Probabilistic processes

Tad Dallas

Contents

What do I mean by "probabilistic processes"?

We have already gone over a bit of probabilistic processes in a roundabout way. Throughout this course, we have generated random data from probability distributions. For instance, we pulled values from a uniform distribution, in which all values are equally probable of being drawn between two bounds (`runif(100, 1, 10)`).

Probabilistic processes are simply those processes whose outcome is determined by some probability (e.g., every time I flip a coin, I don't know the outcome, right?). We can code this up, right? Every trial of a coin flip results in 1 of 2 outcomes (i.e., heads or tails) with the probability we assume is 0.5 (both outcomes are equally likely). The probability distribution corresponding to this situation is the binomial distribution.

For instance, the following code would simulate flipping a coin 100 times.

```
coin <- rbinom(100, 1, p=0.5)
sum(coin == 1)
```

```
## [1] 54
```

As a biological example, consider an animal who disperses with some probability p, and moves a distance determined by a Poisson distribution with mean 10km. How do we code this up?

This starts to explore how you might go about setting up a simulation model, but this is already getting spatial and going into things that are a bit more complicated, so let's dial it back.

But the above incorporation of probabilistic processes is based on phenomenological modeling, where many of us might also want to do some statistical modeling. We will essentially do both in this lecture, where we first go into probability distributions as they relate to gaining statistical insight from your data, and end with some simulation modeling to show how probabilistic processes influence population dynamics.

notation in R

"d" returns the height of the probability density function "p" returns the cumulative density function "q" returns the inverse cumulative density function (quantiles) "r" returns randomly generated numbers

Generate random draws from probability distributions

```
rnorm(100, 1, 1)
```

```
##   [1] -0.207815198  1.735425684  3.022737582  1.765059792  2.923215067
##   [6] -1.033732709  1.130215690 -0.070227435  2.402246223  2.519555240
##  [11]  0.731362567 -0.008904507  2.864372817  0.226620554  0.764379440
##  [16] -0.022273194  0.294408839  0.532211622  0.729840517  0.417417817
##  [21] -0.635768419 -0.605143267  2.782459463  0.586327161  3.174169071
##  [26]  4.469588695  0.134562715  1.060307740  2.188889367  2.289343351
##  [31]  1.176786363  2.973824801  1.509972537  0.417561020  1.140100277
##  [36]  0.706914479  1.806367893  1.750433350  0.355281894  1.003308571
##  [41] -1.438251949  2.197036866  0.045066292  4.227891765  1.476427922
##  [46]  2.220215888 -0.159897737  1.539919985 -0.039342728  0.765164024
##  [51]  1.073036551  1.263528638  0.813985759  1.121236151  0.239384719
##  [56]  0.206093836  1.232692440  0.816891037  1.363614968 -1.032706340
##  [61]  3.209946548  2.407492950  2.016514657  0.996177526  1.552273961
##  [66]  2.374693133 -0.848268156  1.998123310  0.724522213 -1.108265058
##  [71]  1.629738701 -1.338993460  1.355273478 -0.563007643  0.925402371
##  [76] -0.568906835  1.414239816  1.585914529 -0.179448868  1.380567019
##  [81]  2.449959510  1.289224907  0.980259367  1.749492891  1.332534490
##  [86] -0.438303126  0.270345840  2.458104637  1.051537902  0.918136272
##  [91]  0.830343239  1.003501418 -0.099023777  1.199002519  1.475610969
##  [96]  1.129417662  0.390920411  1.625113980  1.371034583  1.619484870
```

```r
rbinom(100, 1, 0.25)
```

```
##   [1] 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0
##  [38] 1 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0
##  [75] 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
```

```r
rpois(100, 1)
```

```
##   [1] 0 0 1 1 1 1 1 3 1 1 1 2 0 0 0 0 2 2 0 0 0 1 4 1 3 1 2 1 1 3 1 1 0 1 3 0 2
##  [38] 0 2 0 0 1 0 0 0 1 2 0 2 0 1 4 2 2 2 0 0 1 2 0 1 0 1 2 1 1 0 0 1 1 1 1 1 0
##  [75] 0 1 1 0 0 1 0 0 2 1 3 1 3 0 1 3 0 1 1 1 1 2 0 0 2 1
```

```r
runif(100, 1, 20)
```

```
##   [1] 16.722478  9.082572  2.045442  1.719540 14.103334 16.217595  2.999743
##   [8]  4.460468 12.939825  8.995811 19.249649 19.373585  7.048109 12.479884
##  [15] 18.726843  7.320018 11.994626  9.527106 13.108992  4.391945  8.013457
##  [22]  1.945672 12.567383 13.399425 14.364195  2.028241  5.899670  7.230115
##  [29] 15.390832  7.482674  7.647650  5.615011  8.387686  8.955306  3.155938
##  [36] 18.134074 12.166372  7.753933  9.023951 12.456360 14.860852  4.141080
##  [43]  3.182872  9.986903  2.944929 15.566049  8.432180 13.541108  9.179593
##  [50]  7.392131  1.605569 10.027342  7.054897  8.397027  6.704133  4.122388
##  [57] 15.617456  1.119612 13.844664 19.721395 10.559125 18.422971 12.892680
##  [64]  3.524112 15.111855 17.710418 15.696766  8.986082  1.642030 15.657701
##  [71]  6.810477  2.643002  7.915549 14.820772  4.488889  6.599429  3.652822
##  [78] 19.836044 16.009891 14.424747  9.807975  7.393215  5.356093  2.583091
##  [85]  4.144963  9.548753  6.289554 15.303903  5.951972  6.057229  1.427868
##  [92] 15.765468 15.725649  4.481562  4.009345  8.109520  7.953005  5.995981
##  [99] 17.009672 14.217406
```

```r
rgamma(100, 1, 1)
```

```
##   [1] 0.85607888 1.04454900 0.61786513 0.02622163 0.17439112 0.78634397
##   [7] 1.58792894 0.10499650 1.31049261 0.92941049 0.63789329 0.41645897
##  [13] 0.51385282 0.72190025 0.34214867 2.20493003 1.28121283 0.01541322
##  [19] 1.41582315 0.27523164 0.18067967 0.52021602 0.47825697 0.60131404
##  [25] 1.28085876 1.50664430 2.68007839 0.13388303 0.30838015 0.46782382
##  [31] 1.58494010 1.06577056 0.78144785 0.97390114 1.76626155 0.18655360
##  [37] 0.68040785 0.21965346 1.03922289 0.76442735 0.32075984 0.51599956
##  [43] 1.01639427 0.09526688 1.74400501 0.30420548 2.81965872 0.79729077
##  [49] 0.74476896 3.10712074 0.73706376 0.39998180 0.33680384 0.57322322
##  [55] 0.53628815 0.20455454 0.40644605 2.20260946 0.24535299 1.93283741
##  [61] 2.97385192 1.43450748 0.73865164 0.58581542 0.68832861 0.13359753
##  [67] 0.18540230 1.79105545 0.85911041 0.15261883 1.26646095 0.82906902
##  [73] 0.09583302 1.17591614 0.58243357 0.15627711 0.03567596 0.96235024
##  [79] 0.40203932 2.24194649 0.36741158 1.57676730 0.11557406 0.15500849
##  [85] 0.63057155 2.76734473 2.10788832 0.91104455 1.28494189 0.34518280
##  [91] 0.41471890 0.16912928 0.28531922 0.05492508 0.52466526 0.62954640
##  [97] 1.17679820 0.57925186 0.53297503 0.27148328
```

Given a number or a list it computes the probability that a random number will be less than that number.

```
pnorm(-1, mean=0, sd=1)
```

```
## [1] 0.1586553
```

```
pnorm(1, mean=0, sd=1)
```

```
## [1] 0.8413447
```

```
pnorm(1, mean=0, sd=1, lower.tail=FALSE)
```

```
## [1] 0.1586553
```

```
pbinom(0, 1, 0.15)
```

```
## [1] 0.85
```

```
ppois(1, lambda=2)
```

```
## [1] 0.4060058
```

```
punif(0.25, 0, 1)
```

```
## [1] 0.25
```

The next function we look at is `q----` which is the inverse of `p----`. The idea behind `q----` is that you give it a probability, and it returns the number whose cumulative distribution matches the probability.

```
pnorm(0.25, mean=0, sd=1)
```

```
## [1] 0.5987063
```

```
qnorm(0.25, mean=0, sd=1)
```

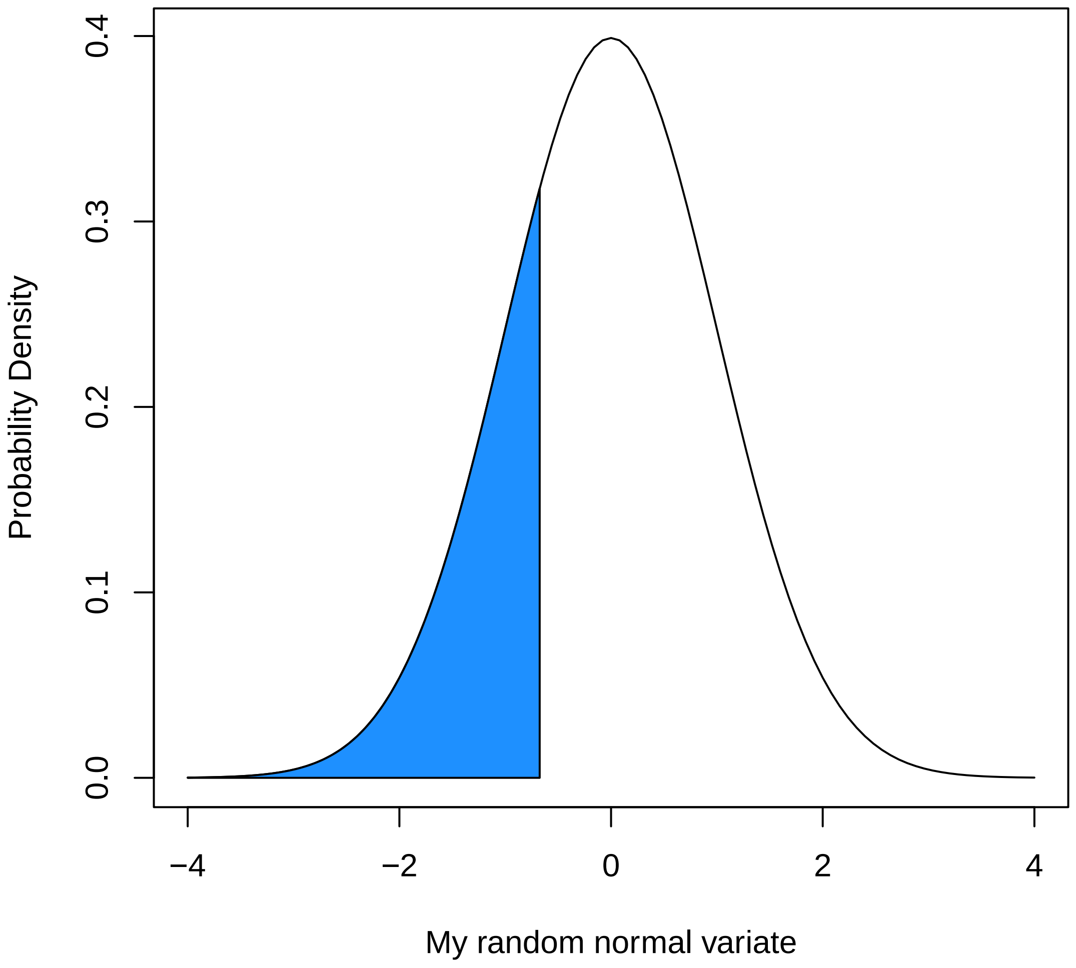
```
## [1] -0.6744898
```

```
pnorm(-0.6744898, mean=0, sd=1)
```

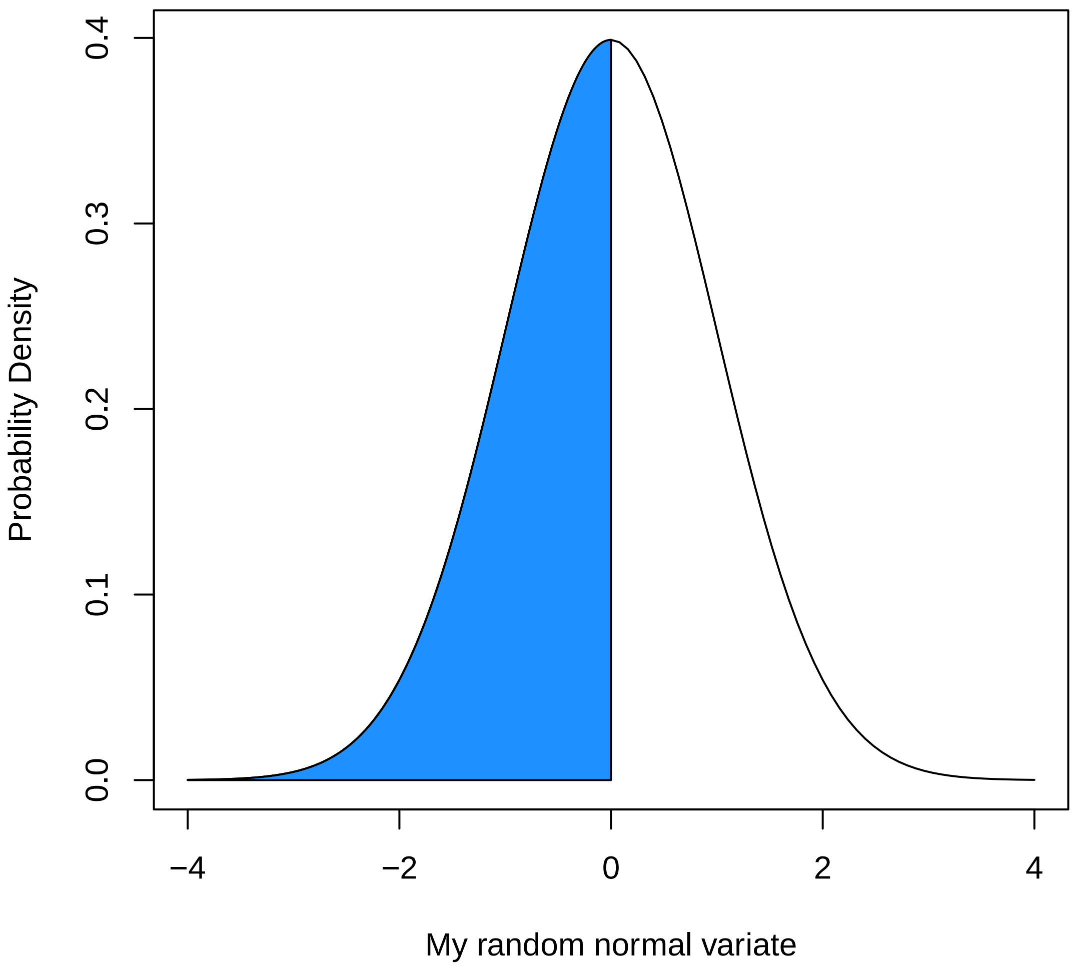
```
## [1] 0.25
```

So this means that to the left of -0.6745, we should have 25% of the weight of the data, right?

```
colorArea <- function(from, to, density, ..., col="dodgerblue", dens=NULL){
    y_seq <- seq(from, to, length.out=500)
    d <- c(0, density(y_seq, ...), 0)
    polygon(c(from, y_seq, to), d, col=col, density=dens)
}

curve(dnorm(x), from=-4, to=4,
  ylab = "Probability Density",
  xlab = "My random normal variate")

colorArea(from=-4, to=qnorm(0.25), dnorm)
```

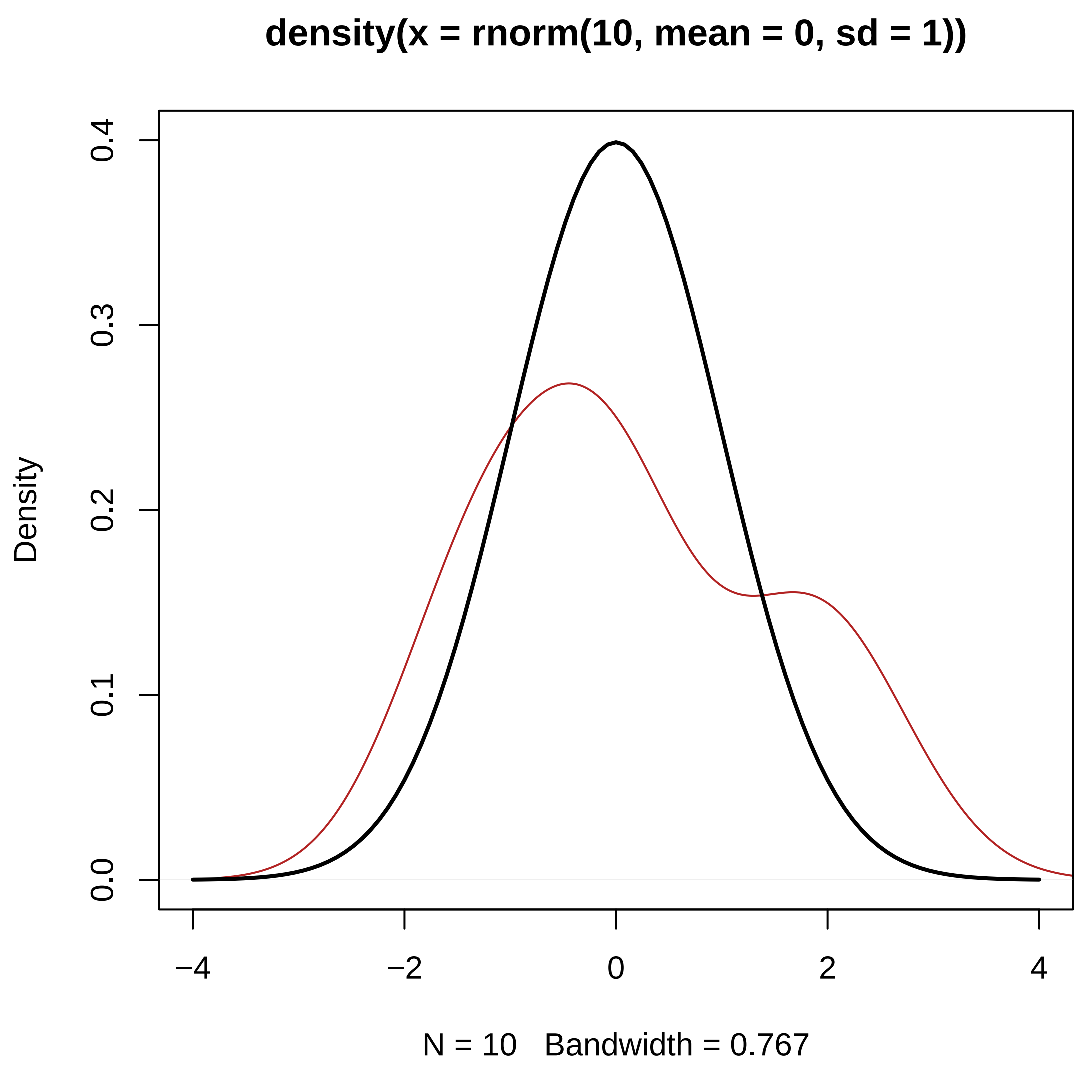


```
curve(dnorm(x), from=-4, to=4,
  ylab = "Probability Density",
  xlab = "My random normal variate")

colorArea(from=-4, to=qnorm(0.5), dnorm)
```

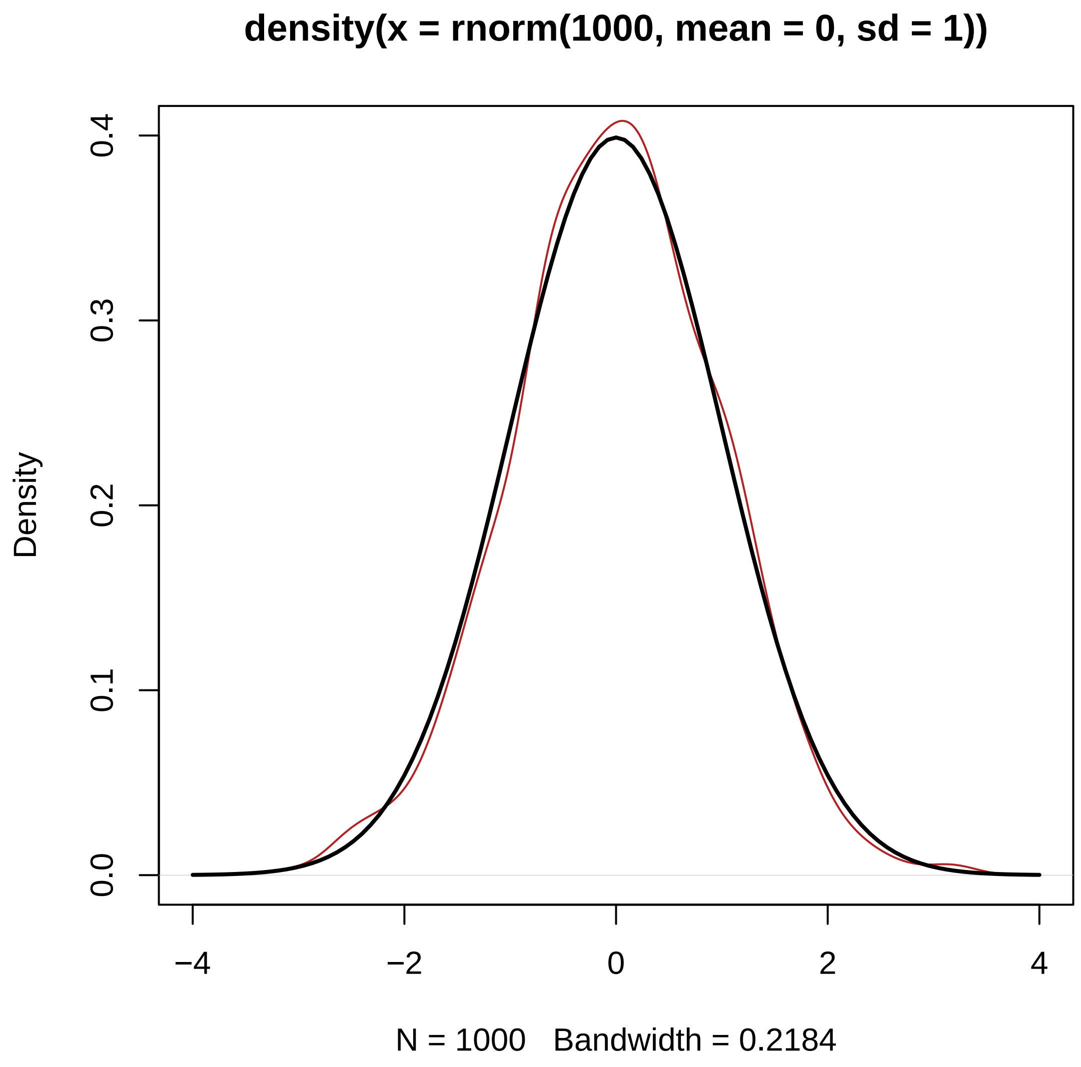


So this means that if we take random draws from a distribution, we should be able to build up something that looks like the results of dnorm, right? But building a distribution from the random variates requires that we have enough samples to accurately capture the true underlying distribution. Let's explore this.

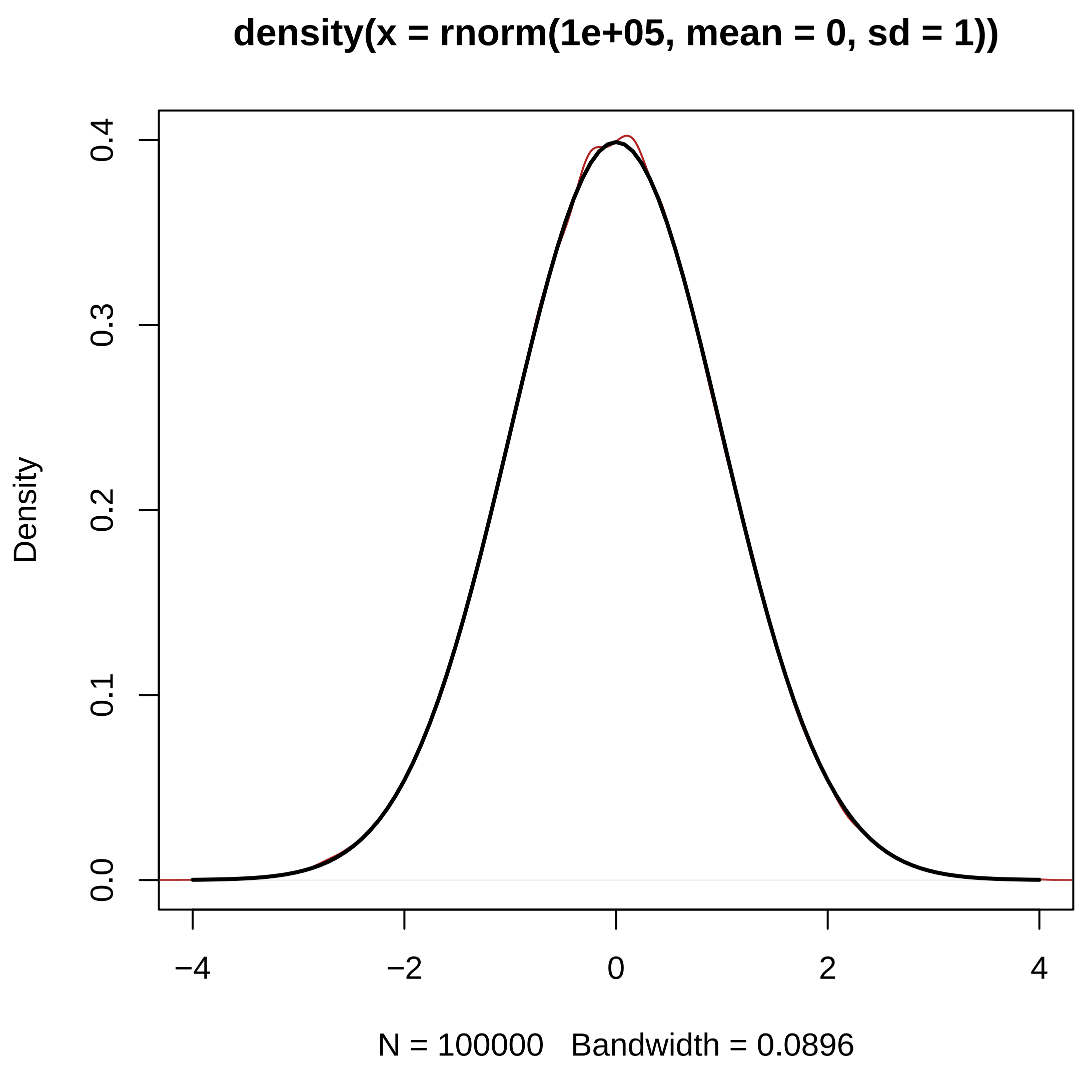
```r
plot(density(rnorm(10, mean=0, sd=1)), col='firebrick',
  xlim=c(-4,4), ylim=c(0,0.4))
curve(dnorm(x), from=-4, to=4, add=TRUE, lwd=2)
```



```
plot(density(rnorm(1000, mean=0, sd=1)), col='firebrick',
  xlim=c(-4,4), ylim=c(0,0.4))
curve(dnorm(x), from=-4, to=4, add=TRUE, lwd=2)
```

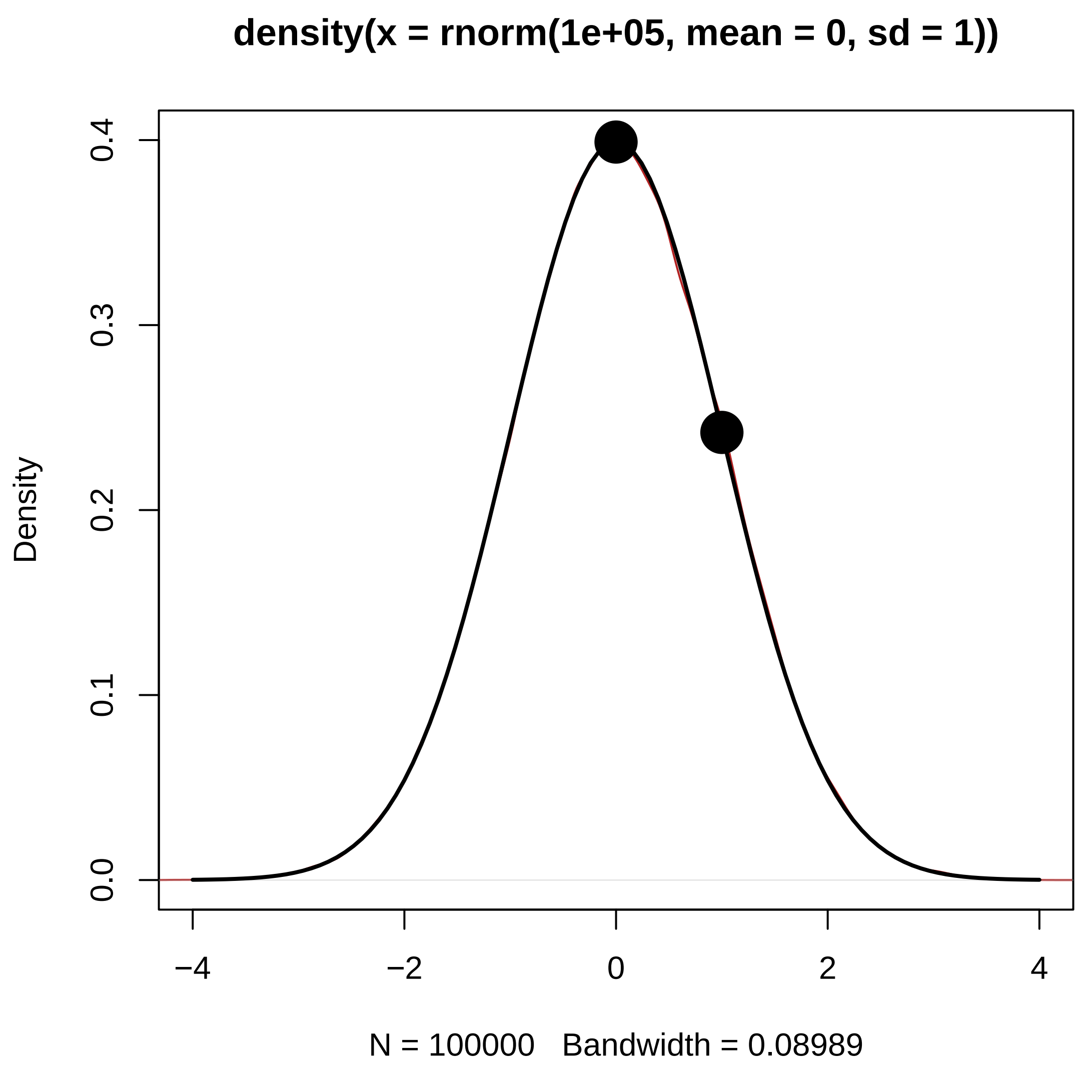


```r
plot(density(rnorm(100000, mean=0, sd=1)), col='firebrick',
  xlim=c(-4,4), ylim=c(0,0.4))
curve(dnorm(x), from=-4, to=4, add=TRUE, lwd=2)
```



The above code uses the `d---` function, where the `d` stands for the density. This gives the expected density of values that would occur at a certain point given the probability distribution. The `d` here corresponds to the probability density function, so while `p---` gave us probabilities and `q---` gave us the area under the curve, `d---` gives us the actual value of the density function.

```
plot(density(rnorm(100000, mean=0, sd=1)), col='firebrick',
  xlim=c(-4,4), ylim=c(0,0.4))
curve(dnorm(x), from=-4, to=4, add=TRUE, lwd=2)
points(x=0, y=dnorm(0), pch=16, cex=3)
points(x=1, y=dnorm(1), pch=16, cex=3)
```



But how does the above relate to statistical tests? We will explore this by considering an example of the t-test, but think about other statistical tests that may (and likely do) follow a similar structure in terms of how the test statistic is calculated and how we think about p-values.

A case study of the t-test

The t-test can be used to test for differences between two groups of data, or of one group of data and some mean μ. It is a comparison of means, so we implicitly assume no differences in variance or distributional shape between the two groups.

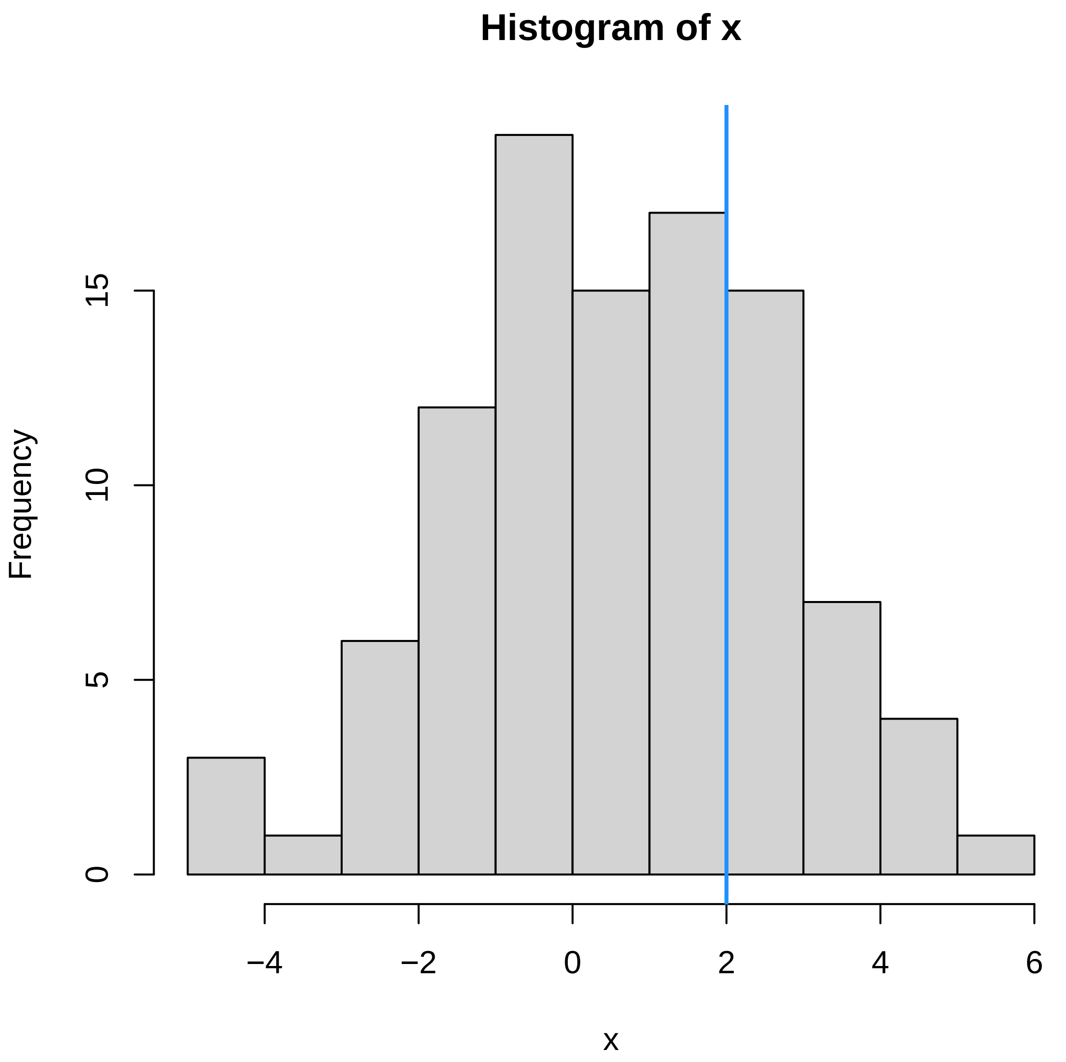
One sample

$$\frac{X - \mu}{sd(X)/\sqrt(n)}$$

This tests if the population mean is different from some value that you provide. In the example below, we generate random data from a normal distribution with mean 1 and variance 2. We want to know if the population mean of the random data are significantly different from a value of 2 ($\mu = 2$).

```
x <- rnorm(100, 1, 2)
mu <- 2
```

```r
hist(x)
abline(v=mu, col='dodgerblue', lwd=2)
```



```r
t.test(x, mu=mu)
```

```
##
##  One Sample t-test
##
## data:  x
## t = -6.6173, df = 99, p-value = 1.893e-09
## alternative hypothesis: true mean is not equal to 2
## 95 percent confidence interval:
##  0.2108829 1.0363220
## sample estimates:
## mean of x
## 0.6236025
```

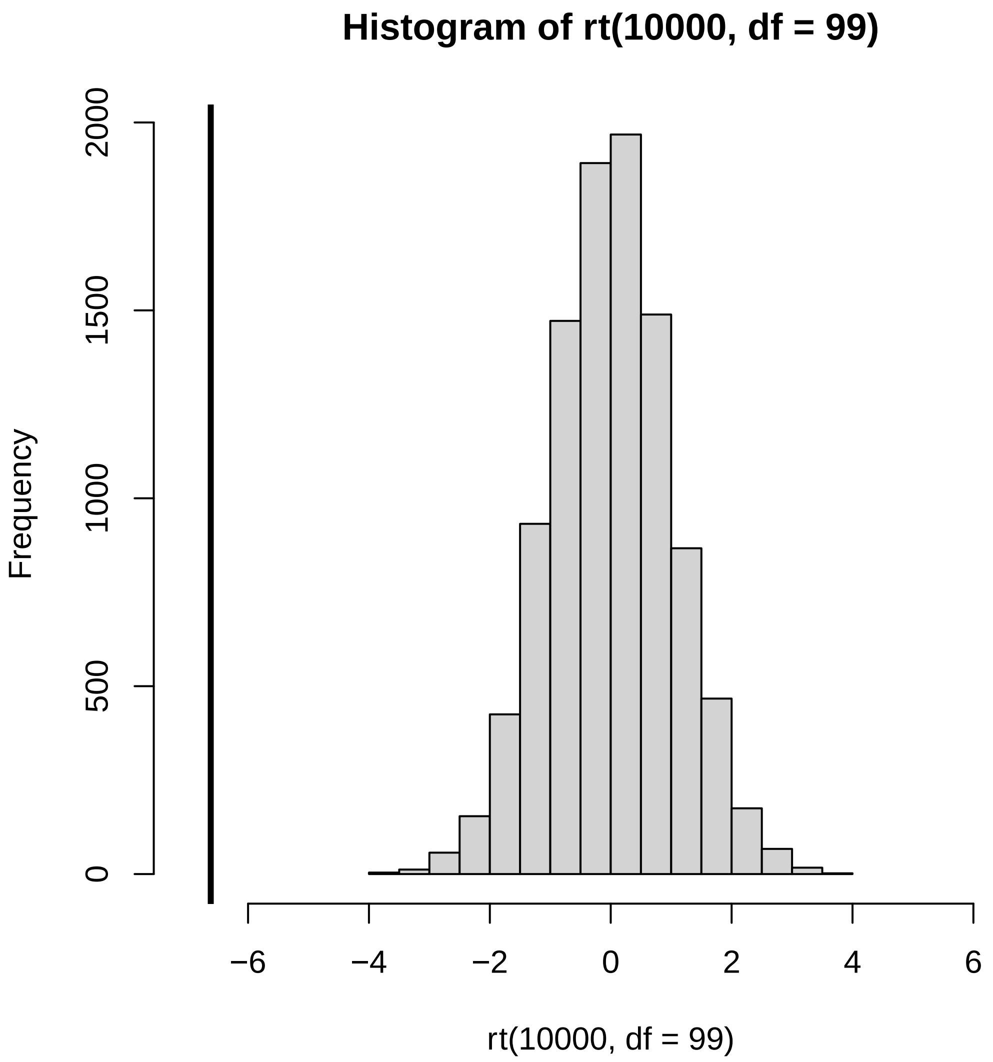
```r
tt <- (mean(x) - mu) / (sd(x) / sqrt(length(x)))
pt(tt, df=length(x)-1)
```

```
## [1] 9.464833e-10
```

```
# why is the above value different from the t-test calculation from `t.test`?

pt(tt, df=length(x)-1)*2
```

```
## [1] 1.892967e-09
```

```
hist(rt(10000, df=99), xlim=c(-7,7))
abline(v=tt, lwd=3)
```



Two-sample

$$\frac{X1 - X2}{sp}$$

$$sp = \frac{\sqrt{varx1 + varx2}}{2}$$

where `sp` is pooled variance

```
x1 <- rnorm(100, 1, 2)
x2 <- rnorm(100, 2, 1)
```

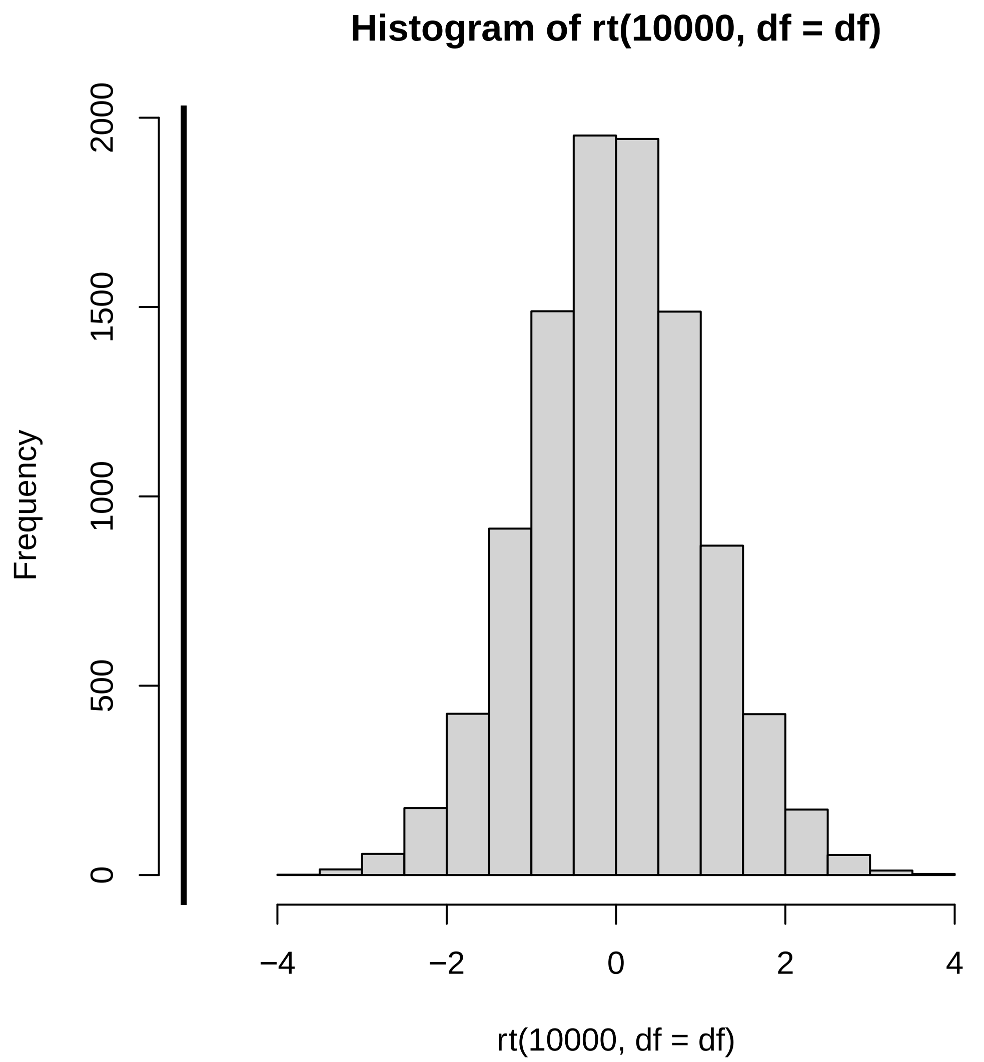
```r
t.test(x1, x2, var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  x1 and x2
## t = -5.1059, df = 198, p-value = 7.705e-07
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.5862588 -0.7023482
## sample estimates:
## mean of x mean of y
## 0.9600249 2.1043284
```

```r
sp <- sqrt((var(x1)+var(x2)) / 2)
tt2 <- (mean(x1)-mean(x2)) / (sp*sqrt(2/length(x1)))

df <- (2*length(x1)) - 2

pt(tt2, df=df) * 2
```

```
## [1] 7.705483e-07
```

```r
hist(rt(10000, df=df), xlim=c(-5,5))
abline(v=tt2, lwd=3)
```



```
## at larger sample sizes, the normal distribution is approximately equal to the t-distribution (a z-te

pt(tt2, df=df)
```

```
## [1] 3.852741e-07
```

```
pnorm(tt2)
```

```
## [1] 1.645982e-07
```

I flip a coin 100 times and get the following outcomes. What is the probability that this is a fair coin ($p = 0.5$)? (there are many ways to solve this problem. One approach would be to simulate a fair coin, calculate the number of heads, and compare this to the 'test' coin).

```
outcomes <- c(0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0)
```

```r
# the cheat way (binomial exact test)
binom.test(sum(outcomes), n=100, p=0.5)
```

```
##
##  Exact binomial test
##
## data:  sum(outcomes) and 100
## number of successes = 21, number of trials = 100, p-value = 4.337e-09
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.1349437 0.3029154
## sample estimates:
## probability of success
##                   0.21
```

```r
pbinom(sum(outcomes), 100, 0.5) * 2
```

```
## [1] 4.337367e-09
```

Effect size

One thing you may be thinking to yourself at this stage is that the outcomes of these functions seems pretty tied to the number of samples you have. You're right. We could observe a 'significant' result from 10 trials, like the probability of having 3 heads and 7 tails out of a sample of 10 is

```r
pbinom(3, 10, 0.5)
```

```
## [1] 0.171875
```

but the probability of having 30 heads and 70 tails is

```r
pbinom(30, 100, 0.5)
```

```
## [1] 3.92507e-05
```

But how much we 'trust' the outcome is not reported. One way of doing this is to calculate effect sizes. Cohen's d is a pretty classic measure of effect size (though certainly others exist). And the best part is that it is pretty simple. In the case above, we could compute the effect size as

$$d = \frac{\bar{X} - \mu_0}{\hat{\sigma}}$$

or the mean of the sample minus the expected value divided by the standard deviation of the sample.

Some practice problems

You flip a fair coin 10 times. What is the probability that you will have 2 heads and 8 tails outcomes?

An individual produces n offspring per year following a Poisson distribution with lambda = 2. Calculate the probability that a mother has more than 5 offspring in a year.

```r
1-dpois(5, 2)
```

```
## [1] 0.9639106
```

How about the probability of having less than 2 offspring, 2 years in a row? (we may not do this one, as it gets us a bit further in probability territory where some are quite uncomfortable)

What's the probability of observing a sentence with fewer than 25 characters given the sentences data?

```r
library(stringr)
data(sentences)
```

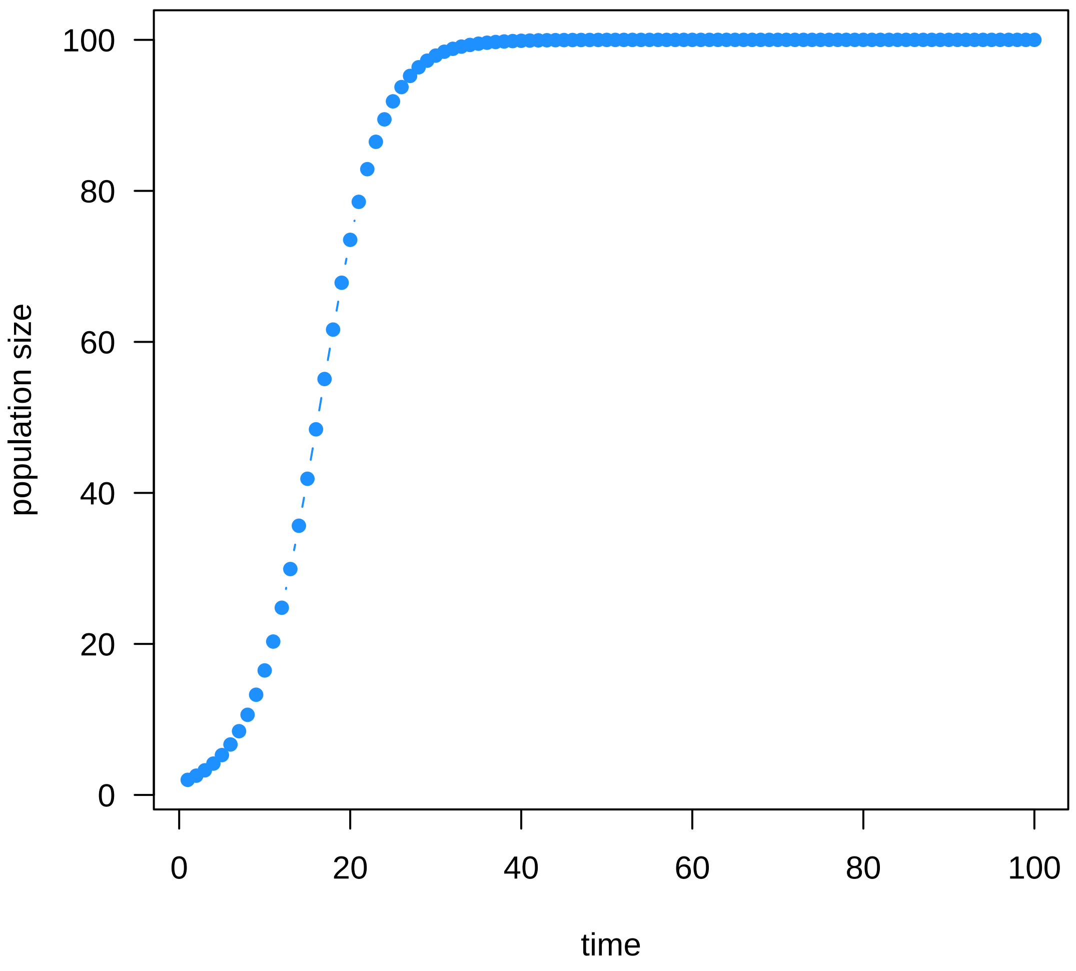
What is the above solution assuming about the distribution of the number of characters per sentence from the `sentences` data?

The logistic model of population growth

Incorporating probabilistic processes into simulation models allows researchers to not only see one realization of the dynamics given the parameters, but also to quantify and explore the variability in outcome solely as a result of random chance. We'll explore this a bit using a simple model of population dynamics.

In deterministic logistic growth, a population grows with growth rate r until it reaches a carrying capacity k. I code this model below and we can explore the dynamics.

```r
logisticGrowth <- function(n, r, k){
  n*exp(r*(1-(n / k)))
}



logisticDynamics <- function(n,r,k, steps=100){
  ret <- c()
  ret[1] <- n
  if(length(r) == 1){
    r <- rep(r, steps)
  }
  for(i in 1:(steps-1)){
    ret[i+1] <- logisticGrowth(ret[i], r[i], k)
  }
  return(ret)
}



plot(logisticDynamics(n=2, r=0.25, k=100, steps=100),
  type='b', las=1, ylab='population size', xlab='time',
  col='dodgerblue', pch=16)
```



But this is not how populations change over time, right?

Why not?

- No individual variation in birth rates
- No temporal changes to carrying capacity or growth rate

but perhaps most importantly …

Birth and death are probabilistic processes

This process is often called 'demographic stochasticity', and is especially important when thinking about small population sizes. As a thought experiment, imagine flipping a fair coin 5 times. The probability of landing on 'heads' is 0.5, but with only 5 trials, the resulting number of heads is far more variable than if we flipped that same coin 100 times. Considering birth and death as probabilistic processes, we can start to understand how we might expect population dynamics to be more variable at small population sizes. That is, the effects of 'demographic stochasticity' are dependent on population density.

This does take us a bit away from the logistic model, as the code below does not consider the influence of carrying capacity k.

What if we treated birth as offspring drawn from a Poisson distribution, with some mean number of offspring λ?

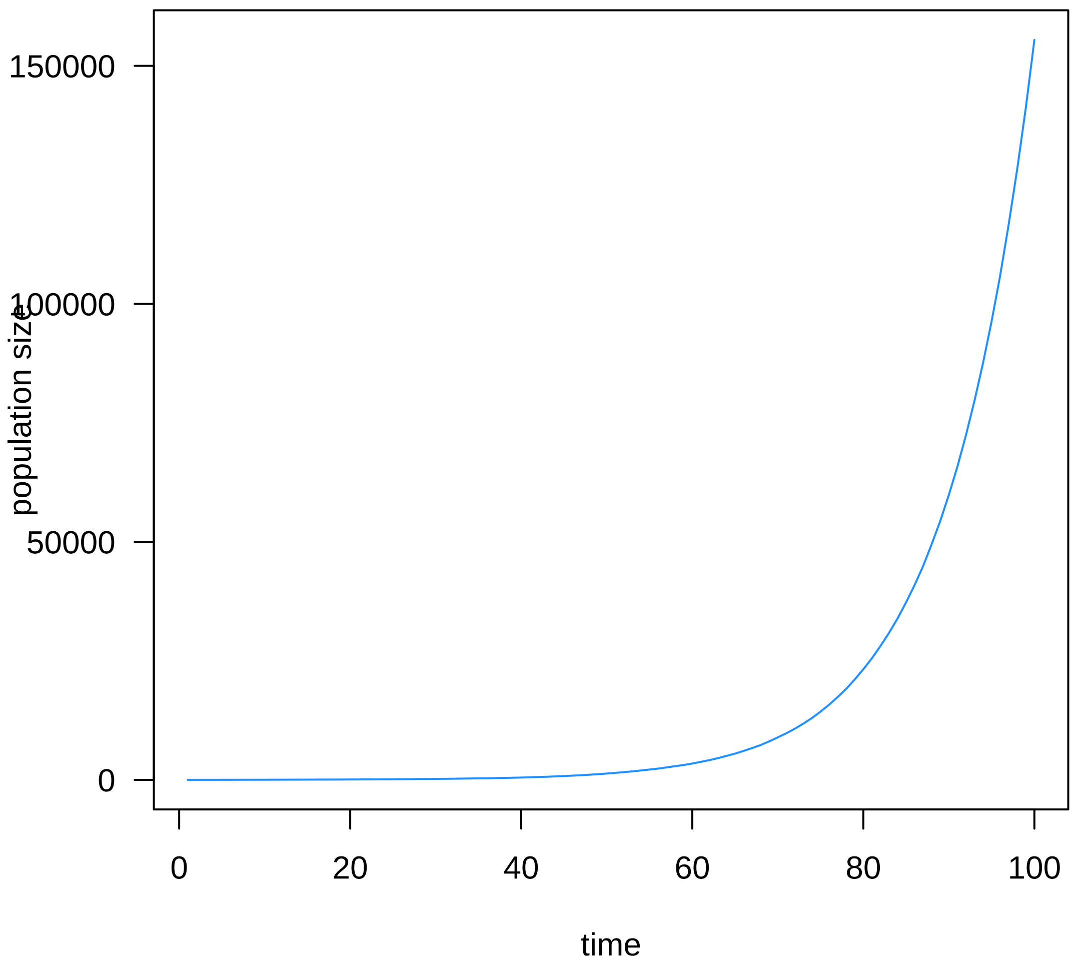
```r
logisticP1 <- function(Nt, b=0.1, d=0.1) {
  births <- sum(rbinom(Nt,1,b))
  deaths <- 0
  pop <- (Nt + births - deaths)
  return(pop)
}
```

So the model above treats birth as drawn from a Poisson distribution, and death as dependent on the population density.

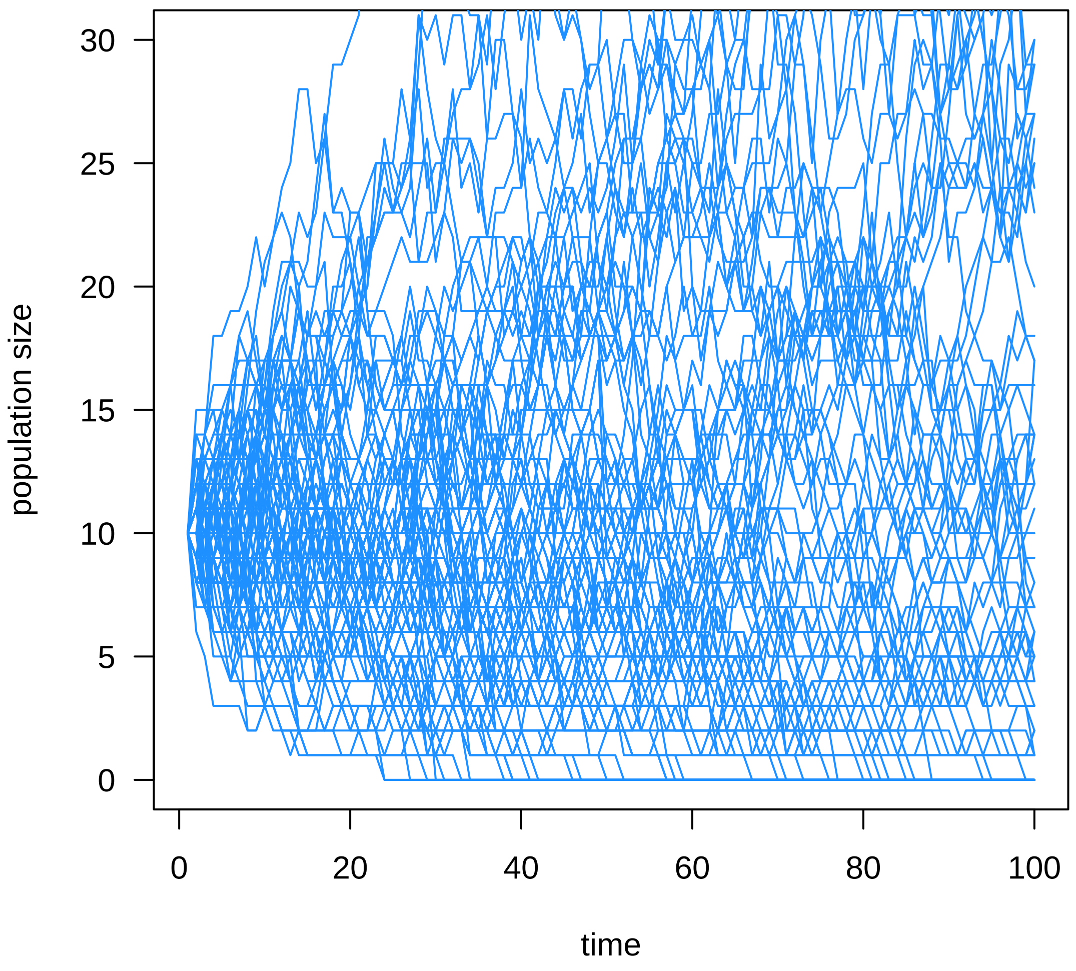
What if we treated death as binomial, with some probability p?

```r
logisticP2 <- function(Nt, b=0.1, d=0.1) {
  births <- sum(rbinom(Nt, 1, b))
  deaths <- sum(rbinom(Nt, 1, d))
  pop <- (Nt + births - deaths)
  return(pop)
}
```

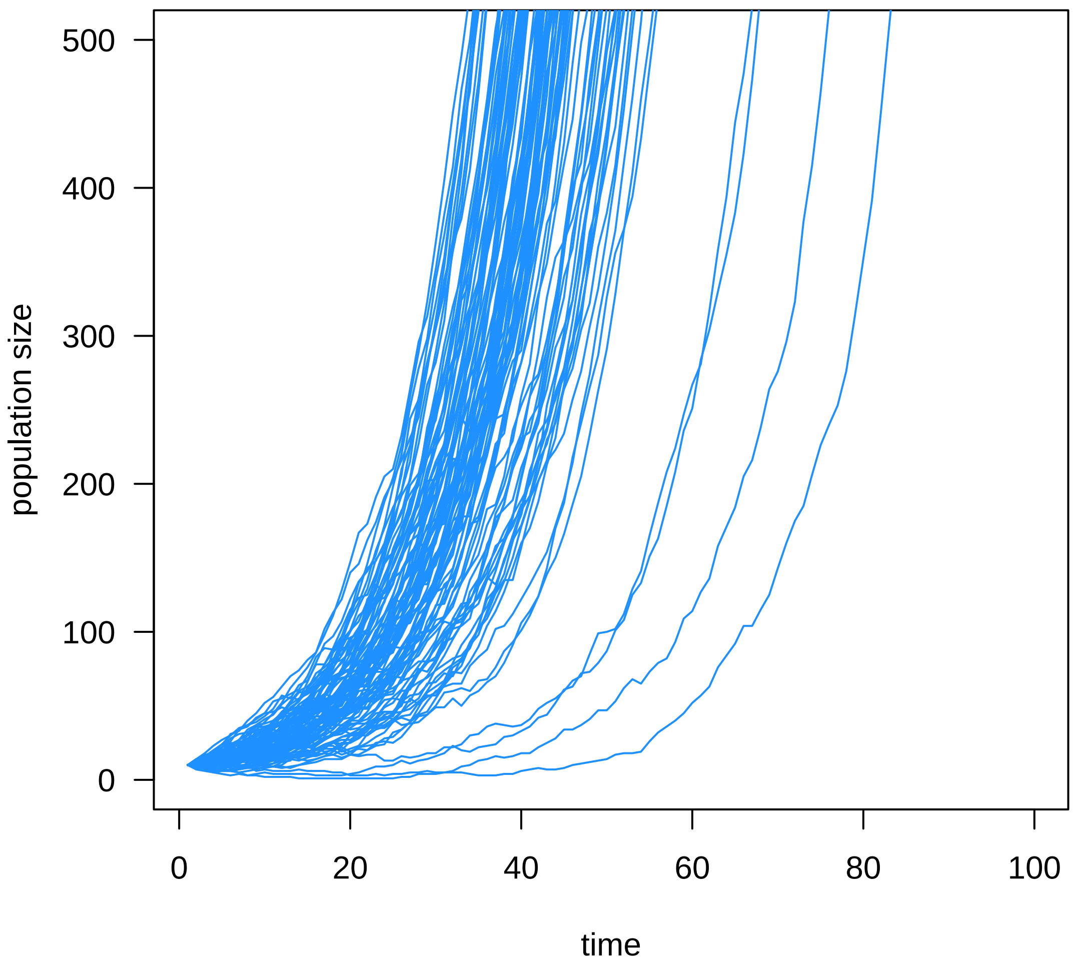
```r
logisticPDynamics <- function(Nt, b, d, steps=1000, mod=logisticP1){
  ret <- c()
  ret[1] <- Nt
  if(length(b) == 1){
    b <- rep(b, steps)
  }
  if(length(d) == 1){
    d <- rep(d, steps)
  }
  for(i in 1:(steps-1)){
    ret[i+1] <- mod(ret[i], b=b[i], d=d[i])
  }
  return(ret)
}


plot(logisticPDynamics(10, b=0.1, d=0.1, mod=logisticP1, steps=100),
  type='l', las=1, ylab='population size',
  xlab='time', col='dodgerblue')
```



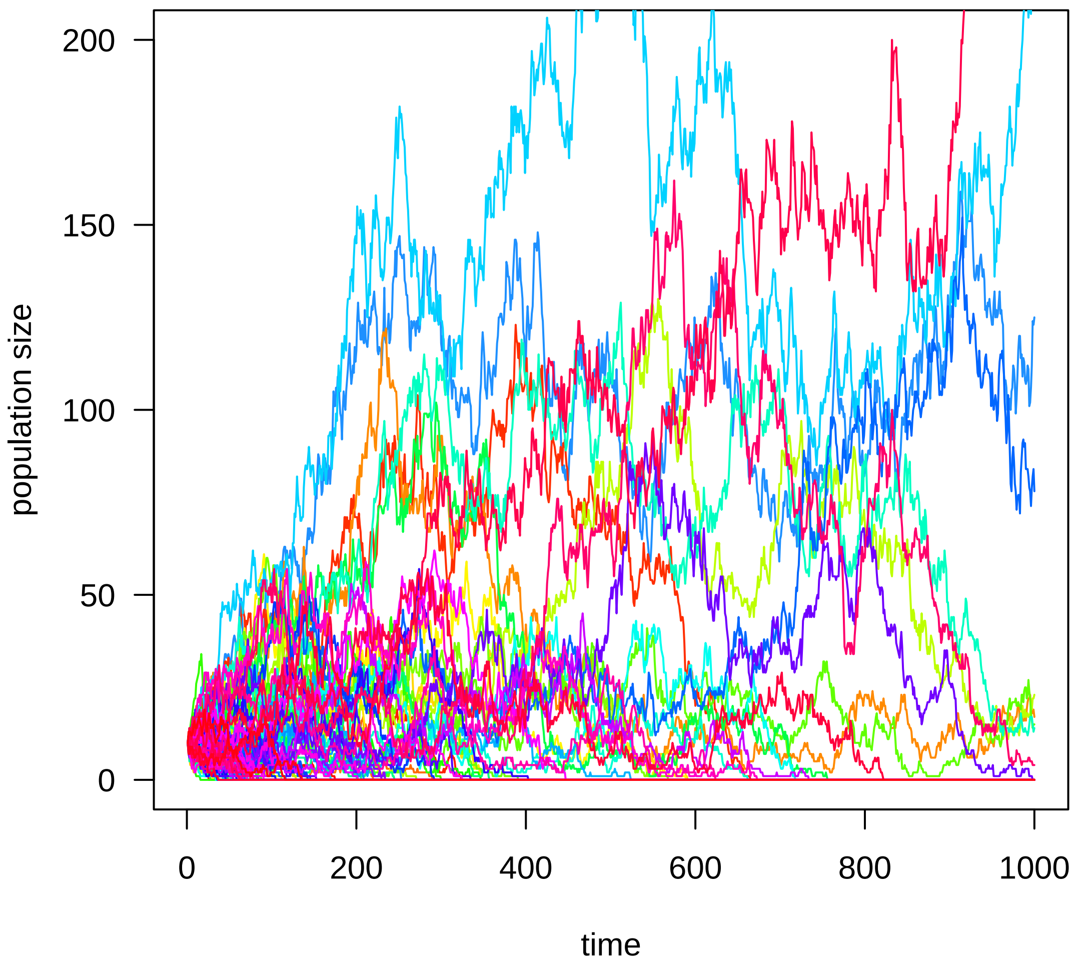
```
# birth death equal
plot(logisticPDynamics(10, b=0.1, d=0.1, mod=logisticP2, steps=100),
  type='l', ylim=c(0,30),
  las=1, ylab='population size', xlab='time',
  col='dodgerblue')
lapply(1:100, function(x){
  lines(logisticPDynamics(10, b=0.1, d=0.1, mod=logisticP2, steps=100), col='dodgerblue')
})
```



```
# birth 2x death
plot(logisticPDynamics(10, b=0.2, d=0.1, mod=logisticP2, steps=100),
  type='l', ylim=c(0,500),
  las=1, ylab='population size', xlab='time',
  col='dodgerblue')
lapply(1:100, function(x){
  lines(logisticPDynamics(10, b=0.2, d=0.1, mod=logisticP2, steps=100), col='dodgerblue')
})
```



```
# 1000 time steps
plot(logisticPDynamics(10, b=0.1, d=0.1, mod=logisticP2, steps=1000),
  type='l', ylim=c(0,200),
  las=1, ylab='population size', xlab='time',
  col='dodgerblue')
lapply(1:100, function(x){
  lines(logisticPDynamics(10, b=0.1, d=0.1, mod=logisticP2, steps=1000), col=rainbow(100)[x])
})
```



How would you incorporate the carrying capacity k into the `logisticP2` function? Show how it influences population dynamics.

Infectious disease modeling

The SIR model is a *compartmental* model of infectious disease, where the goal is to track the fraction of the population that is in each state (susceptible, infected, or recovered). Individuals move between these different states based on some probability of becoming infected (β) and some probability of recovery (γ).

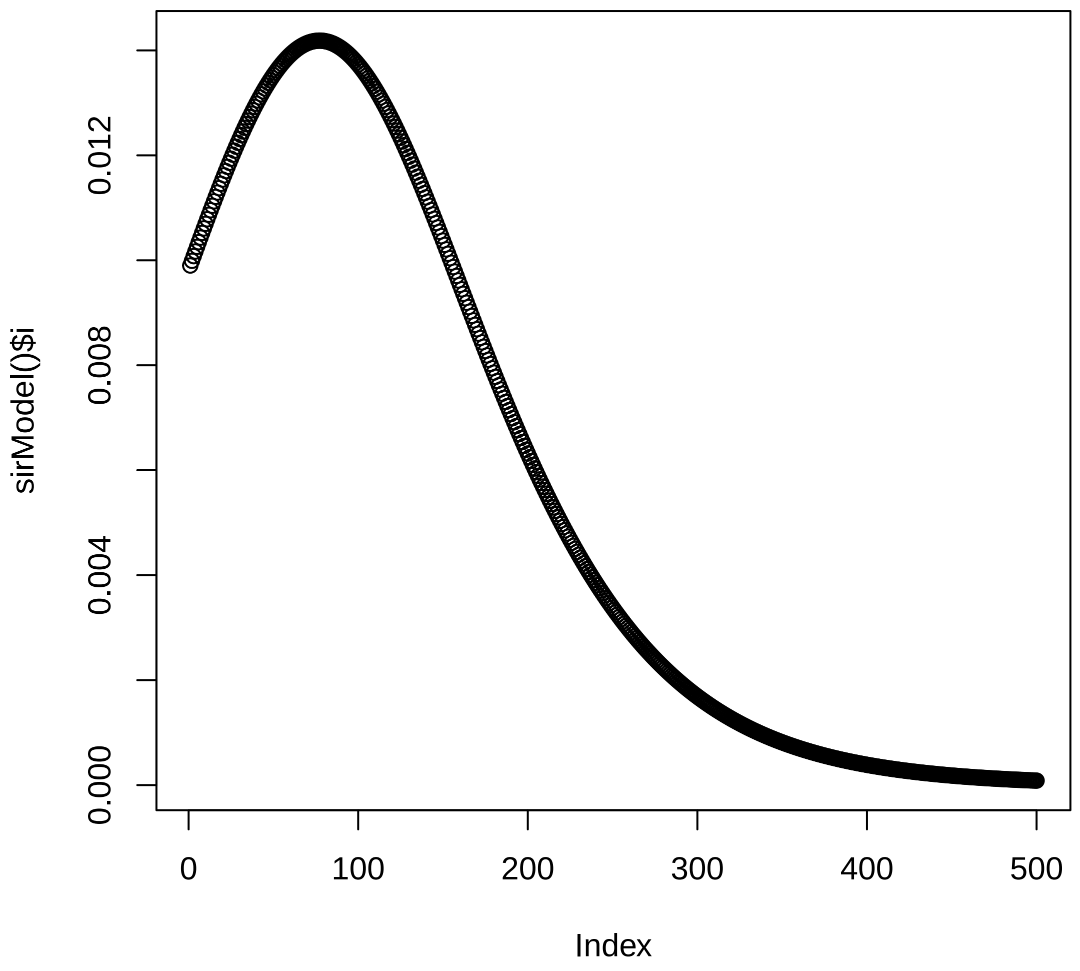
$$S = -\beta SI$$

$$I = \beta SI - \gamma I$$

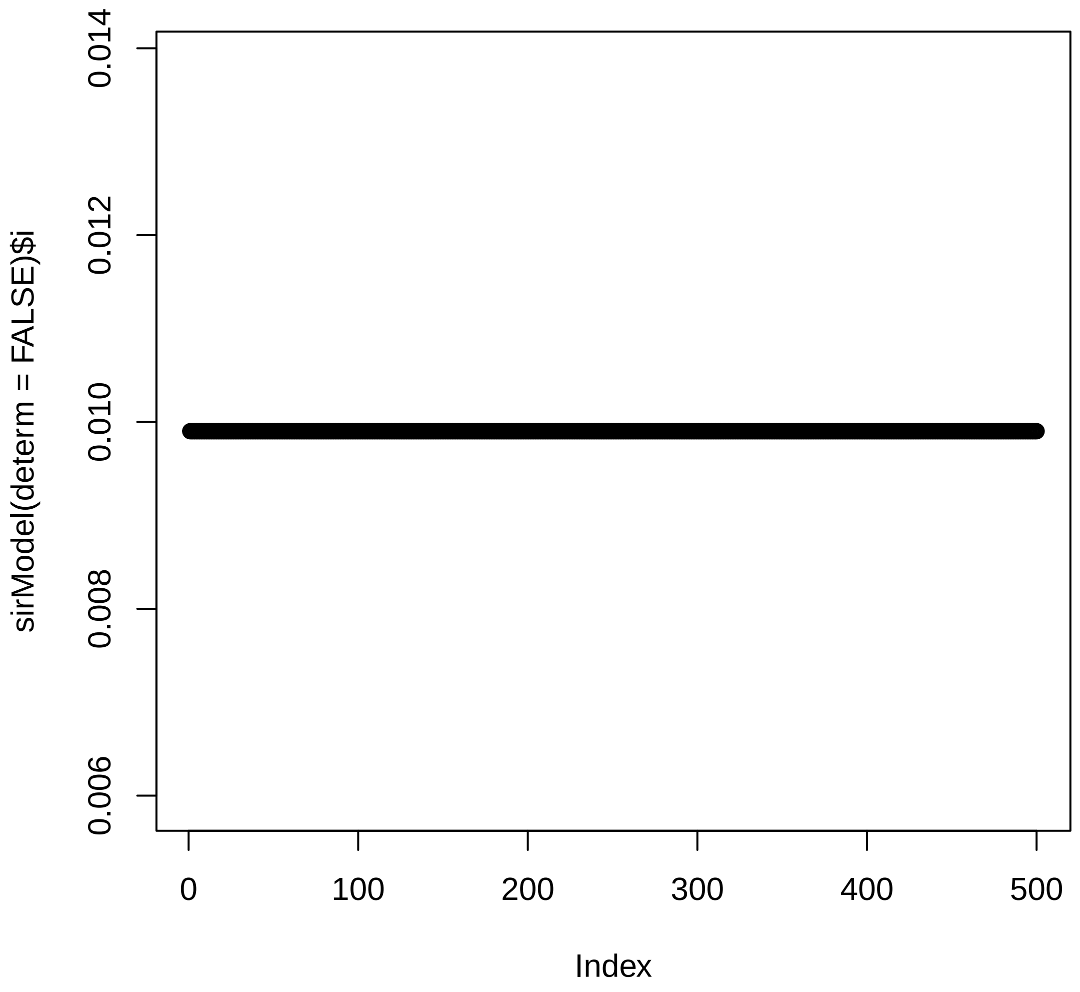
$$R = \gamma I$$

```
sirModel <- function(init=c(100,1,0), beta=0.1, gamma=0.09,
  steps=500, determ=TRUE){
  s <- i <- r <- c()
  s[1] <- init[1]/sum(init)
  i[1] <- init[2]/sum(init)
  r[1] <- init[3]/sum(init)

  if(determ){
    for(z in 2:steps){
      s[z] <- s[z-1] - (beta*s[z-1]*i[z-1])
      i[z] <- i[z-1] + (beta*s[z-1]*i[z-1]) - (gamma*i[z-1])
      r[z] <- r[z-1] + (gamma*i[z-1])
    }
  }
  if(determ==FALSE){
    for(z in 2:steps){
      infection <- (sum(rbinom(s[z-1], 1, beta*i[z-1])))
      recovery <- (sum(rbinom(i[z-1], 1, gamma)))
      s[z] <- s[z-1] - infection
      i[z] <- i[z-1] + infection - recovery
      r[z] <- r[z-1] + recovery
    }
  }
  return(data.frame(s,i,r))
}
```

```
plot(sirModel()$i)
```



```
plot(sirModel(determ=FALSE)$i)
```



sessionInfo

```
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 22.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
## LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3;  LAPACK version 3.10.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/New_York
## tzcode source: system (glibc)
```

```
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] stringr_1.5.0
##
## loaded via a namespace (and not attached):
##  [1] digest_0.6.33   fastmap_1.1.1   xfun_0.39       magrittr_2.0.3
##  [5] glue_1.6.2      knitr_1.43      htmltools_0.5.5 rmarkdown_2.23
##  [9] lifecycle_1.0.3 cli_3.6.1       compiler_4.3.1  highr_0.10
## [13] tools_4.3.1     evaluate_0.21   yaml_2.3.7      rlang_1.1.1
## [17] stringi_1.7.12
```