variance as a parameter:

```
> sd_vals = sqrt(var_vals)
```

Generate 10,000 normal deviates using this range of standard deviations:

```
> x = rnorm(10000,mean=0,sd=sd_vals)
```

Figure 4 shows a histogram of the following commands:

```
> hist(x,prob=TRUE,breaks=100,col="gray")
> curve(dt(x,df=11),add=TRUE,lwd=2)
```

```
> hist(x,prob=TRUE,breaks=100,col="gray")
> curve(dt(x,df=11),add=TRUE,lwd=2)
```

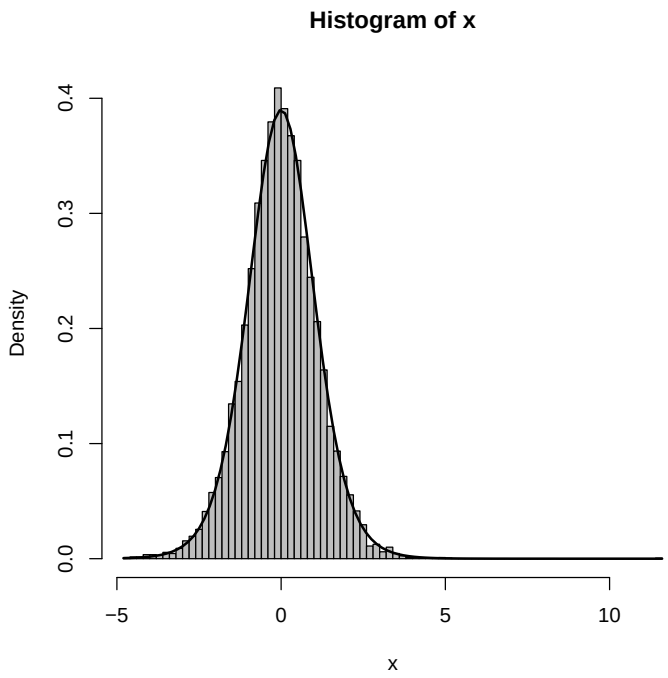


Figure 4: Clark model: inverse gamma compounded with normal, equivalent to the Student t distribution

The superimposed curve is a t distribution with 11 degrees of freedom; it turns out that if the underlying gamma distribution has shape parameter p, the resulting t distribution has $df = 2p + 1$. (Figuring out the analytical form of the compounded probability distribution or density function, or its equivalence to some existing distribution, is the hard part; for the most part, though, you can find these answers in the ecological and statistical literature if you search hard enough.

****Exercise 5**: Generate 10,000 values from a gamma-Poisson compounded distribution with parameters shape=$k = 0.5$, scale=$\mu/k = 4/0.5 = 8$ and demonstrate that it matches to a negative binomial with the equivalent parameters $\mu = 4$ and $k = 0.5$.

Optional: generate 10,000 values from a lognormal-Poisson distribution with the same expected mean and variance (the variance of the lognormal should equal the variance of the gamma distribution you used as a compounding distribution; you will have to do some algebra to figure out the values of `meanlog` and `sdlog` needed to produce a lognormal with a specified mean and variance. Plot the distribution and superimpose the theoretical distribution of the negative binomial with the same mean and variance to see how different the shapes of the distributions are.

References

Morris, W. F. 1997. Disentangling effects of induced plant defenses and food quantity on herbivores by fitting nonlinear models. American Naturalist **150**:299–327.

Schmitt, R. J., S. J. Holbrook, and C. W. Osenberg. 1999. Quantifying the effects of multiple processes on local abundance: a cohort approach for open populations. Ecology Letters **2**:294–303.