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]  3.28025994 -0.40512684  0.24706024  2.70575765  1.00818071  0.98328298
##   [7] -0.96114254  0.70308522  0.12035133  1.09105958  0.53410240  1.75794700
## [13]  0.90617909  0.08627741  1.24624641  2.30967806  0.90607337  1.72965262
## [19]  0.25837326 -0.66179362  1.01661246  1.55481599  1.41491170 -0.31969334
## [25] -0.53960177  1.26763016  0.76093520  0.76976010  1.54592621 -0.24122310
## [31] -0.85909007  1.10732570  1.15812913  0.65632774  3.19308510 -0.81368696
## [37]  2.11454527  2.97782954  3.12708365  1.48499639  3.09056668  1.62250609
## [43]  1.40689129  1.44130110  0.37077268  0.33252369  0.91239400  2.19661855
## [49]  1.48692692  0.28737256  1.87061974  0.32536607  0.80905591  1.41591491
## [55]  1.16358022  0.03173189  1.02030638  1.51531749  1.69376610  0.17601435
## [61]  2.47931219  0.62262985  0.92651805  1.73687990  0.08935934  0.55165197
## [67]  0.78441144  0.65974957 -0.04198136 -0.11941752  0.09963213  1.54020195
## [73]  0.55427184  0.67100816  1.12068657  2.48451988  1.69765598  1.31851115
## [79]  0.84495306  1.46818408  1.09273295  1.15910327  0.79060878  1.80665163
## [85]  0.29687600  0.47004417  1.55687528 -0.06546898  1.40783473  1.43844812
## [91]  2.64328441  1.62746261  2.10596445  1.17000740  1.42753204  1.61361928
## [97]  1.87360943  0.81760790  1.52072328  0.51193625
```

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

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

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

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

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

```
##   [1]  4.457272 17.297431  6.415969  9.366698  6.209018 14.760675 19.462991
##   [8] 15.901740 10.887645 12.561765 15.766536 10.720933 15.819442 17.807213
##  [15] 16.982689  1.218036 11.153642 18.148921 17.374117  3.830975  8.488651
##  [22]  7.192203 17.580261  1.934695  3.298959 11.548412 16.891821 19.497043
##  [29] 12.716458 12.252551  6.665927 18.780392  8.091907  7.905568  6.376382
##  [36] 12.890200  6.025618 15.130347  7.365225  5.394556  7.947370 11.789996
##  [43] 12.398582 11.596268 13.344525 16.403494 17.048228  1.910391 19.854194
##  [50]  5.293265 16.496134  9.190277  5.125587  4.998241 12.144360 12.517437
##  [57] 15.874403 10.437823  3.763191  1.189995 11.810697  3.339779 18.492191
##  [64]  5.475067  3.511879 12.281350 19.047958  7.209246  2.544201  9.444325
##  [71] 18.231396 17.553068 19.678839  2.794819  5.215411 14.605903 15.917572
##  [78] 19.746534  5.442467  3.701755  5.105681 14.233109  2.623442  7.022629
##  [85]  7.226645  7.565368 12.377365 15.986583  5.481763  4.907060 18.347767
##  [92]  4.979769  1.286370 16.553211 15.695860  7.909720 19.479555 10.218247
##  [99]  8.112052 10.486930
```

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

```
##   [1] 1.58732023 0.38480006 3.25890569 1.29380704 0.16072868 0.03718568
##   [7] 6.23551230 0.80409299 0.68432720 1.49361375 2.82255792 1.22372976
##  [13] 0.01935336 1.09459420 1.24691459 0.19051736 2.93592239 0.14013254
##  [19] 1.11622269 0.33400485 0.27820100 0.01870822 1.19601650 1.56649191
##  [25] 0.80673233 1.18368911 1.40298699 0.03364638 1.24019194 0.78656666
##  [31] 1.40163980 1.12260153 2.63530420 0.40449334 0.44651239 0.68618524
##  [37] 0.24471282 0.93229163 0.70992669 1.79126112 1.19399700 0.11471331
##  [43] 0.50894746 6.06111236 0.37762580 0.24155842 0.96269903 0.21380407
##  [49] 0.07531730 1.11883345 3.07226298 0.16736285 0.30327009 0.07223826
##  [55] 1.89454392 0.54331910 0.97436752 1.58303539 0.12441756 0.97475463
##  [61] 0.19791256 1.50586913 0.43067163 2.75464306 0.33503916 1.55453779
##  [67] 0.25350535 0.45406611 0.42546447 1.00780755 0.82944259 2.90394045
##  [73] 0.65217477 0.29303988 0.19003185 2.27238613 0.72506243 0.12039229
##  [79] 1.04355446 0.54935948 0.97521075 1.69399069 0.44172460 1.58600258
##  [85] 3.06674282 1.97929750 1.75640996 0.15793269 0.88336116 0.91894126
##  [91] 1.59951809 3.86069716 1.31855670 1.94282189 1.57990358 1.42448607
##  [97] 1.29402307 2.85706269 0.95519771 1.76654309
```

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
```

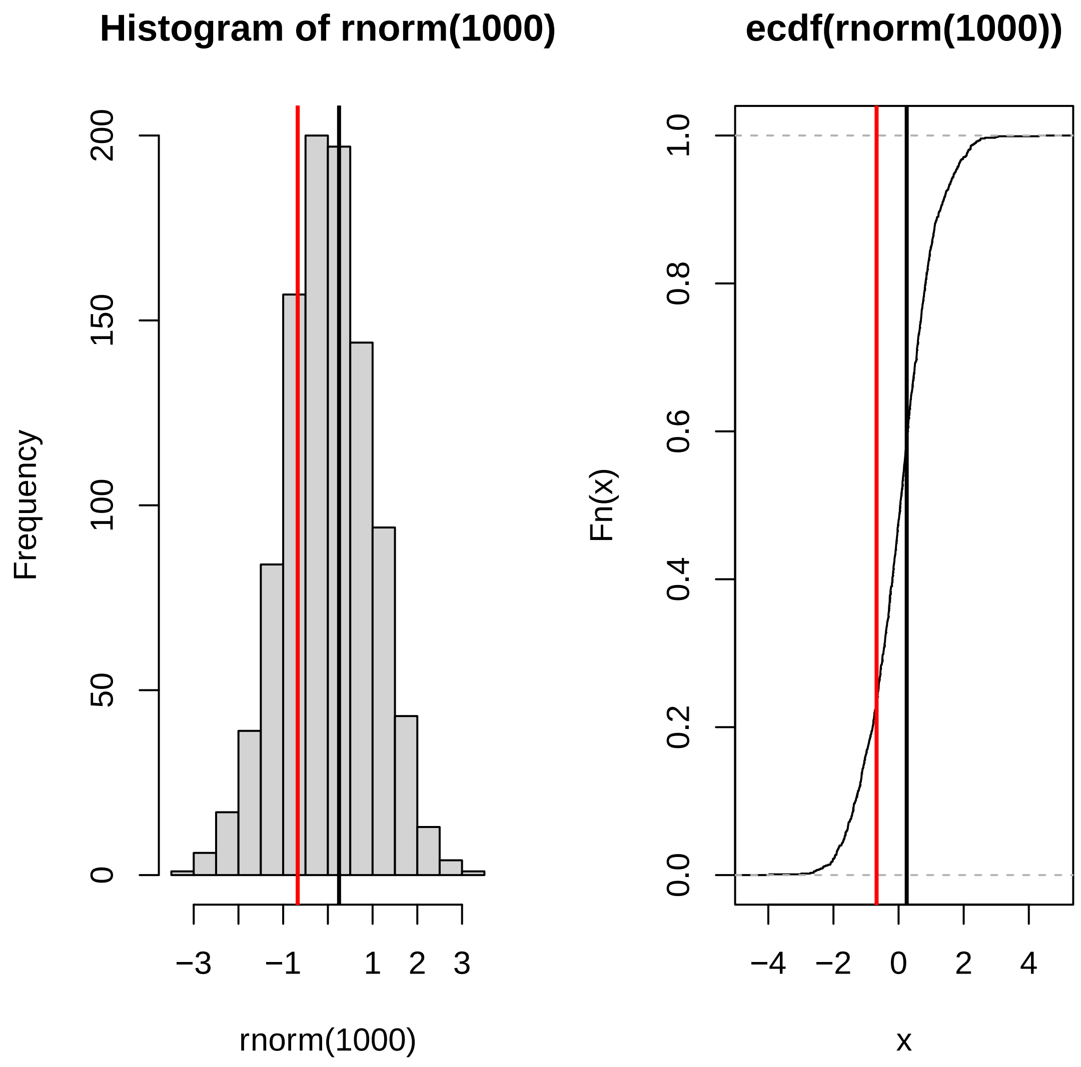
```
qnorm(0.25, mean=0, sd=1)
```

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

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

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

```
par(mfrow=c(1,2))
hist(rnorm(1000))
abline(v=0.25, lwd=2)
abline(v=-0.6744898, col='red', lwd=2)
plot(ecdf(rnorm(1000)))
abline(v=0.25, lwd=2)
abline(v=-0.6744898, col='red', lwd=2)
```



The last is `d----`, which we will go ahead and ignore for now, as the `r---`, `p----`, and `q----` variants tend to be a bit more useful for thinking about statistical testing.

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

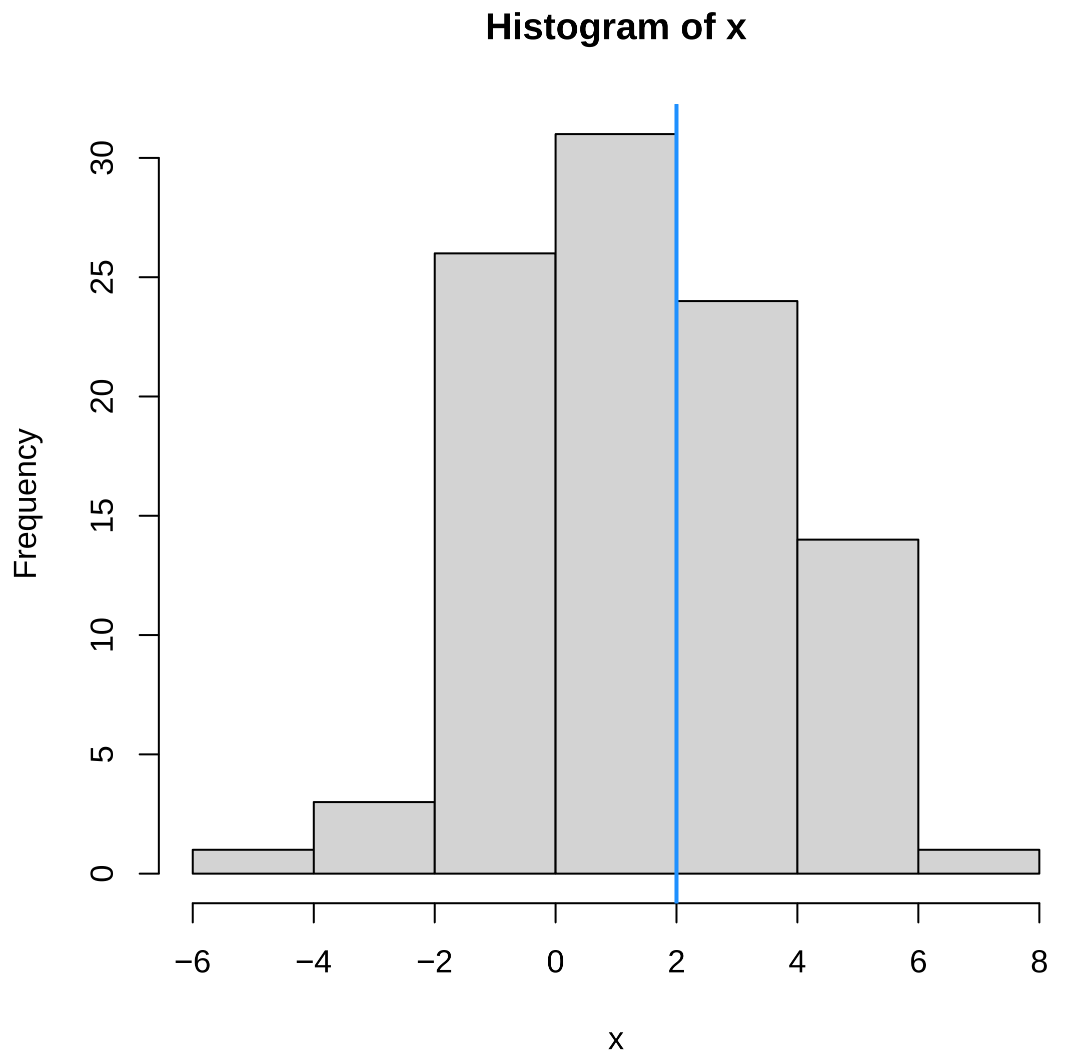
The t-test can be used to test for differences between two groups of data, or of one group of data and some mean μ. It is a comparison of means, so we implicitly assume no differences in variance or distributional shape between the two groups.

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 ($\mu = 2$).

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

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



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

```
##
##  One Sample t-test
##
## data:  x
## t = -2.9776, df = 99, p-value = 0.003652
## alternative hypothesis: true mean is not equal to 2
## 95 percent confidence interval:
##  0.896879 1.779143
## sample estimates:
## mean of x
##  1.338011
```

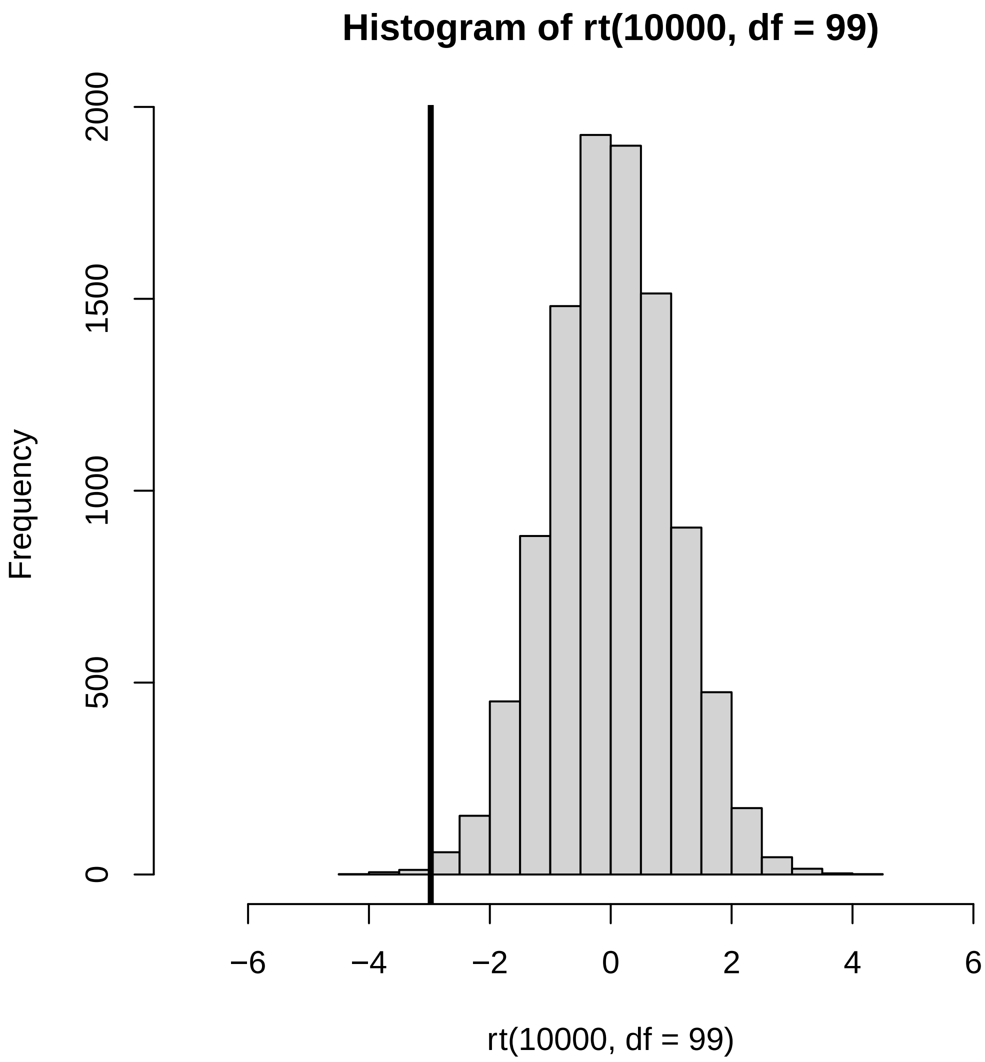
```
tt <- (mean(x) - mu) / (sd(x) / sqrt(length(x)))
pt(tt, df=length(x)-1)
```

```
## [1] 0.001826186
```

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

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

```
## [1] 0.003652372
```

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



Two-sample

X1-X2 / (sp)

sp = sqrt((varx1 + varx2) / 2)

sp is pooled variance

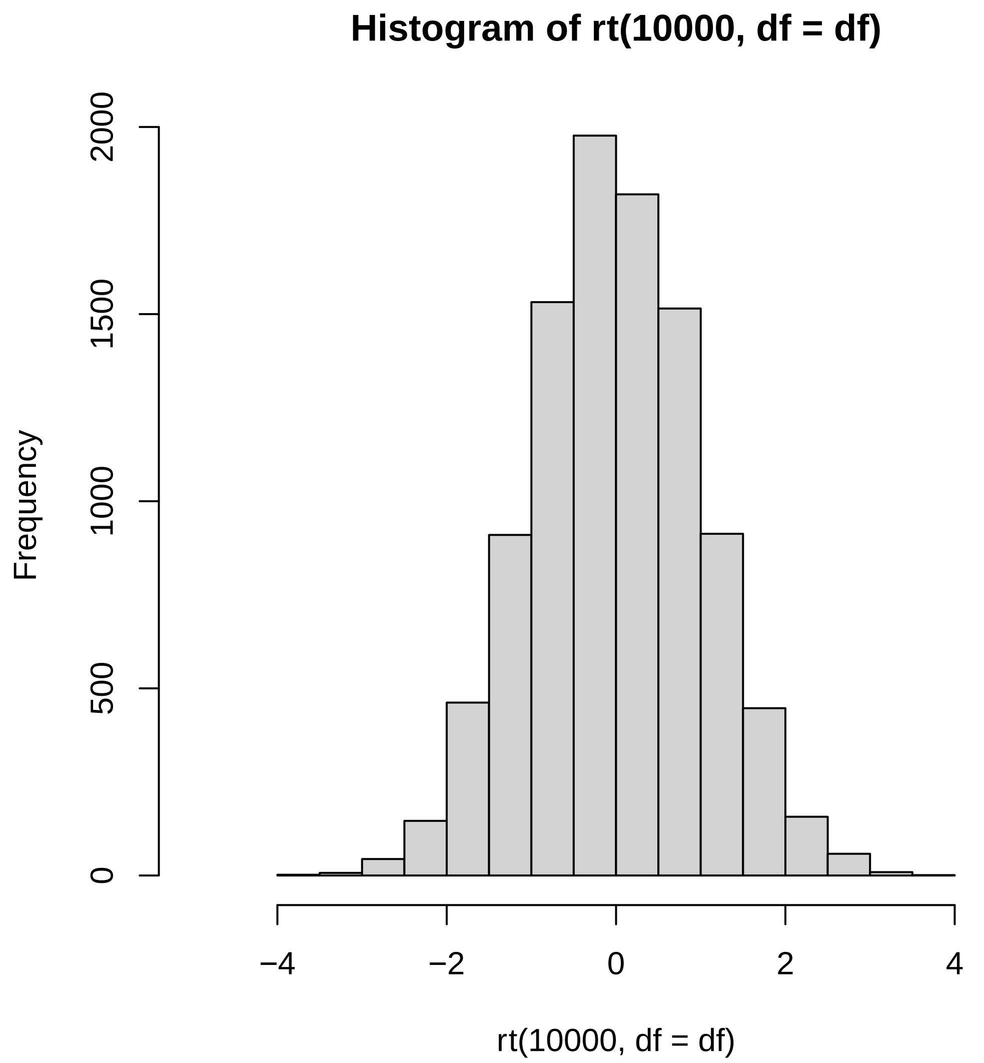
```r
x1 <- rnorm(100, 1, 2)
x2 <- rnorm(100, 2, 1)

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

```
##
##  Two Sample t-test
##
## data:  x1 and x2
## t = -5.6696, df = 198, p-value = 5.011e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.6906277 -0.8180481
## sample estimates:
## mean of x mean of y
##  0.538058  1.792396
```

```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] 5.010775e-08
```

```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] 2.505388e-08
```

```
pnorm(tt2)
```

```
## [1] 7.157752e-09
```

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
```

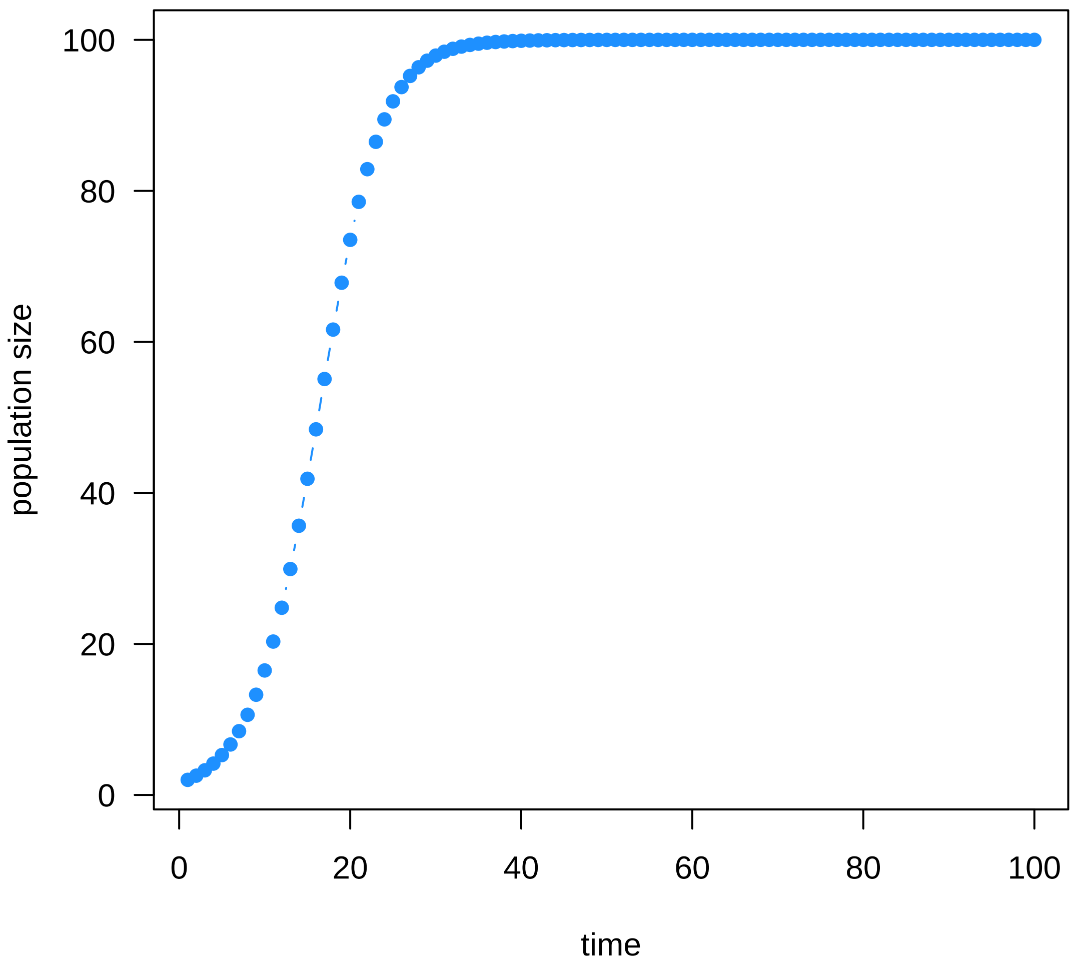
```
## [1] 4.337367e-09
```

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?

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

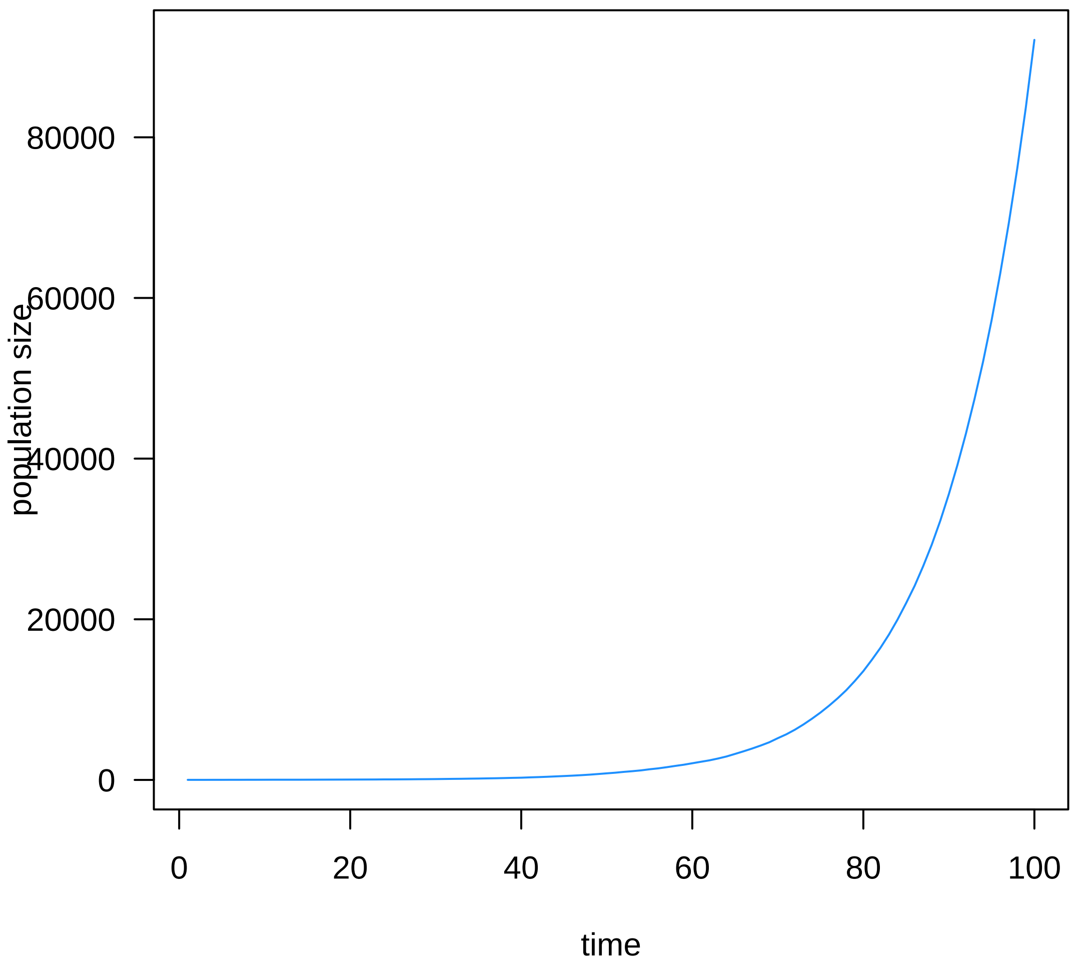
```r
  pop <- (Nt + births - deaths)
  return(pop)
}
```

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

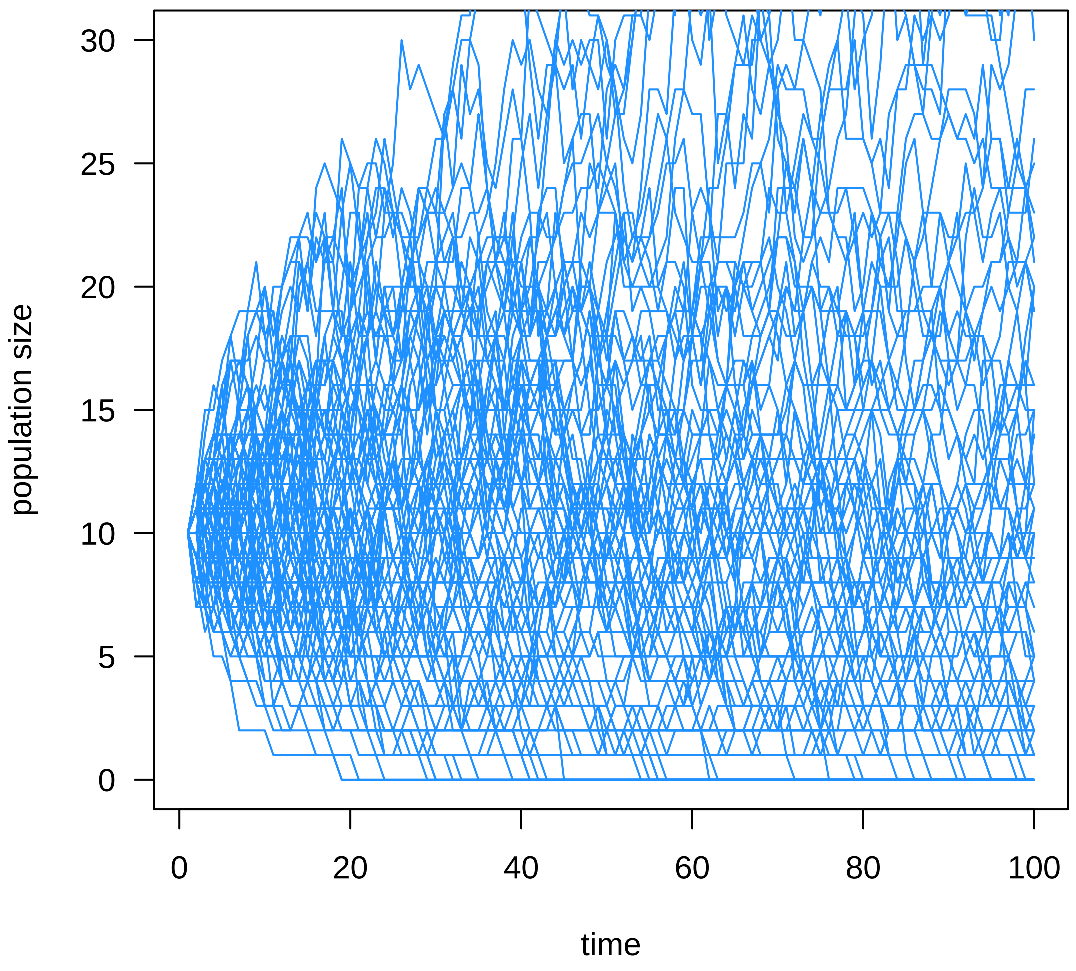
What if we treated death as binomial, with some probability p?

```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)
}
```

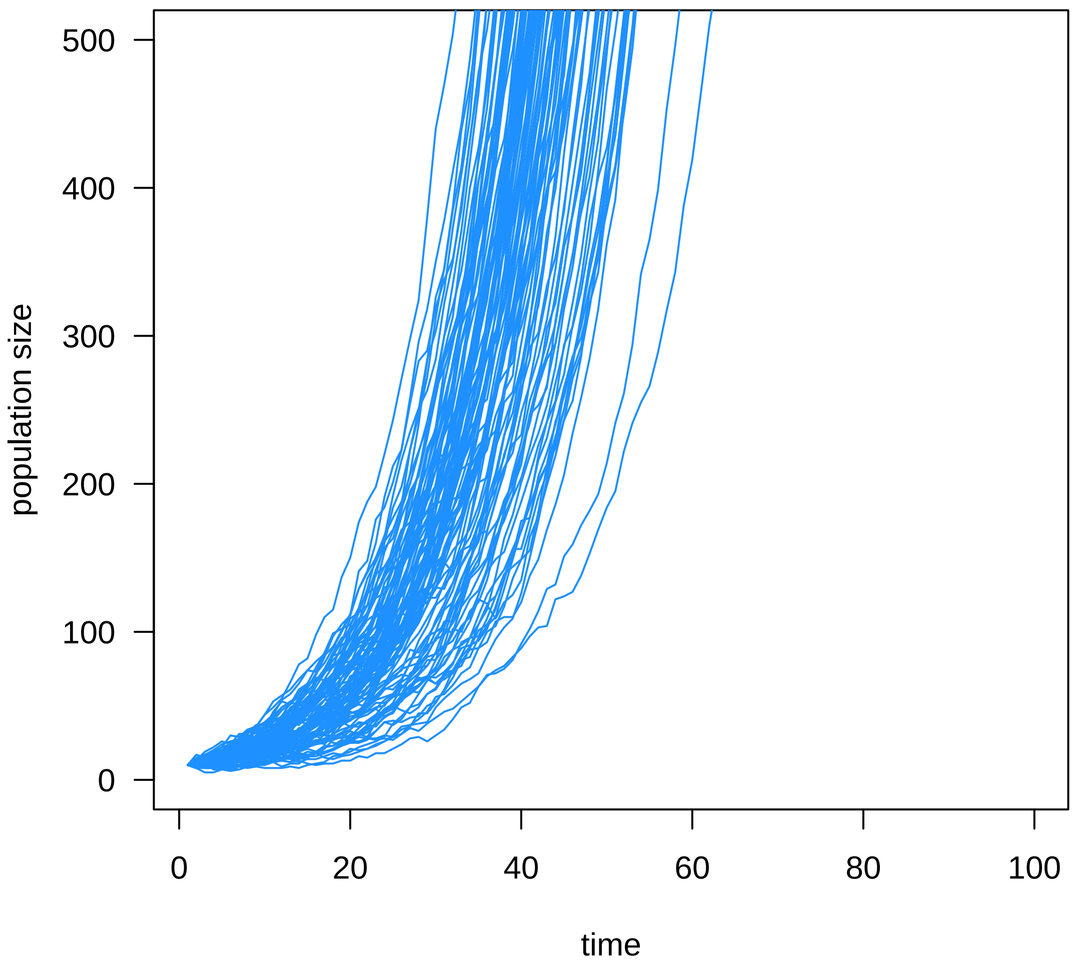
```r
logisticPDynamics <- function(Nt, b, d, steps=1000, mod=logisticP1){
  ret <- c()
  ret[1] <- Nt
  if(length(b) == 1){
    b <- rep(b, steps)
  }
  if(length(d) == 1){
    d <- rep(d, steps)
  }
  for(i in 1:(steps-1)){
    ret[i+1] <- mod(ret[i], b=b[i], d=d[i])
  }
  return(ret)
}


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



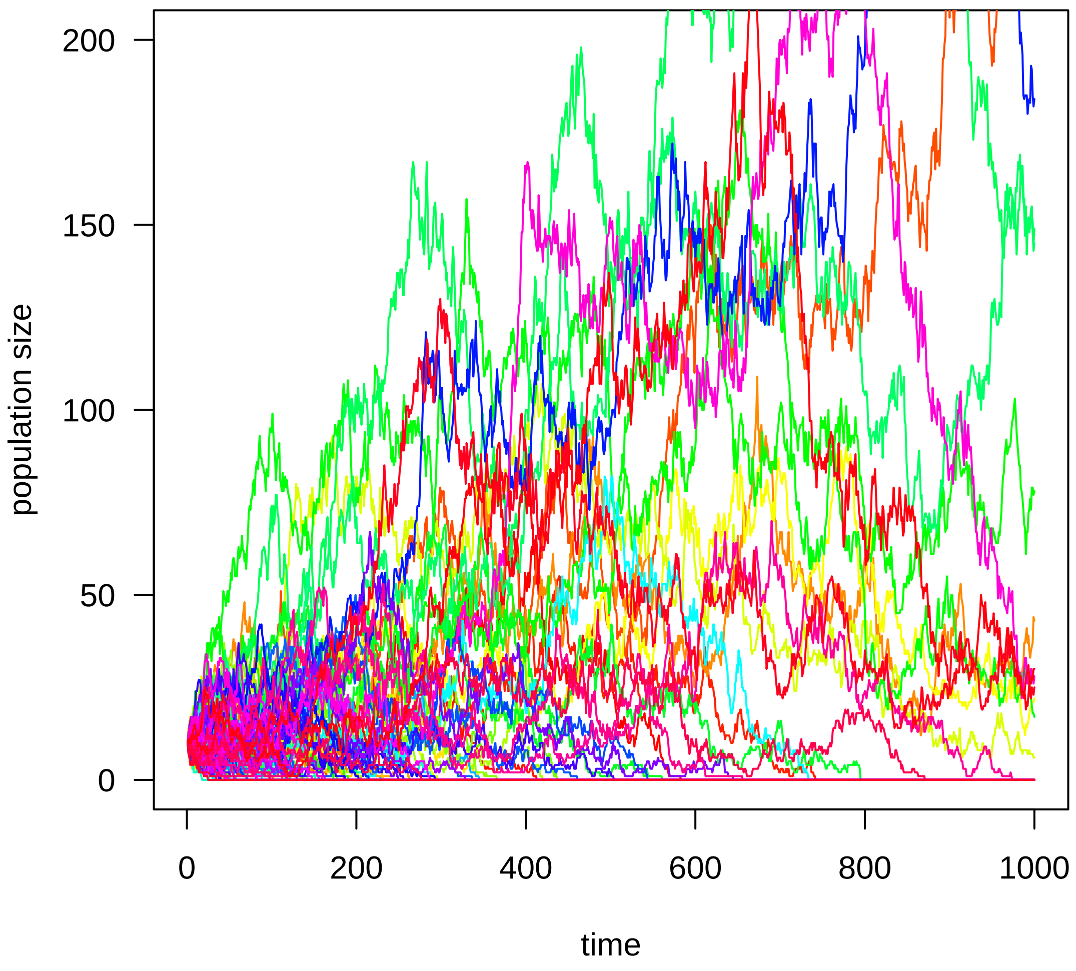
```
# birth death equal
plot(logisticPDynamics(10, b=0.1, d=0.1, mod=logisticP2, steps=100),
  type='l', ylim=c(0,30),
  las=1, ylab='population size', xlab='time',
  col='dodgerblue')
lapply(1:100, function(x){
  lines(logisticPDynamics(10, b=0.1, d=0.1, mod=logisticP2, steps=100), col='dodgerblue')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



```r
# 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).

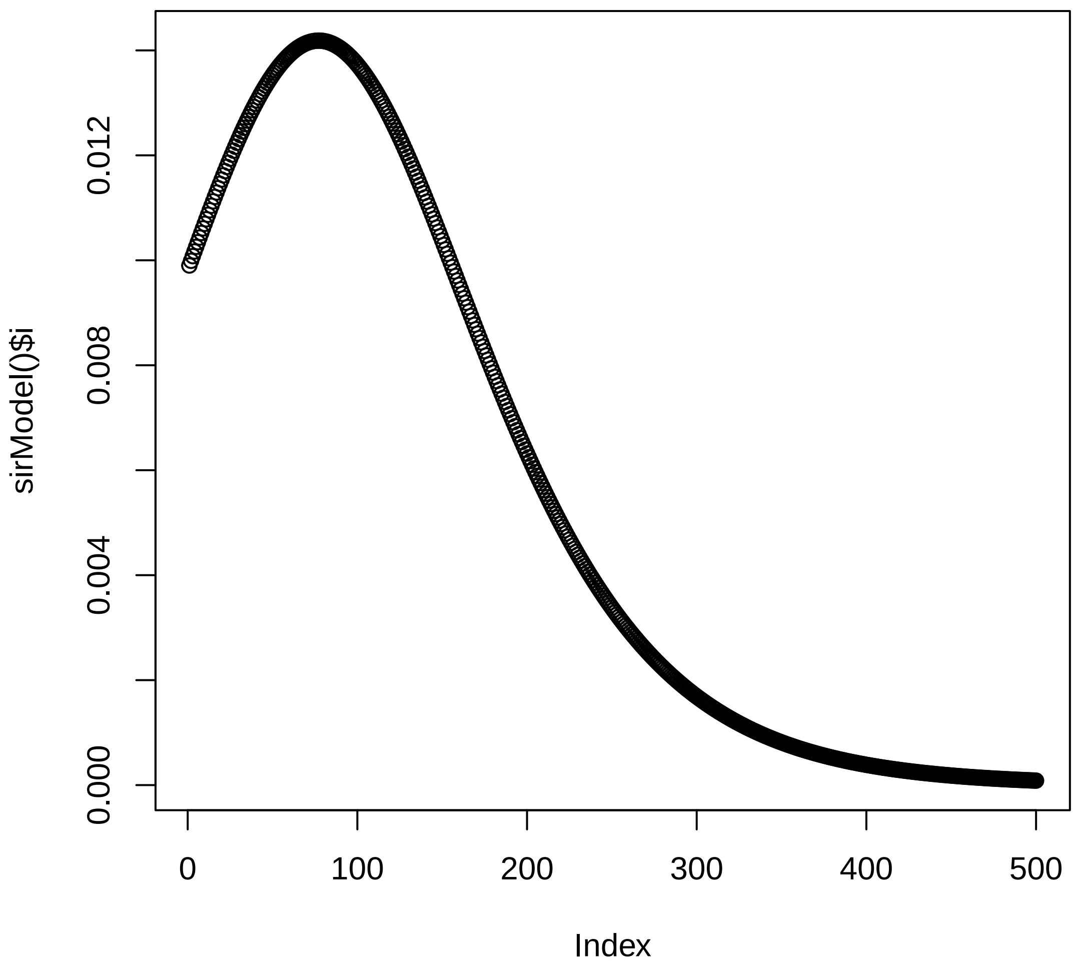
$$S = -\beta SI$$

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

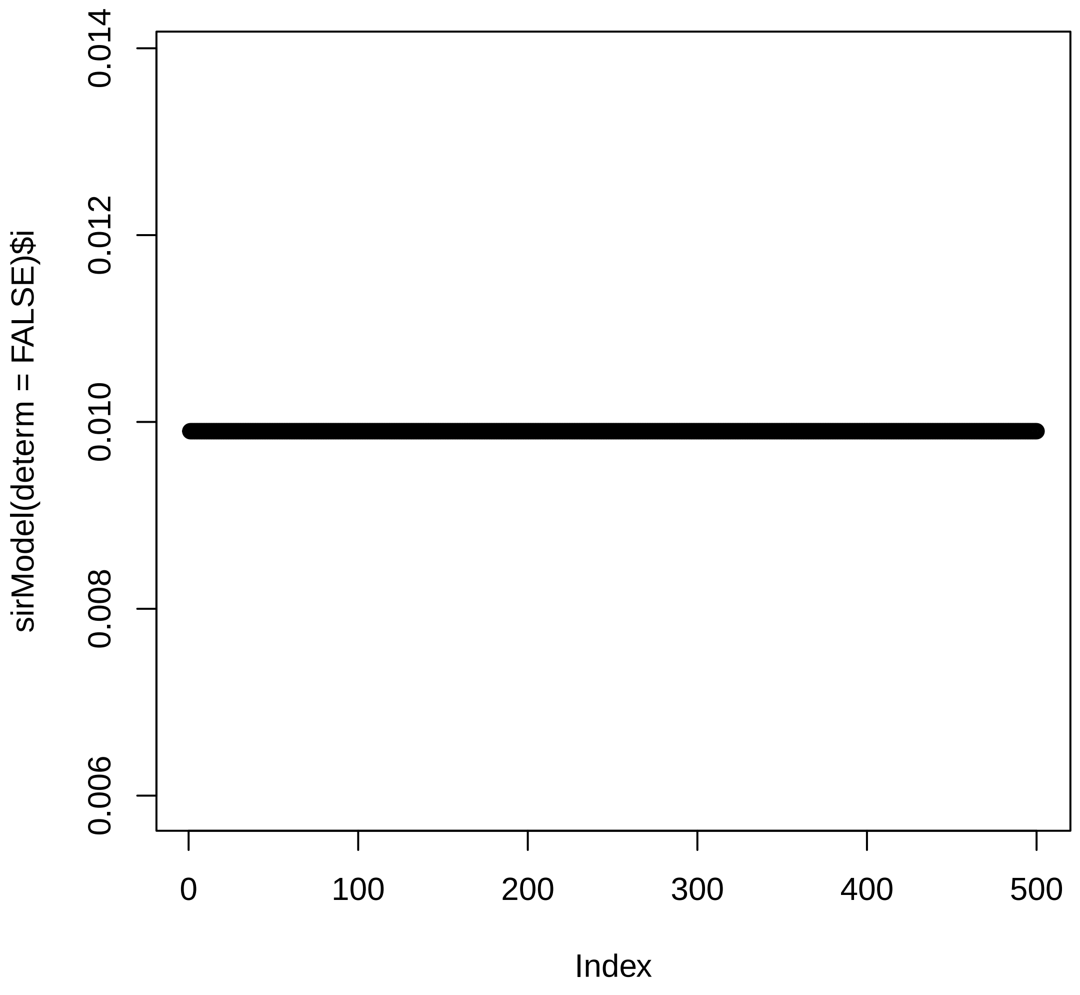
$$R = \gamma I$$

```
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 22.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.10.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.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] ROCR_1.0-11    geodata_0.5-8  terra_1.7-39   maps_3.4.1     rgbif_3.7.7
## [6] gbm_2.1.8.1    rmarkdown_2.23
##
## loaded via a namespace (and not attached):
##  [1] viridis_0.6.3     utf8_1.2.3        generics_0.1.3    xml2_1.3.5
##  [5] stringi_1.7.12    lattice_0.21-8    httpcode_0.3.0    digest_0.6.33
##  [9] magrittr_2.0.3    evaluate_0.21     grid_4.3.1        fastmap_1.1.1
## [13] plyr_1.8.8        jsonlite_1.8.7    Matrix_1.6-0      whisker_0.4.1
## [17] crul_1.4.0        tinytex_0.45      survival_3.5-5    urltools_1.7.3
## [21] gridExtra_2.3     httr_1.4.6        fansi_1.0.4       viridisLite_0.4.2
## [25] scales_1.2.1      oai_0.4.0         codetools_0.2-19  lazyeval_0.2.2
## [29] cli_3.6.1         rlang_1.1.1       triebeard_0.4.1   munsell_0.5.0
## [33] splines_4.3.1     yaml_2.3.7        parallel_4.3.1    tools_4.3.1
## [37] dplyr_1.1.2       colorspace_2.1-0  ggplot2_3.4.2     curl_5.0.1
## [41] vctrs_0.6.3       R6_2.5.1          lifecycle_1.0.3   stringr_1.5.0
## [45] pkgconfig_2.0.3   pillar_1.9.0      gtable_0.3.3      data.table_1.14.8
## [49] glue_1.6.2        Rcpp_1.0.11       xfun_0.39         tibble_3.2.1
## [53] tidyselect_1.2.0  highr_0.10        knitr_1.43        htmltools_0.5.5
## [57] compiler_4.3.1
```