ESMA 5015 Simulation

Contents

1 R

For a detailed introduction to R you can read the material of my course Computing with R

1.1 Installation and Updating

1.1.1 Installing R

You can get a free version of R for your computer from a number of sources. The download is about 70MB and setup is fully automatic. Versions for several operating systems can be found on the R web site

https://cran.r-project.org

Note

- the one item you should change from the defaults is to install R into a folder under the root, aka C:\R
- You might be asked at several times whether you want to do something (allow access, run a program, save a library, . . .), always just say yes!
- You will need to connect to a reasonably fast internet for these steps.
- This will take a few minutes, just wait until the > sign appears.

FOR MAC OS USERS ONLY

There are a few things that are different from MacOS and Windows. Here is one thing you should do:

Download XQuartz - XQuartz-2.7.11.dmg
Open XQuartz
Type the letter R (to make XQuartz run R)
Hit enter Open R Run the command .First()
Then, every command should work correctly.

1.1.2 RStudio

We will run R using an interface called **RStudio**. You can download it at RStudio.

1.1.3 Updating

R releases new versions about every three months or so. In general it is not necessary to get the latest version every time. Every now and then a package won't run under the old version, and then it is time to do so. In essence this just means to install the latest version of R from CRAN. More important is to now also update ALL your packages to the latest versions. This is done simply by running

```r
update.packages(ask=FALSE, dependencies=TRUE)
```

1.2 R Markdown, HTML and Latex

1.2.1 R Markdown

R Markdown is a program for making dynamic documents with R. An R Markdown document is written in *markdown*, an easy-to-write plain text format with the file extension .Rmd. It can contain chunks of embedded R code. It has a number of great features:

- easy syntax for a number of basic objects
- code and output are in the same place and so are always synced
- several output formats (html, latex, word)

In recent years I (along with many others) who work a lot with R have made Rmarkdown the basic way to work with R. So when I work on a new project I immediately start a corresponding R markdown document.

---

to start writing an R Markdown document open RStudio, File > New File > R Markdown. You can type in the title and some other things.

The default document starts like this:

```
---
title: “My first R Markdown Document”
author: “Dr. Wolfgang Rolke”
date: “April 1, 2018”
output: html_document
---
```

This follows a syntax called YAML (also used by other programs). There are other things that can be put here as well, or you can erase all of it.

YAML stands for *Yet Another Markup Language*. It has become a standard for many computer languages to describe different configurations. For details go to yaml.org

Then there is other stuff you should erase. Next File > Save. Give the document a name with the extension .Rmd

1.2.2 Basic R Markdown Syntax

for a list of the basic syntax go to

https://rmarkdown.rstudio.com/articles_intro.html

or to

https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

1.2.3 Embedded Code

There are two ways to include code chunks (yes, that's what they are called!) into an R Markdown document:

1.2.3.1 stand alone code

simultaneously enter CTRL-ALT-i and you will see this:

“‘{r}

“‘

Here ‘ is the *backtick*, on most keyboards on the key in the upper left below Esc.

you can now enter any R code you like:

“‘{r}
x <- rnorm(10)
mean(x)
“‘

which will appear in the final document as

```
x <- rnorm(10)
mean(x)
```

Actually, it will be like this:

```
x <- rnorm(10)
mean(x)
```

```
## [1] -0.09751059
```

so we can see the result of the R calculation as well. The reason it didn't appear like this before was that I added the argument eval=FALSE:

“‘{r eval=FALSE}
x <- rnorm(10)
mean(x)
“‘

which keeps the code chunk from actually executing (aka *eval*uating). This is useful if the code takes along time to run, or if you want to show code that is actually faulty, or …

there are a number of useful arguments:

- eval=FALSE (shows but doesn't run the code)

- echo=FALSE (the code chunk is run but does not appear in the document)
- warning=FALSE (warnings are not shown)
- message=FALSE (messages are not shown)
- cache=TRUE (code is run only if there has been a change, useful for lengthy calculations)
- error=TRUE (if there are errors in the code R normally terminates the parsing (executing) of the markdown document. With this argument it will ignore the error, which helps with debugging)

1.2.3.2 inline code

here is a bit of text:

... and so the mean was -0.0975106.

Now I didn't type in the number, it was done with the chunk `` `r mean(x)` ``.

Many of these options can be set globally, so they are active for the whole document. This is useful so you don't have to type them in every time. I have the following code chunk at the beginning of all my Rmd files:

```r
library(knitr)
opts_chunk$set(fig.width=6, fig.align = "center",
      out.width = "70%", warning=FALSE, message=FALSE)
```

We have already seen the message and warning options. The other one puts any figure in the middle of the page and sizes it nicely.

If you have to override these defaults just include that in the specific chunk.

1.2.4 Creating Output

To create the output you have to "knit" the document. This is done by clicking on the *knit* button above. If you click on the arrow you can change the output format.

1.2.5 HTML vs Latex(Pdf)

In order to knit to pdf you have to install a latex interpreter. My suggestion is to use Miktex, but if you already have one installed it might work as well.

There are several advantages / disadvantages to each output format:

- HTML is much faster

- HTML looks good on a webpage, pdf looks good on paper
- HTML needs an internet connection to display math, pdf does not
- HTML can use both html and latex syntax, pdf works only with latex (and a little bit of html)

I generally use HTML when writing a document, and use pdf only when everything else is done. There is one problem with this, namely that a document might well knit ok to HTML but give an error message when knitting to pdf. Moreover, those error messages are weird! Not even the line numbers are anywhere near right. So it's not a bad idea to also knit to pdf every now and then.

As far as this class is concerned, we will use HTML exclusively.

1.2.6 Tables

One of the more complicated things to do in R Markdown is tables. For a nice illustration look at

https://stackoverflow.com/questions/19997242/simple-manual-rmarkdown-tables-that-look-good-in-html-p

My preference is to generate a data frame and the use the *kable.nice* function:

```r
Gender <- c("Male", "Male", "Female")
Age <- c(20, 21, 19)
kable.nice(data.frame(Gender, Age))
```

Gender	Age
Male	20
Male	21
Female	19

probably with the argument echo=FALSE so only the table is visible.

1.2.7 LATEX

You have not worked with latex (read: latek) before? Here is your chance to learn. It is well worthwhile, latex is the standard document word processor for science. And once you get used to it, it is WAY better and easier than (say) Word.

A nice list of common symbols is found on https://artofproblemsolving.com/wiki/index.php/LaTeX:Symbols.

1.2.7.1 inline math

A LATEX expression always starts and ends with a $ sign. So this line:

The population mean is defined by $E[X] = \int_{-\infty}^{\infty} xf(x)dx$.

was done with the code

The population mean is defined by $E[X] = \int_{-\infty}^{\infty} xf(x) dx$.

1.2.7.2 displayed math

sometimes we want to highlight a piece of math:

The population mean is defined by

$$E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

this is done with two dollar signs:

```
$$
E[X] = \int_{-\infty}^{\infty} xf(x) dx
$$
```

1.2.7.3 multiline math

say you want the following in your document:

$$\begin{aligned}
&E[X] = \int_{-\infty}^{\infty} xf(x) dx=\\
&\int_{0}^{1} x dx = \frac12 x^2 |_0^1 = \frac12
\end{aligned}$$

for this to display correctly in HTML and PDF you need to use the format

```
$$
\begin{aligned}
&E[X] = \int_{-\infty}^{\infty} xf(x) dx=\\
&\int_{0}^{1} x dx    = \frac12 x^2 |_0^1 = \frac12
\end{aligned}
$$
```

1.3 R Basics I

To start run

```
ls()
```

This shows you a "listing"" of the files (data, routines etc.) in the current project. (Likely there is nothing there right now)

Everything in R is either a data set or a function. It is a function if it is supposed to do something (maybe calculate something, show you something like a graph or something else etc.). If it is a function is ALWAYS NEEDS (). Sometimes the is something in between the parentheses, like in

```
mean(x)
```

```
## [1] 6
```

Sometimes there isn't like in the ls(). But the () has to be there anyway.

If you have worked for a while you might have things you need to save, do that by clicking on

File > Save

RStudio has a nice recall feature, using the up and down arrow keys. Also, clicking on the History tab shows you the recently run commands. Finally, typing the first three letters of a command in the console and then typing CTRL-ˆ shows you a list of when you ran commands like this the last times.

R is case-sensitive, so a and A are two different things.

Often during a session you create objects that you need only for a short time. When you no longer need them use **rm** to get rid of them:

```
x <- 10
x^2
```

```
## [1] 100
```

```
rm(x)
```

the **<-** is the *assignment* character in R, it assigns what is on the right to the symbol on the left. (Think of an arrow to the left)

1.3.1 Data Entry

For a few numbers the easiest thing is to just type them in:

```
x <-  c(10, 2, 6, 9)
x
```

```
## [1] 10  2  6  9
```

c() is a function that takes the objects inside the () and **c**ombines them into one single object (a vector).

1.3.2 Data Types in R

the most basic type of data in R is a **vector**, simply a list of values.

Say we want the numbers 1.5, 3.6, 5.1 and 4.0 in an R vector called x, then we can type

```
x <- c(1.5, 3.6, 5.1, 4.0)
x
```

```
## [1] 1.5 3.6 5.1 4.0
```

Often the numbers have a structure one can make use of:

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
10:1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
1:20*2
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

```
c(1:10, 1:10*2)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  2  4  6  8 10 12 14 16 18 20
```

Sometimes you need parentheses:

```
n <- 10
1:n-1
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
1:(n-1)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

The *rep* ("repeat") command is very useful:

```
rep(1, 10)
```

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

```
rep(1:3, 10)
```

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

```
rep(1:3, each=3)
```

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

```
rep(c("A", "B", "C"), c(4,7,3))
```

```
##  [1] "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B" "C" "C" "C"
```

what does this do?

```
rep(1:10, 1:10)
```

1.3.3 Commands for Vectors

To find out how many elements a vector has use the *length* command:

```
x <- c(1.4, 5.1, 2.0, 6.8, 3.5, 2.1, 5.6, 3.3, 6.9, 1.1)
length(x)
```

```
## [1] 10
```

The elements of a vector are accessed with the bracket [ ] notation:

```
x[3]
```

```
## [1] 2
```

```
x[1:3]
```

```
## [1] 1.4 5.1 2.0
```

```
x[c(1, 3, 8)]
```

```
## [1] 1.4 2.0 3.3
```

```
x[-3]
```

```
## [1] 1.4 5.1 6.8 3.5 2.1 5.6 3.3 6.9 1.1
```

```
x[-c(1, 2, 5)]
```

```
## [1] 2.0 6.8 2.1 5.6 3.3 6.9 1.1
```

Instead of numbers a vector can also consist of characters (letters, numbers, symbols etc.) These are identified by quotes:

```
c("A", "B", 7, "%")
```

```
## [1] "A" "B" "7" "%"
```

A vector is either numeric or character, but never both (see how the 7 was changed to “7”).

You can turn one into the other (if possible) as follows:

```
x <- 1:10
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
as.character(x)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

```
x <- c("1", "5", "10", "-3")
x
```

```
## [1] "1"  "5"  "10" "-3"
```

```
as.numeric(x)
```

```
## [1]  1  5 10 -3
```

A third type of data is logical, with values either TRUE or FALSE.

```
x <- 1:10
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
x > 4
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

these are often used as conditions:

```r
x[x>4]
```

```
## [1]  5  6  7  8  9 10
```

This, as we will see shortly, is EXTREMELY useful!

1.3.4 Data Frames

data frames are the basic format for data in R. They are essentially vectors of equal length put together as columns.

A data frame can be created as follows:

```r
df <- data.frame(
  Gender=c("M", "M", "F", "F", "F"),
  Age=c(23, 25, 19, 22, 21),
  GPA=c(3.5, 3.7, 2.9, 2.8, 3.1)
)
df
```

```
##   Gender Age GPA
## 1      M  23 3.5
## 2      M  25 3.7
## 3      F  19 2.9
## 4      F  22 2.8
## 5      F  21 3.1
```

1.3.5 Lists

The most general data structures are lists. They are simply a collection of objects. There are no restrictions on what those objects are.

1.3.5.1 Example

```r
lst <- list(
  Gender=c("M", "M", "F", "F", "F"),
  Age=c(23, 25, 19, 22, 21, 26, 34),
  f=function(x) x^2,
  list(A=c(1, 1), B=c("X", "X", "Y"))
)
lst
```

```
## $Gender
## [1] "M" "M" "F" "F" "F"
##
## $Age
## [1] 23 25 19 22 21 26 34
##
## $f
## function (x)
## x^2
## <environment: 0x0000020c48e9b7d8>
##
## [[4]]
## [[4]]$A
## [1] 1 1
##
## [[4]]$B
## [1] "X" "X" "Y"
```

A data frame is a list with an additional requirement, namely that the elements of the list be of equal length.

1.3.5.2 Case Study: UPR Admissions

consider the **upr** data set . This is the application data for all the students who applied and were accepted to UPR-Mayaguez between 2003 and 2013.

```r
dim(upr)
```

```
## [1] 23666    16
```

tells us that there were 23666 applications and that for each student there are 16 pieces of information.

```r
colnames(upr)
```

```
##  [1] "ID.Code"         "Year"            "Gender"          "Program.Code"
##  [5] "Highschool.GPA" "Aptitud.Verbal" "Aptitud.Matem"  "Aprov.Ingles"
##  [9] "Aprov.Matem"    "Aprov.Espanol"  "IGS"            "Freshmen.GPA"
## [13] "Graduated"      "Year.Grad."     "Grad..GPA"      "Class.Facultad"
```

shows us the variables

```r
head(upr, 3)
```

```
##      ID.Code Year Gender Program.Code Highschool.GPA Aptitud.Verbal
## 1 00C2B4EF77 2005      M          502           3.97            647
## 2 00D66CF1BF 2003      M          502           3.80            597
## 3 00AB6118EB 2004      M         1203           4.00            567
##   Aptitud.Matem Aprov.Ingles Aprov.Matem Aprov.Espanol IGS Freshmen.GPA
## 1           621          626         672           551 342         3.67
## 2          726          618         718           575 343           2.75
## 3          691          424         616           609 342           3.62
##   Graduated Year.Grad. Grad..GPA Class.Facultad
## 1        Si       2012      3.33           INGE
## 2        No         NA        NA           INGE
## 3        No         NA        NA       CIENCIAS
```

shows us the first three cases.

Let's say we want to find the number of males and females. We can use the table command for that:

```
table(Gender)
```

```
## Error: object 'Gender' not found
```

What happened? Right now R does not know what Gender is because it is "hidden" inside the upr data set. Think of **upr** as a box that is currently closed, so R can't look inside and see the column names. We need to open the box first:

```
attach(upr)
table(Gender)
```

```
## Gender
##     F     M
## 11487 12179
```

there is also a detach command to undo an attach, but this is not usually needed because the attach goes away when you close R.

Note: you need to attach a data frame only once in each session working with R.

Note: Say you are working first with a data set "students 2016" which has a column called Gender, and you attached it. Later (but in the same R session) you start working with a data set "students 2017" which also has a column called Gender, and you are attaching this one as well. If you use Gender now it will be from "students 2017".

1.3.6 Subsetting of Data Frames

Consider the following data frame (not a real data set):

```
students
```

```
##    Age GPA Gender
## 1   22 3.1   Male
## 2   23 3.2   Male
## 3   20 2.1   Male
## 4   22 2.1   Male
## 5   21 2.3 Female
## 6   21 2.9   Male
## 7   18 2.3 Female
## 8   22 3.9   Male
## 9   21 2.6 Female
## 10  18 3.2 Female
```

Here each single piece of data is identified by its row number and its column number. So for example in row 2, column 2 we have "3.2", in row 6, column 3 we have "Male".

As with the vectors before we can use the [ ] notation to access pieces of a data frame, but now we need to give it both the row and the column number, separated by a ,:

```
students[6, 3]
```

```
## [1] "Male"
```

As before we can pick more than one piece:

```
students[1:5, 3]
```

```
## [1] "Male"   "Male"   "Male"   "Male"   "Female"
```

```
students[1:5, 1:2]
```

```
##   Age GPA
## 1  22 3.1
## 2  23 3.2
## 3  20 2.1
## 4  22 2.1
## 5  21 2.3
```

```
students[-c(1:5), 3]
```

```
## [1] "Male"   "Female" "Male"   "Female" "Female"
```

```
students[1, ]
```

```
##   Age GPA Gender
## 1  22 3.1   Male
```

```
students[, 2]
```

```
##  [1] 3.1 3.2 2.1 2.1 2.3 2.9 2.3 3.9 2.6 3.2
```

```
students[, -3]
```

```
##    Age GPA
## 1   22 3.1
## 2   23 3.2
## 3   20 2.1
## 4   22 2.1
## 5   21 2.3
## 6   21 2.9
## 7   18 2.3
## 8   22 3.9
## 9   21 2.6
## 10  18 3.2
```

another way of subsetting a data frame is by using the $ notations:

```
students$Gender
```

```
##  [1] "Male"   "Male"   "Male"   "Male"   "Female" "Male"   "Female"
##  [8] "Male"   "Female" "Female"
```

1.3.7 Subsetting of Lists

The double bracket and the $ notation also work for lists:

1.3.7.1 Example

```
lst <- list(
  Gender=c("M", "M", "F", "F", "F"),
  Age=c(23, 25, 19, 22, 21, 26, 34),
  f=function(x) x^2,
  list(A=c(1, 1), B=c("X", "X", "Y"))
)
lst[[4]][[2]]
```

```
## [1] "X" "X" "Y"
```

```
lst$Gender
```

```
## [1] "M" "M" "F" "F" "F"
```

1.3.8 Vector Arithmetic

R allows us to apply any mathematical functions to a whole vector:

```
x <- 1:10
2*x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x^2
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
log(x)
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

```
sum(x)
```

```
## [1] 55
```

```
y <- 21:30
```

```r
x+y
```

```
##  [1] 22 24 26 28 30 32 34 36 38 40
```

```r
x^2+y^2
```

```
##  [1]  442  488  538  592  650  712  778  848  922 1000
```

```r
mean(x+y)
```

```
## [1] 31
```

Let's try something strange:

```r
c(1, 2, 3) + c(1, 2, 3, 4)
```

```
## [1] 2 4 6 5
```

so R notices that we are trying to add a vector of length 3 to a vector of length 4. This should not work, but it actually does!

When it runs out of values in the first vector, R simply starts all over again.

In general this is more likely a mistake by you, check that this is what you really wanted to do!

1.3.9 *apply*

A very useful routine in R is *apply*, and its brothers.

Let's say we have the following matrix:

```r
Age <- matrix(sample(20:30, size=100, replace=TRUE), 10, 10)
Age[1:5, 1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   20   27   27   20   29
## [2,]   25   23   26   25   27
## [3,]   25   26   20   21   24
## [4,]   25   30   23   22   20
## [5,]   23   26   30   30   26
```

and we want to find the sums of the ages in each column. Easy:

```r
sum(Age[, 1])
```

```
## [1] 249
```

```r
sum(Age[, 2])
```

```
## [1] 263
```

...

```r
sum(Age[, 10])
```

```
## [1] 269
```

or much easier

```r
apply(Age, 2, sum)
```

```
##  [1] 249 263 252 226 251 248 271 252 271 269
```

There are a number of apply routines for different data formats.

1.3.9.1 Case Study: upr admissions

Let's say we want to find the mean Highschool GPA:

```r
mean(Highschool.GPA)
```

```
## [1] 3.65861
```

But what if we want to do this for each year separately? Notice that *apply* doesn't work here because the Years are not in separated columns. Instead we can use

```r
tapply(Highschool.GPA, Year, mean)
```

```
##     2003     2004     2005     2006     2007     2008     2009     2010
## 3.646627 3.642484 3.652774 3.654729 3.628072 3.648552 3.642946 3.665298
##     2011     2012     2013
## 3.685485 3.695046 3.710843
```

1.4 R Basics II - Writing Functions

1.4.1 General Information

In R/RStudio you have several ways to write your own functions:

- In the R console type

```r
myfun <- function(x) {
  out <- x^2
  out
}
```

- RStudio: click on File > New File > R Script. A new empty window pops up. Type fun, hit enter, and the following text appears:

name <- function(variables) {

}

change the name to *myfun*, save the file as myfun.R with File > Save. Now type in the code. When done click the Source button.

- fix: In the R console run

```
fix(myfun)
```

now a window with an editor pops up and you can type in the code. When you are done click on Save. If there is some syntax error DON'T run fix again, instead run

```
myfun <- edit()
```

myfun will exist only until you close R/RStudio unless you save the project file.

- Open any code editor outside of RStudio, type in the code, save it as myfun.R, go to the console and run

```
source('../some.folder/myfun.R')
```

Which of these is best? In large part that depends on your preferences. In my case, if I expect to need that function just for a bit I use the fix option. If I expect to need that function again later I start with the first method, but likely soon open the .R file outside RStudio because most code editors have many useful features not available in RStudio.

If *myfun* is open in RStudio there are some useful keyboard shortcuts. If the curser is on some line in the RStudio editor you can hit

- CTRL-Enter run current line or section

- CTRL-ALT-B run from beginning to line

- CTRL-Shift-Enter run complete chunk
- CTRL-Shift-P rerun previous

1.4.2 Testing

As always you can test whether an object is a function:

```
x <- 1
f <- function(x) x
is.function(x)
```

```
## [1] FALSE
```

```
is.function(f)
```

```
## [1] TRUE
```

1.4.3 Arguments

There are several ways to specify arguments in a function:

```
calc.power <- function(x, y, n=2) x^n + y^n
```

here n has a *default value*, x and y do not.

if the arguments are not named they are matched in order:

```
calc.power(2, 3)
```

```
## [1] 13
```

If an argument does not have a default it can be tested for

```
f <- function(first, second) {
  if(!missing(second))
      out <- first + second
  else out <- first
  out
}
f(1)
```

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

```
f(1, s=3)
```

```
## [1] 4
```

There is a special argument ..., used to pass arguments on to other functions:

```
f <- function(x, which, ...) {
  f1 <- function(x, mult) mult*x
  f2 <- function(x, pow) x^pow
  if(which==1)
    out <- f1(x, ...)
  else
    out <- f2(x, ...)
  out
}
f(1:3, 1, mult=2)
```

```
## [1] 2 4 6
```

```
f(1:3, 2, pow=3)
```

```
## [1]  1  8 27
```

This is one of the most useful programming structures in R!

Note this example also shows that in R functions can call other functions. In many computer programs there are so called *sub-routines*, in R this concept does not exist, functions are just functions.

1.4.4 Return Values

A function can either return nothing or exactly one thing. It will automatically return the last object evaluated:

```r
f <- function(x) {
  x^2
}
f(1:3)
```

```
## [1] 1 4 9
```

however, it is better programming style to have an explicit return object:

```r
f <- function(x) {
  out <- x^2
  out
}
f(1:3)
```

```
## [1] 1 4 9
```

There is another way to specify what is returned:

```r
f <- function(x) {
  return(x^2)
}
f(1:3)
```

```
## [1] 1 4 9
```

but this is usually used to return something early in the program:

```r
f <- function(x) {
  if(!any(is.numeric(x)))
    return("Works only for numeric!")
  out <- sum(x^2)
  out
}
f(1:3)
```

```
## [1] 14
```

```r
f(letters[1:3])
```

```
## [1] "Works only for numeric!"
```

If you want to return more than one item use a list:

```r
f <- function(x) {
  sq <- x^2
  sm <- sum(x)
  list(sq=sq, sum=sm)
}
f(1:3)
```

```
## $sq
## [1] 1 4 9
```

```
##
## $sum
## [1] 6
```

1.4.5 Basic Programmming Structures in R

R has all the standard programming structures:

1.4.5.1 Conditionals (if-else)

```
f <- function(x) {
  if(x>0) y <- log(x)
  else y <- NA
  y
}
f(c(2, -2))
```

```
## [1] 0.6931472       NaN
```

A useful variation on the *if* statement is *switch*:

```
centre <- function(x, type) {
  switch(type,
         mean = mean(x),
         median = median(x),
         trimmed = mean(x, trim = .1))
}
x <- rcauchy(10)
centre(x, "mean")
```

```
## [1] -2.13241
```

```
centre(x, "median")
```

```
## [1] -0.2208695
```

```
centre(x, "trimmed")
```

```
## [1] -0.8328785
```

special R construct: *ifelse*

```
x <- sample(1:10, size=7, replace = TRUE)
x
```

```
## [1] 9 9 7 8 3 1 6
```

```
ifelse(x<5, "Yes", "No")
```

```
## [1] "No"  "No"  "No"  "No"  "Yes" "Yes" "No"
```

1.4.5.2 Loops

there are two standard loops in R:

- for loop

```
y <- rep(0, 10)
for(i in 1:10) y[i] <- i*(i+1)/2
y
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

sometimes we don't know the length of y ahead of time, then we can use

```
for(i in seq_along(y)) y[i] <- i*(i+1)/2
y
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

If there is more than one statement inside a loop use curly braces:

```
for(i in seq_along(y)) {
  y[i] <- i*(i+1)/2
  if(y[i]>40) y[i] <- (-1)
}
y
```

```
##  [1]  1  3  6 10 15 21 28 36 -1 -1
```

You can nest loops:

```
A <- matrix(0, 4, 4)
for(i in 1:4) {
  for(j in 1:4)
    A[i, j] <- i*j
}
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2    4    6    8
## [3,]    3    6    9   12
## [4,]    4    8   12   16
```

- repeat loop

```
k <- 0
repeat {
  k <- k+1
  x <- sample(1:6, size=3, replace=TRUE)
  if(length(table(x))==1) break
}
k
```

```
## [1] 35
```

Notice that a repeat loop could in principle run forever. I usually include a counter that ensures the loop will eventually stop:

```
k <- 0
counter <- 0
repeat {
  k <- k+1
  counter <- counter+1
  x <- sample(1:6, size=3, replace=TRUE)
  if(length(table(x))==1 | counter>1000) break
}
k
```

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

1.5 Random Numbers and Simulation

1.5.1 Random Numbers

Everything starts with generating X_1, X_2, .. iid U[0,1]. These are simply called random numbers. There are some ways to get these:

- random number tables
- numbers taken from things like the exact (computer) time
- quantum random number generators
- ...

The R package *random* has the routine *randomNumbers* which gets random numbers from a web site which generates them based on (truly random) atmospheric phenomena.

```
require(random)
randomNumbers(20, 0, 100)
```

```
##      V1 V2 V3 V4 V5
## [1,]  4 67 17 30 82
## [2,] 50 75 82 21 24
## [3,] 75 46 19 68 85
## [4,] 52 49 93 42 52
```

Most of the time we will use *pseudo-random numbers*, that is numbers that are not actually random but are indistinguishable from those. In R this is done with

```
runif(5)
```

```
## [1] 0.239821392 0.007167395 0.793589296 0.004363637 0.543807629
```

```r
runif(5, 100, 300)
```

```
## [1] 285.6445 149.1793 111.0248 288.6031 224.0891
```

1.5.2 Standard Probability Distributions

Not surprisingly many standard distributions are part of base R. For each the format is

- dname = density
- pname = cumulative distribution function
- rname = random generation
- qname = quantile function

Note we will use the term *density* for both discrete and continuous random variable.

1.5.2.1 Example Poisson distribution

We have $X \sim \text{Pois}(\lambda)$ if

$$P(X = x) = \frac{\lambda^x}{x!}e^{-\lambda}, \, x = 0, 1, \ldots$$

```r
# density
dpois(c(0, 8, 12, 20), lambda=10)
```

```
## [1] 4.539993e-05 1.125990e-01 9.478033e-02 1.866081e-03
```

```r
10^c(0, 8, 12, 20)/factorial(c(0, 8, 12, 20))*exp(-10)
```

```
## [1] 4.539993e-05 1.125990e-01 9.478033e-02 1.866081e-03
```

```r
# cumulative distribution function
ppois(c(0, 8, 12, 20), 10)
```

```
## [1] 4.539993e-05 3.328197e-01 7.915565e-01 9.984117e-01
```

```r
# random generation
rpois(5, 10)
```

```
## [1] 11 15  9 14 11
```

```r
# quantiles
qpois(1:4/5, 10)
```

```
## [1]  7  9 11 13
```

Here is a list of the distributions included with base R:

- beta distribution: dbeta.

- binomial (including Bernoulli) distribution: dbinom.
- Cauchy distribution: dcauchy.
- chi-squared distribution: dchisq.
- exponential distribution: dexp.
- F distribution: df.
- gamma distribution: dgamma.
- geometric distribution: dgeom.
- hypergeometric distribution: dhyper.
- log-normal distribution: dlnorm.
- multinomial distribution: dmultinom.
- negative binomial distribution: dnbinom.
- normal distribution: dnorm.
- Poisson distribution: dpois.
- Student's t distribution: dt.
- uniform distribution: dunif.
- Weibull distribution: dweibull.

With some of these a bit of caution is needed. For example, the usual textbook definition of the geometric random variable is the number of tries in a sequence of independent Bernoulli trials until a success. This means that the density is defined as

$$P(X=x)=p(1-p)^{x-1},\ x=1,2,..$$

R however defines it as the number of failures until the first success, and so it uses

$$P(X^*=x)=\text{dgeom}(x,p)=p(1-p)^x,\ x=0,1,2,..$$

Of course this is easy to fix. If you want to generate the "usual" geometric do

```
x <- rgeom(10, 0.4) + 1
x
```

```
##  [1] 2 1 2 4 3 3 1 1 3 4
```

if you want to find the probabilities or cdf:

```
round(dgeom(x-1, 0.4), 4)
```

```
##  [1] 0.2400 0.4000 0.2400 0.0864 0.1440 0.1440 0.4000 0.4000 0.1440 0.0864
```

```
round(0.4*(1-0.4)^(x-1), 4)
```

```
##  [1] 0.2400 0.4000 0.2400 0.0864 0.1440 0.1440 0.4000 0.4000 0.1440 0.0864
```

Another example is the Gamma random variable. Here most textbooks use the definition

$$f(x;\alpha,\beta) = \frac{1}{\Gamma(\alpha)\beta^\alpha}x^{\alpha-1}e^{-x/\beta}$$

but R uses

$$f^*(x;\alpha,\beta) = \frac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x}$$

```r
dgamma(1.2, 0.5, 2)
```

```
## [1] 0.06607584
```

```r
2^0.5/gamma(0.5)*1.2^(0.5-1)*exp(-2*1.2)
```

```
## [1] 0.06607584
```

Again, it is easy to *re-parametrize*:

```r
dgamma(1.2, 0.5, 1/(1/2))
```

```
## [1] 0.06607584
```

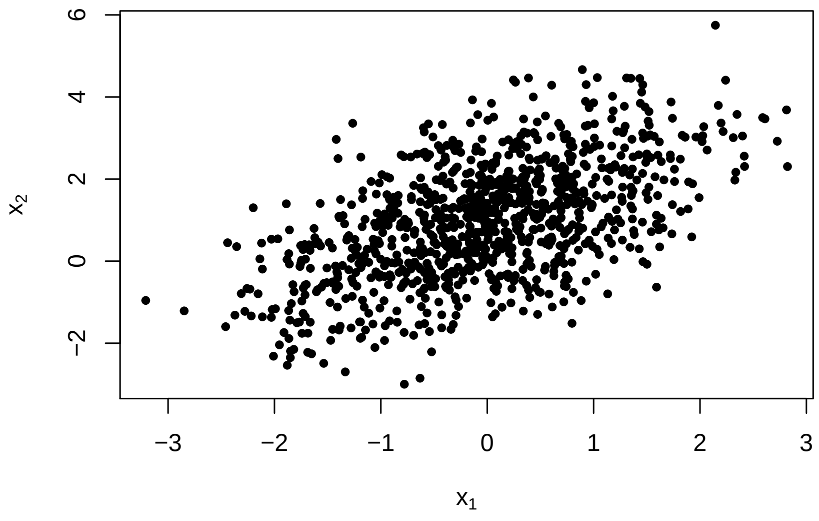
1.5.3 Other Variates

if you need to generate random variates from a distribution that is not part of base R you should first try to find a package that includes it.

1.5.3.1 Example multivariate normal

there are actually several packages, the most commonly used one is *mvtnorm*

```r
library(mvtnorm)
x <- rmvnorm(1000,
             mean = c(0, 1),
             sigma = matrix(c(1, 0.8, 0.8, 2), 2, 2))
plot(x,
     pch=20,
     xlab = expression(x[1]),
     ylab = expression(x[2]))
```



sigma is the variance-covariance matrix, so in the above we have

$$\rho = Cor(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)\,Var(Y)}} = \frac{0.8}{\sqrt{1*2}} = 0.566$$

```
round(c(var(x[, 1]),
        var(x[, 2]),
        cor(x[, 1], x[, 2])), 3)
```

```
## [1] 0.978 1.968 0.555
```

1.5.4 Simulation

In a *simulation* we attempt to generate data just like what we might see in a real-live experiment, except that we control all the details. The we carry out some calculations on that artificial data, and we repeat this many times. Here are some examples:

1.5.4.1 Example

When rolling a fair die 5 times, what is the probability of no sixes? Of no more than one six?

```
B <- 10000 # number of simulation runs
num.sixes <- rep(0, B) # to store results
```

```r
for(i in 1:B) {
  x <- sample(1:6, size=5, replace=TRUE) # roll 5 dice
  num.sixes[i] <- length(x[x==6]) # how many sixes?
}
# Probability of no sixes
length(num.sixes[num.sixes==0])/B
```

```
## [1] 0.3948
```

```r
# Probability of no more than one sixes
length(num.sixes[num.sixes<=1])/B
```

```
## [1] 0.8078
```

Of course one can do this also analytically:

$$
\begin{aligned}
&P(\text{no sixes}) = P(\text{no six on any die}) = \\
&P(\text{no six on first die } \cap \text{ .. } \cap \text{ no six on fifth die}) = \\
&\prod_{i=1}^{5} P(\text{no six on } i^{th}\text{die}) = (\frac{5}{6})^5 = 0.402
\end{aligned}
$$

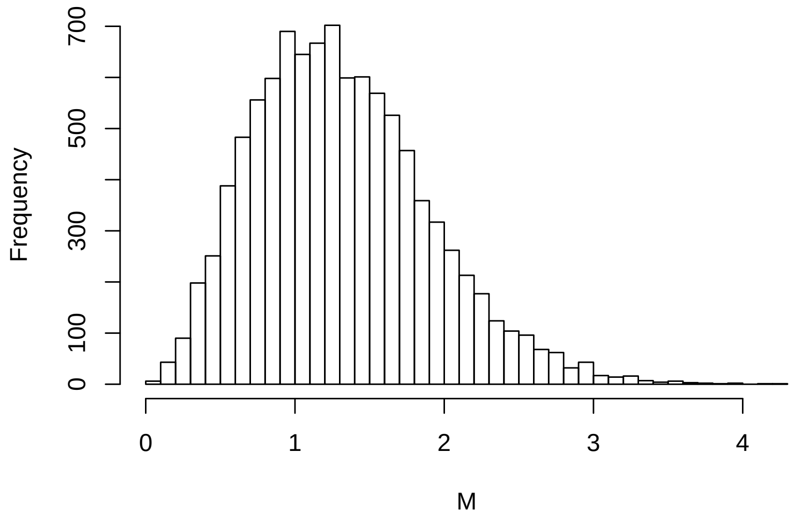
but already the second one is a bit harder to do analytically but not via simulation.

One issue we have with a simulation is the *simulation error*, namely that the simulation will always yield a slightly different answer.

1.5.4.2 Example

Say we have $X, Y, Z \sim N(0,1)$ and set $M = \max\{|X|, |Y|, |Z|\}$. What is the mean and standard deviation of M?

```r
B <- 10000
x <- matrix(abs(rnorm(3*B)), ncol=3)
M <- apply(x, 1, max)
hist(M, 50, main="")
```



```r
round(c(mean(M), sd(M)), 3)
```

```
## [1] 1.315 0.588
```

1.5.4.3 Example Symmetric Random Walk in R

Let $P(Z_i = -1) = P(Z_i = 1) = \frac{1}{2}$ and $X_n = \sum_{i=1}^{n} Z_i$. Let A>0 some integer. Let's write a routine that finds the median number of steps the walk takes until it hits either -A or A.

One issue with simulations of *stochastic processes* is that in general they are very slow. Here I will use a little trick: I will generate part of the process, and then check whether the event of interest has already happened.

```r
first.hit <- function(A) {
  B <- 10000
  num.steps <- rep(0, B)
  for(i in 1:B) {
    x <- 0
    k <- 0
    repeat {
      z <- sample(c(-1, 1), size=1000, replace=TRUE)
      x <- x + cumsum(z)
      if(max(abs(x))>=A) break
      x <- x[1000]
      k <- k+1000
    }
    k <- k+seq_along(x)[abs(x)>=A][1]
    num.steps[i] <- k
  }
  median(num.steps)
}
first.hit(100)
```

```
## [1] 7520
```

1.5.4.4 Example

The following you find in any basic stats course: A $100(1-\alpha)\%$ confidence interval for the success probability in a sequence of n Bernoulli trials is given by

$$\hat{p} \pm z_{\alpha/2}\sqrt{\hat{p}(1-\hat{p})/n}$$

where $\hat{p}$ is the proportion of successes. This method is supposed to work if n is at least 50.

Let's do a simulation to test this method.

```r
ci.prop.sim <- function(p, n, conf.level=95, B=1e4) {
  z <- qnorm(1-(1-conf.level/100)/2)
  bad <- 0
  for(i in 1:B) {
    x <- sample(0:1, size=n, replace = TRUE, prob=c(1-p, p))
    phat <- sum(x)/n
    if(phat - z*sqrt(phat*(1-phat)/n)>p) bad<-bad+1
    if(phat + z*sqrt(phat*(1-phat)/n)<p) bad<-bad+1
  }
  bad/B
}
```

```r
ci.prop.sim(0.5, 100)
```

```
## [1] 0.0582
```

and that is not so bad.

But

```r
ci.prop.sim(0.1, 50)
```

```
## [1] 0.1186
```

and that is very bad indeed!

Soon we will consider a method that is guaranteed to give intervals with correct coverage, no matter what p and n are.

1.5.4.5 Example: Simultaneous Inference

There is a famous (infamous?) case of three psychiatrists who studied a sample of schizophrenic persons and a sample of non schizophrenic persons. They measured 77 variables for each subject - religion, family background, childhood experiences etc. Their goal was to discover what distinguishes persons who later become schizophrenic. Using their data they ran 77 hypothesis tests of the significance of the differences between the two groups of subjects, and found 2 significant at the 2% level.They immediately published their findings.

What's wrong here? Remember, if you run a hypothesis test at the 2% level you expect to reject the null hypothesis of no relationship 2% of the time, but 2% of 77 is about 1 or 2, so just by random fluctuations they could (should?) have rejected that many null hypotheses! This is not to say that the variables they found to be different between the two groups were not really different, only that their method did not proof that.

In its general form this is known as the problem of simultaneous inference and is one of the most difficult issues in Statistics today. One general solution of used is called *Bonferroni's method*. The idea is the following:

Let's assume we carry out k hypothesis tests. All tests are done at α significance level and all the tests are all independent. Then the probability that at least one test rejects the null hypothesis although all null are true is given by

$$\begin{aligned}
&\alpha^* = P(\text{at least one null rejected} \mid \text{all null true}) = \\
&1 - P(\text{none of the nulls rejected} \mid \text{all null true}) = \\
&1 - \prod_{i=1}^{k} P(\text{ith null is not rejected} \mid \text{ith null true}) = \\
&1 - \prod_{i=1}^{k} [1 - P(\text{ith null is rejected} \mid \text{ith null true})] = \\
&1 - [1-\alpha]^k = \\
&1 - \left[1 - k\alpha + \binom{k}{2}\alpha^2 - +..\right] \approx k\alpha
\end{aligned}$$

so if each individual test is done with α/k, the *family-wise* error rate is the desired one.

Let's do a simulation to see how that would work in the case of our psychiatrists experiments. There many details we don't know, so we have to make them up a bit:

```
sim.shiz <- function(m, n=50, B=1e3) {
  counter <- matrix(0, B, 2)
  for(i in 1:B) {
    for(j in 1:77) {
      pval <- t.test(rnorm(n), rnorm(n))$p.value
      if(pval<0.02) counter[i, 1]<-1
      if(pval<0.05/m) counter[i, 2]<-1
    }
  }
  apply(counter, 2, sum)/B
}
sim.shiz(77)
```

```
## [1] 0.784 0.041
```

This works fine here. The main problem in real life is that it is rarely true that these test are independent, and then all we can say is that the needed α is between α/k and α.

1.6 Graphics with ggplot2

A large part of this chapter is taken from various works of Hadley Wickham. Among others The layered grammar of graphics and R for Data Science.

1.6.1 Why ggplot2?

Advantages of ggplot2

- consistent underlying grammar of graphics (Wilkinson, 2005)
- plot specification at a high level of abstraction
- very flexible
- theme system for polishing plot appearance
- mature and complete graphics system
- many users, active mailing list

but really, they are just so much nicer than base R graphs!

1.6.2 Grammar of Graphics

In 2005 Wilkinson, Anand, and Grossman published the book "The Grammar of Graphics". In it they laid out a systematic way to describe any graph in terms of basic building blocks. ggplot2 is an implementation of their ideas.

The use of the word *grammar* seems a bit strange here. The general dictionary meaning of the word grammar is:

the fundamental principles or rules of an art or science

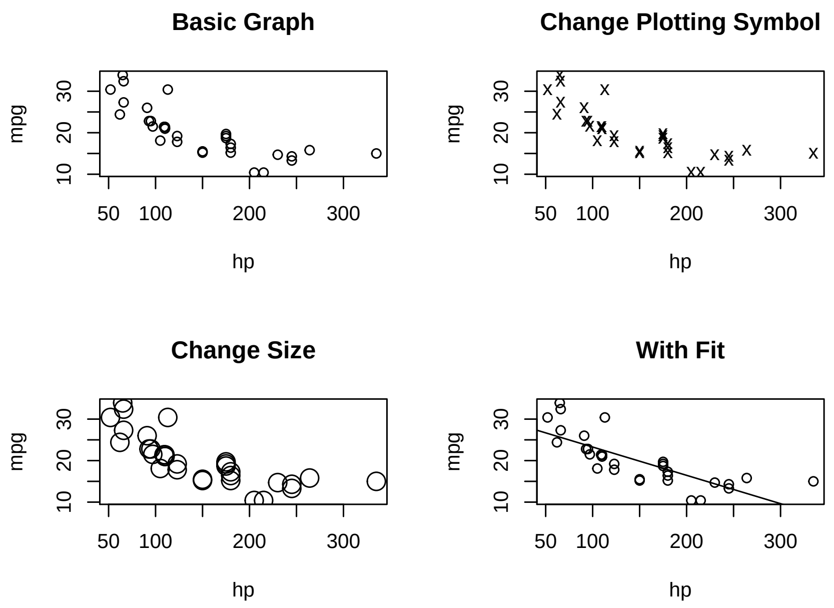
so it is not only about language.

As our running example we will use the *mtcars* data set. It is part of base R and has information on 32 cars:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Say we want to study the relationship of hp and mpg. So we have two quantitative variables, and therefore the obvious thing to do is a scatterplot. But there are a number of different ways we can do this:

```
attach(mtcars)
par(mfrow=c(2, 2))
plot(hp, mpg, main="Basic Graph")
plot(hp, mpg, pch="x", main="Change Plotting Symbol")
plot(hp, mpg, cex=2, main="Change Size")
plot(hp, mpg, main="With Fit");abline(lm(mpg~hp))
```

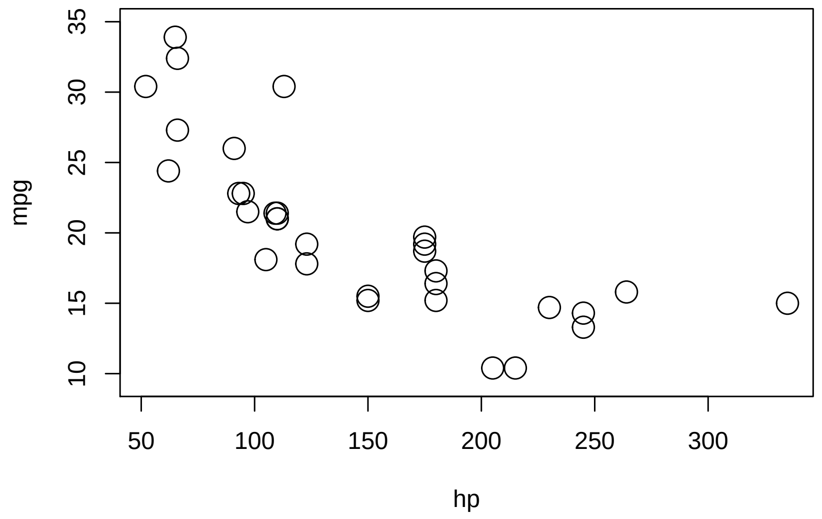


The basic idea of the grammar of graphs is to separate out the parts of the graphs: there is the basic layout, there is the data that goes into it, there is the way in which the data is displayed. Finally there are annotations, here the titles, and other things added, such as a fitted line. In ggplot2 you can always change one of these without worrying how that change effects any of the others.

Take the graph on the lower left. Here I made the plotting symbol bigger (with cex=2). But now the graph doesn't look nice any more, the first and the last circle don't fit into the graph.

The only way to fix this is to start all over again, by making the margins bigger:

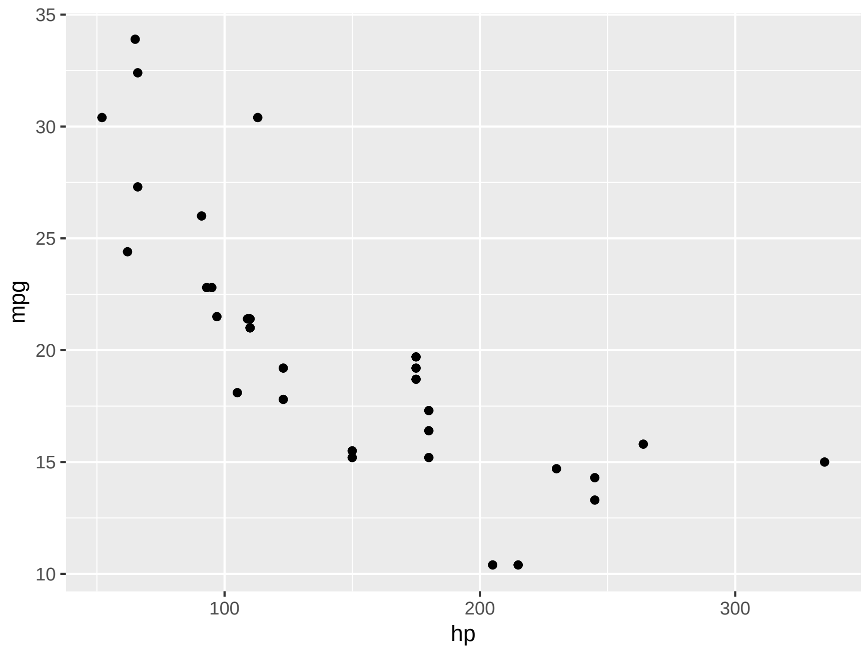
```r
plot(hp, mpg, cex=2, ylim=range(mpg)+c(-1, 1))
```



and that is a bit of work because I have to figure out how to change the margins. In ggplot2 that sort of thing is taken care of automatically!

Let's start by recreating the first graph above.

```r
ggplot(mtcars, aes(hp, mpg)) +
  geom_point()
```

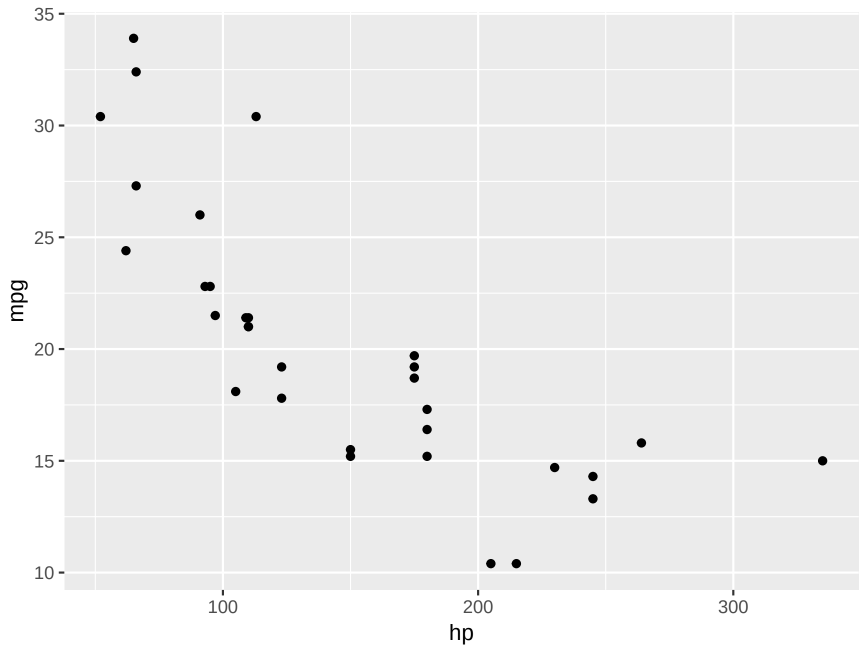


this has the following logic:

- *ggplot* sets up the graph
- it's first argument is the data set (which has to be a dataframe)
- *aes* is the *aestetic mapping*. It connects the data to the graph by specifying which variables go where
- *geom* is the geometric object (circle, square, line) to be used in the graph

Note ggplot2 also has the *qplot* command. This stands for *qick plot*

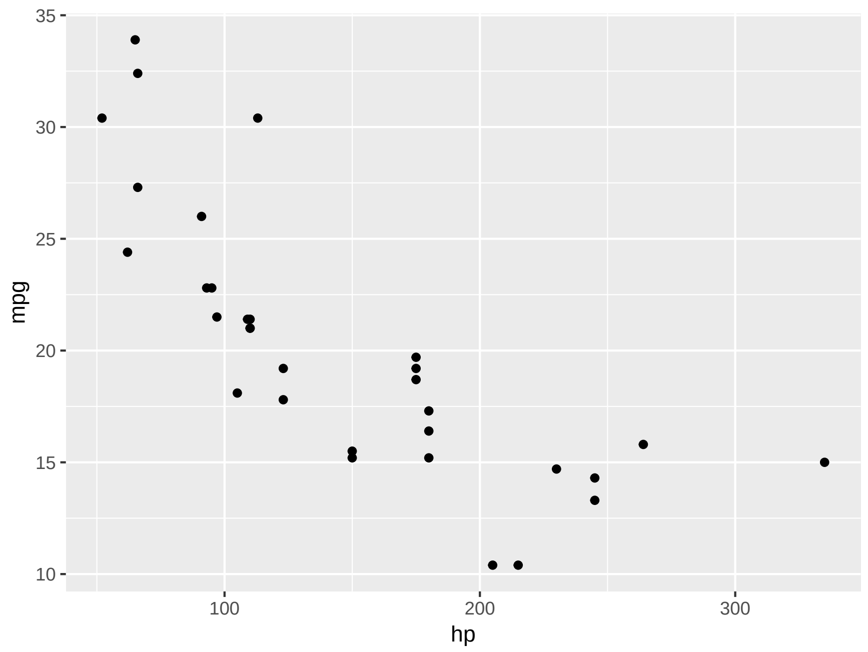
```
qplot(hp, mpg, data=mtcars)
```



This seems much easier at first (and it is) but the qplot command is also quite limited. Very quickly you want to do things that aren't possible with qplot, and so I won't discuss it further here.

Note consider the following variation:

```
ggplot(mtcars) +
  geom_point(aes(hp, mpg))
```



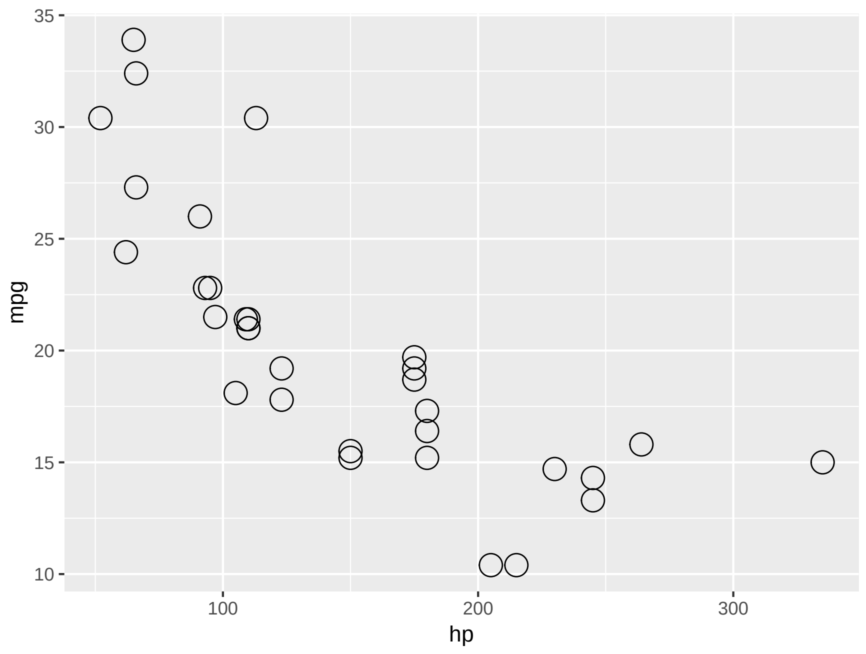
again it seems to do the same thing, but there is a big difference:

- if aes(x, y) is part of ggplot, it applies to all the geom's that come later (unless a different one is specified)

- an aes(x, y) as part of a geom applies only to it.

How about the problem with the graph above, where we had to increase the y margin?

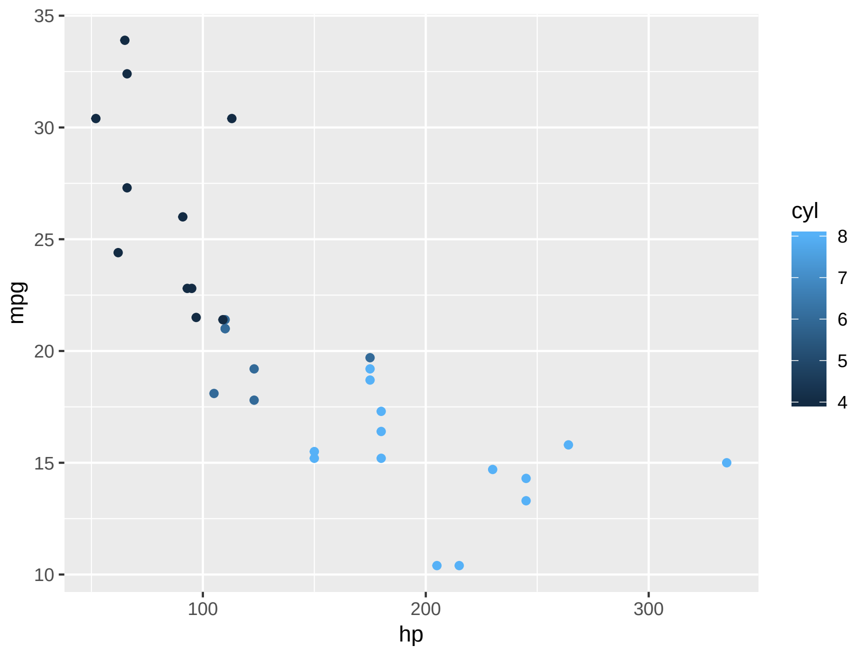
```r
ggplot(mtcars, aes(hp, mpg)) +
  geom_point(shape=1, size=5)
```



so we see that here this is done automatically.

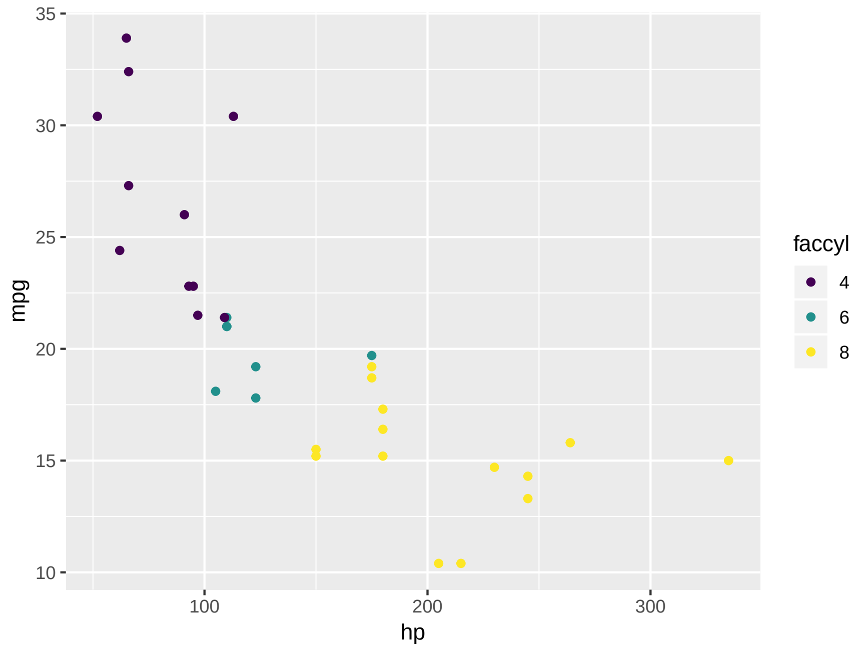
Let's say we want to identify the cars by the number of cylinders:

```r
ggplot(mtcars, aes(hp, mpg, color=cyl)) +
  geom_point()
```



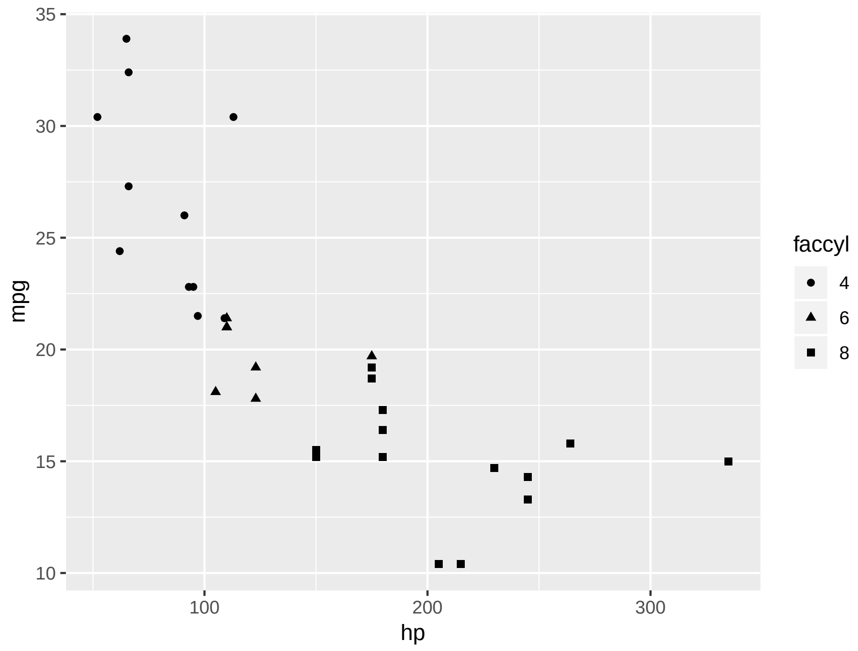
Notice that the legend is a continuous color scale. This is because the variable cyl has values 4, 6, and 8, and so is identified by R as a numeric variable. In reality it is categorical (ever seen a car with 1.7 cylinders?), and so we should change that:

```r
mtcars$faccyl <- factor(cyl,
                        levels = c(4, 6, 8),
                        ordered = TRUE)
ggplot(mtcars, aes(hp, mpg, color=faccyl)) +
  geom_point()
```



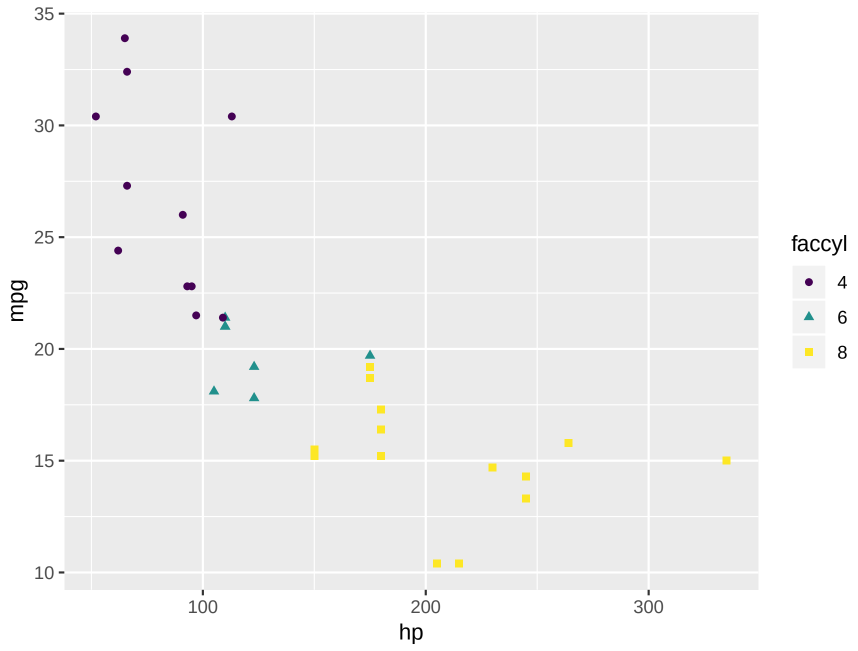
we can also change the shape of the plotting symbols:

```r
ggplot(mtcars, aes(hp, mpg, shape=faccyl)) +
  geom_point()
```



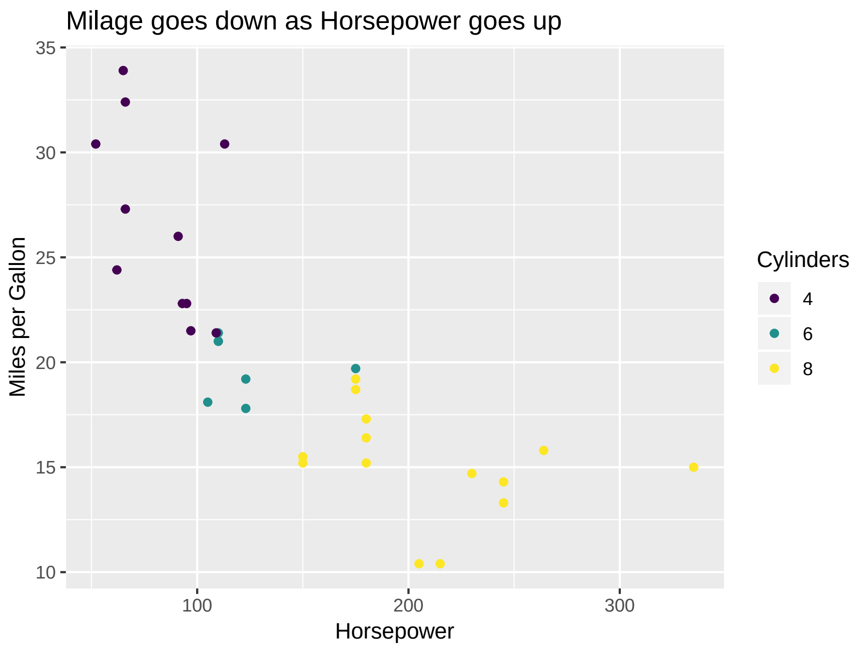
or both:

```r
ggplot(mtcars, aes(hp, mpg, shape=faccyl, color=faccyl)) +
  geom_point()
```



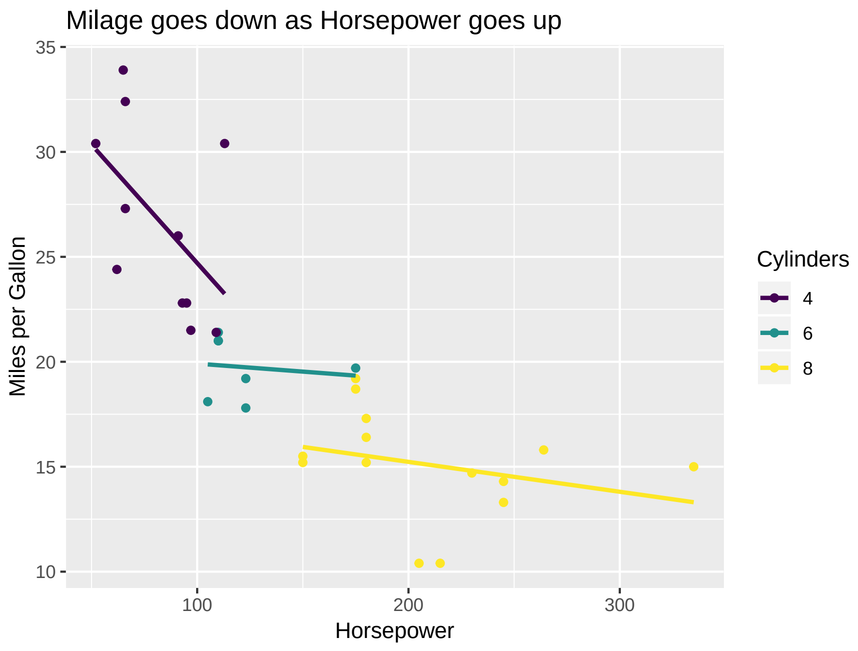
let's pretty up the graph a bit with some labels and a title. We will be playing around with this graph for a while, so I will save some intermediate versions:

```r
plt1 <- ggplot(mtcars, aes(hp, mpg, color=faccyl)) +
  geom_point()
plt2 <- plt1 +
  labs(x = "Horsepower",
       y = "Miles per Gallon",
       color = "Cylinders") +
  labs(title = "Milage goes down as Horsepower goes up")
plt2
```



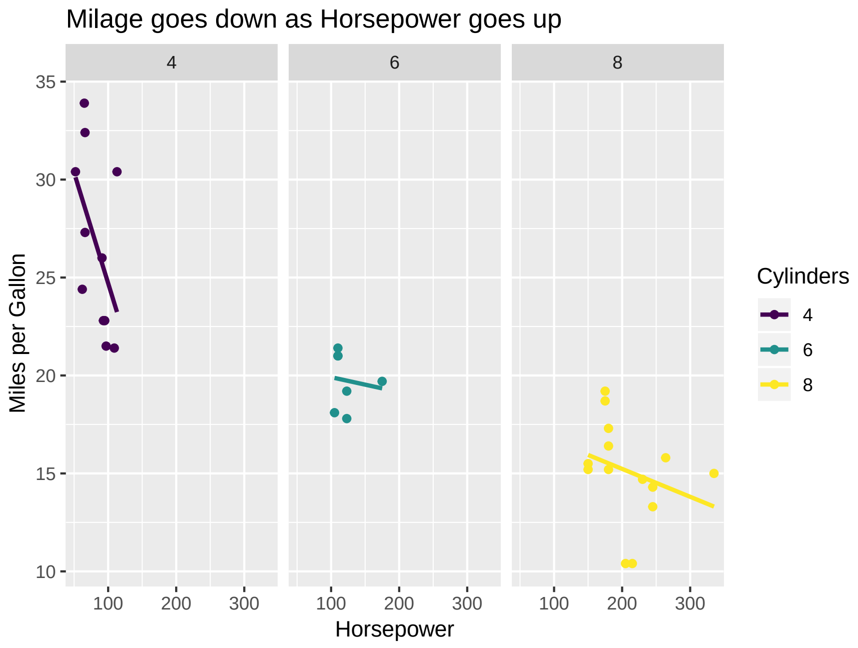
Say we want to add the least squares regression lines for cars with the same number of cylinders:

```
plt3 <- plt2 +
  geom_smooth(method = "lm", se = FALSE)
plt3
```



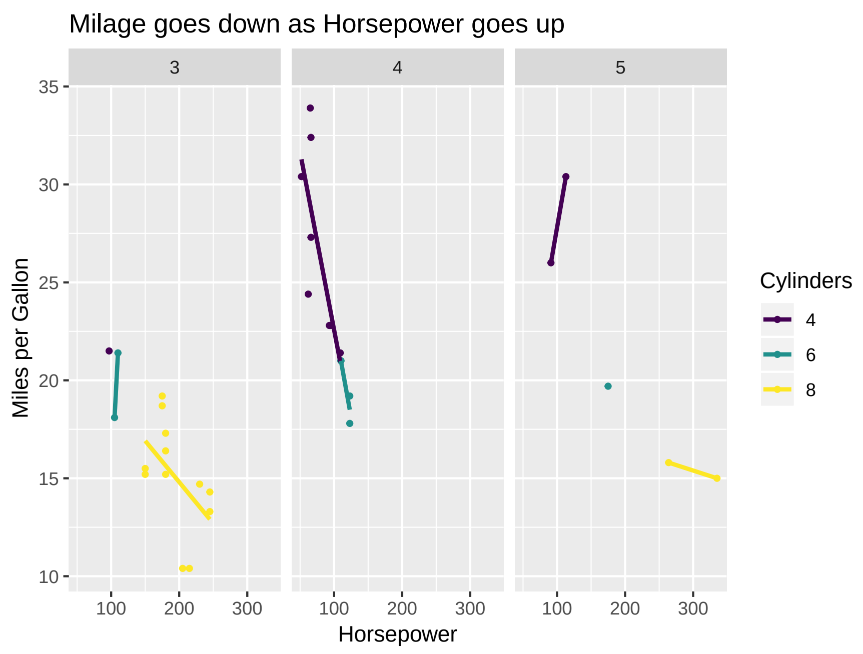
There is another way to include a categorical variable in a scatterplot. The idea is to do several graphs, one for each value of the categorical variable. These are called *facets*:

```r
plt3 +
  facet_wrap(~cyl)
```



The use of facets also allows us to include two categorical variables:

```r
mtcars$facgear <-
  factor(gear, levels = 3:5, ordered = TRUE)
plt4 <- ggplot(aes(hp, mpg, color=faccyl),
               data = mtcars) +
           geom_point(size = 1)
plt4 <- plt4 +
  facet_wrap(~facgear)
plt4 <- plt4 +
  labs(x = "Horsepower",
       y = "Miles per Gallon",
       color = "Cylinders") +
  labs(title = "Milage goes down as Horsepower goes up")
plt4 <- plt4 +
  geom_smooth(method = "lm", se = FALSE)
plt4
```

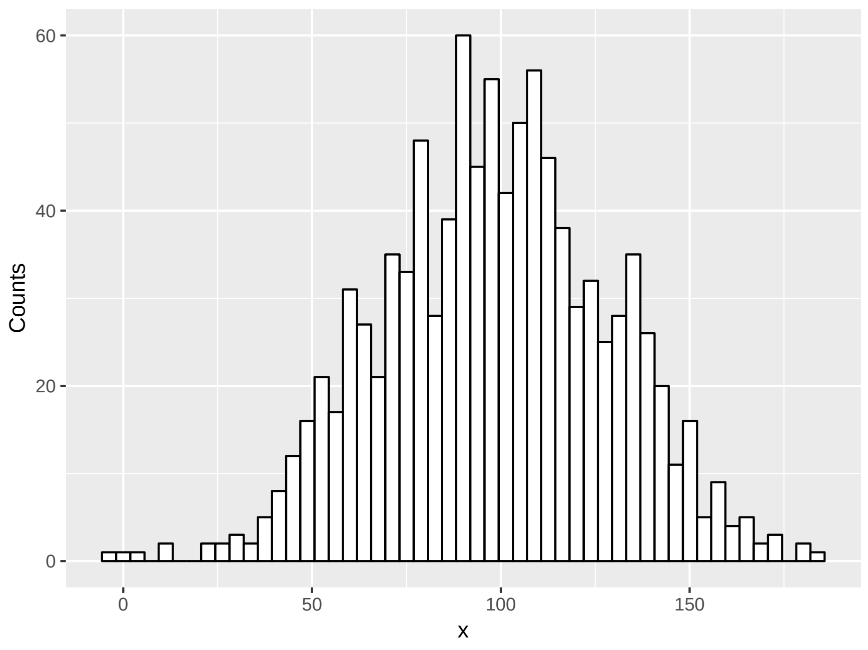


This is almost a bit to much, with just 32 data points there is not really enough for such a split.

Let's see how to use ggplot do a number of basic graphs:

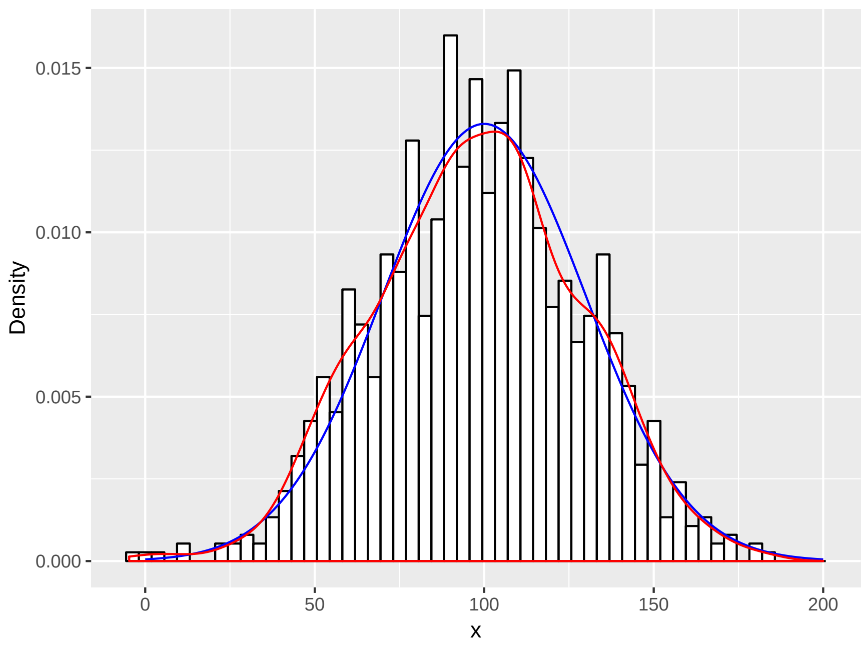
1.6.3 Histograms

```
x <- rnorm(1000, 100, 30)
df3 <- data.frame(x = x)
bw <- diff(range(x))/50 # use about 50 bins
ggplot(df3, aes(x)) +
  geom_histogram(color = "black",
                 fill = "white",
                 binwidth = bw) +
  labs(x = "x", y = "Counts")
```



Often we do histograms scaled to integrate to one. Then we can add the theoretical density and/or a nonparametric density estimate:

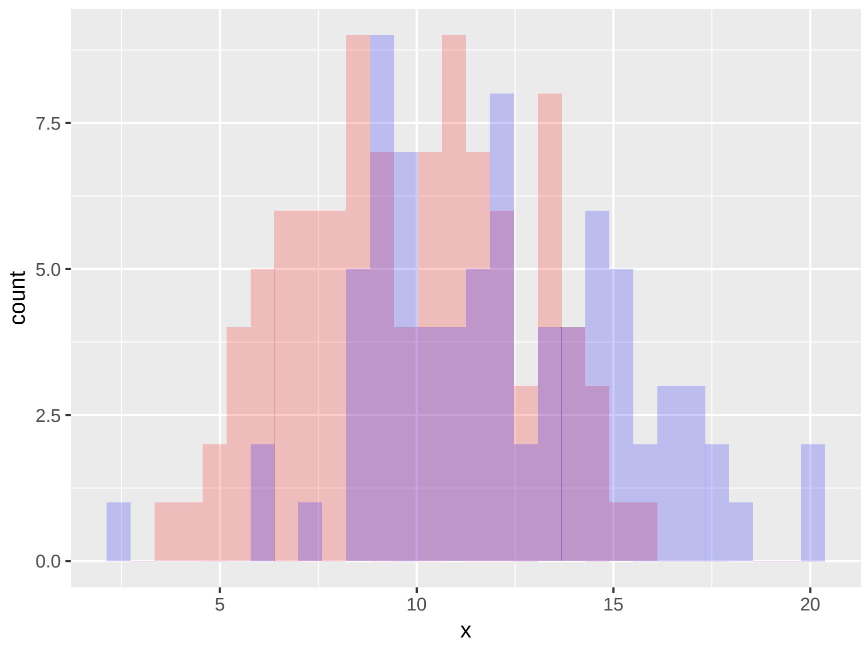
```r
x <- seq(0, 200, length=250)
df4 <- data.frame(x=x, y=dnorm(x, 100, 30))
ggplot(df3, aes(x)) +
  geom_histogram(aes(y = ..density..),
        color = "black",
        fill = "white",
        binwidth = bw) +
  labs(x = "x", y = "Density") +
  geom_line(data = df4, aes(x, y),
            colour = "blue") +
  geom_density(color = "red")
```



Notice the red line on the bottom. This should not be there but seems almost impossible to get rid of!

Here is another interesting case: say we have two data sets and we wish to draw the two histograms, one overlaid on the other:

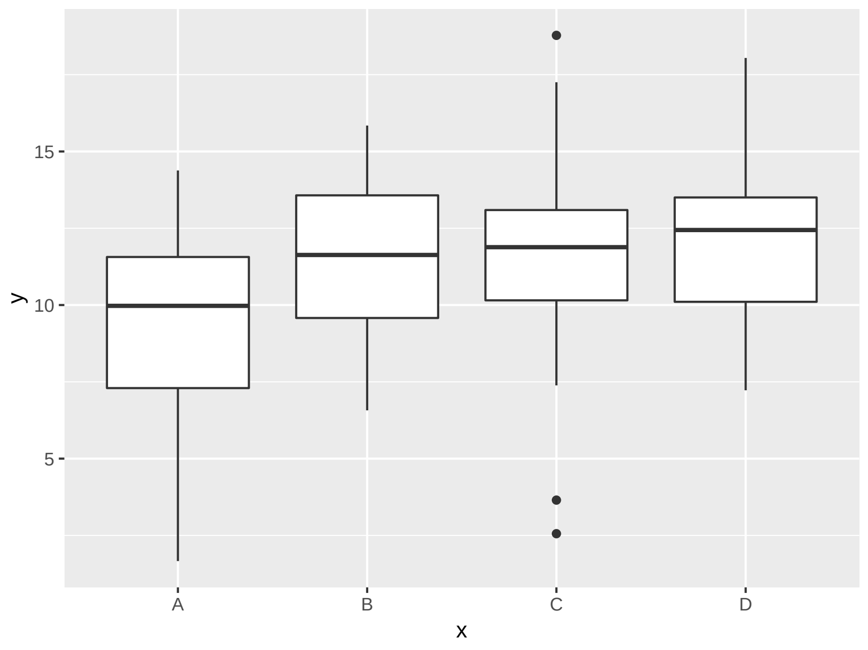
```
df5 <- data.frame(
  x = c(rnorm(100, 10, 3), rnorm(80, 12, 3)),
  y = c(rep(1, 100), rep(2, 80)))
ggplot(df5, aes(x=x)) +
    geom_histogram(data = subset(df5, y == 1),
        fill = "red", alpha = 0.2) +
    geom_histogram(data = subset(df5, y == 2),
        fill = "blue", alpha = 0.2)
```



Notice the use of alpha. In general this “lightens” the color so we can see “behind”.

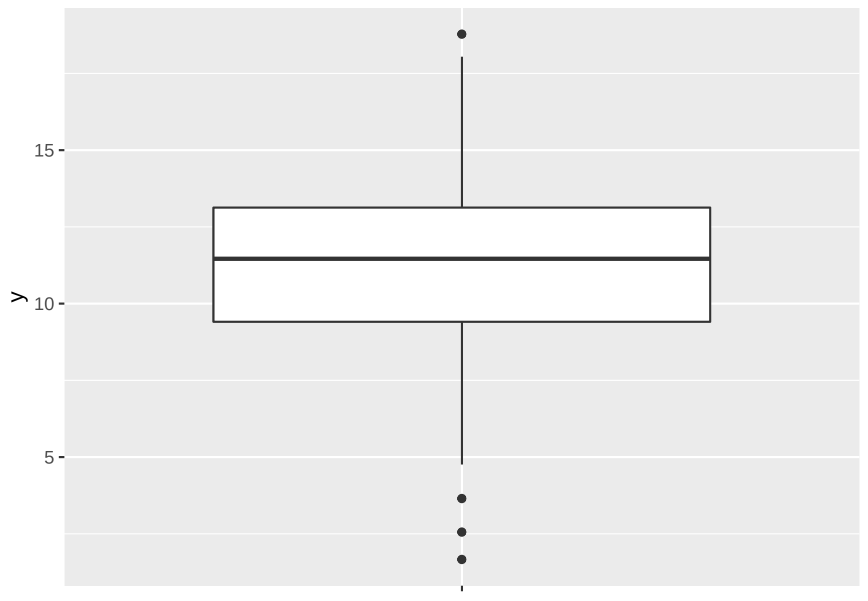
1.6.4 Boxplots

```
y <- rnorm(120, 10, 3)
x <- rep(LETTERS[1:4], each=30)
y[x=="B"] <- y[x=="B"] + rnorm(30, 1)
y[x=="C"] <- y[x=="C"] + rnorm(30, 2)
y[x=="D"] <- y[x=="D"] + rnorm(30, 3)
df6 <- data.frame(x=x, y=y)
ggplot(df6, aes(x, y)) +
  geom_boxplot()
```



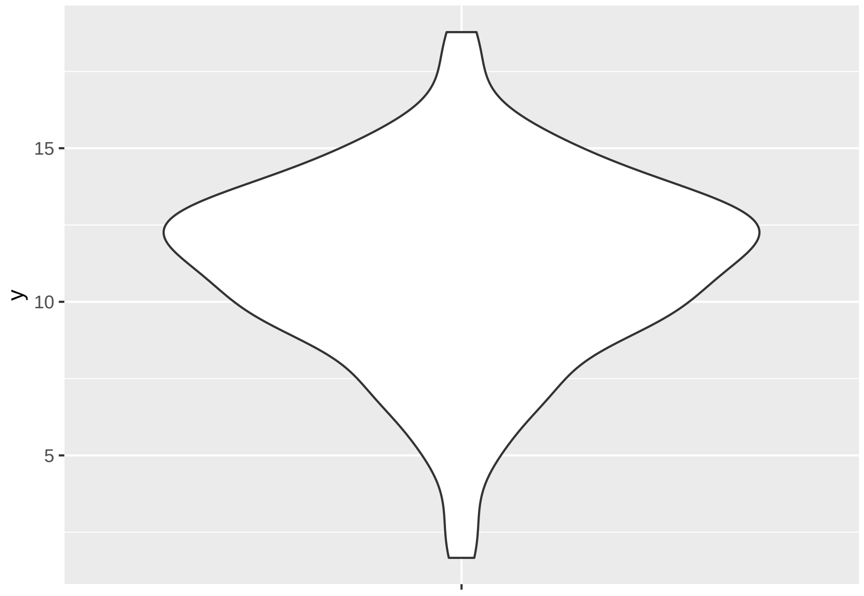
strangely enough doing a boxplot without groups takes a bit of a hack. We have to "invent" a categorical variable:

```
ggplot(df6, aes(x="", y)) +
  geom_boxplot() +
  xlab("")
```



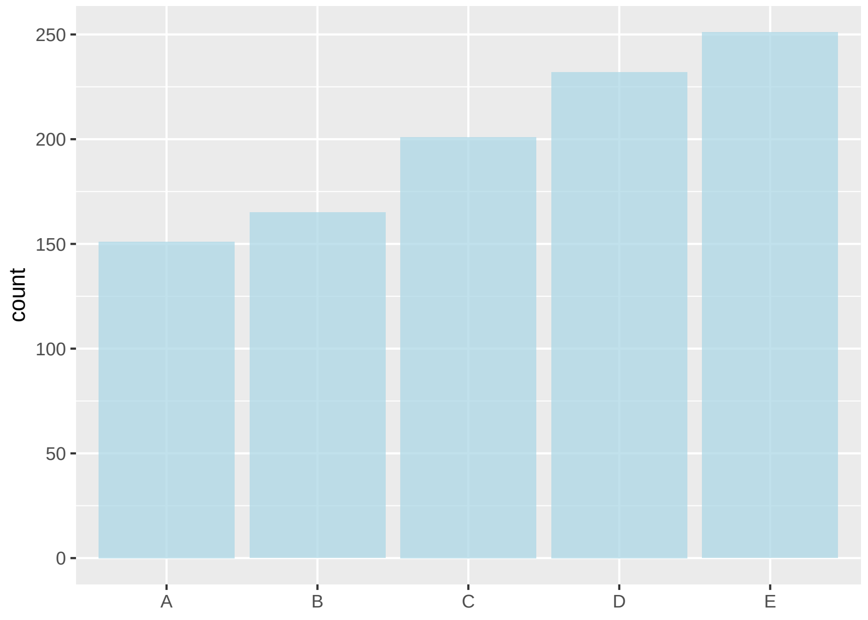
There is a modern version of this graph called a violin plot:

```
ggplot(df6, aes(x="", y)) +
  geom_violin() +
  xlab("")
```



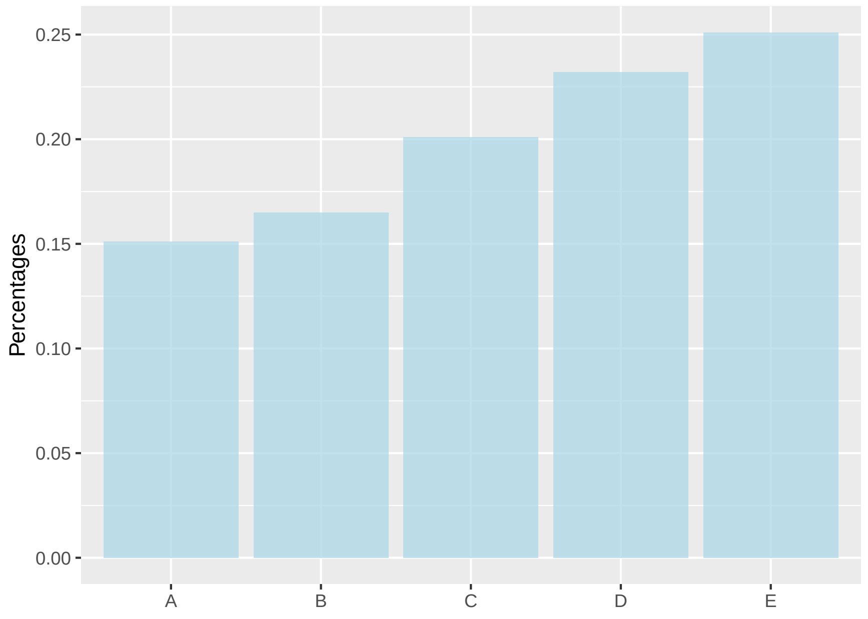
1.6.5 Barcharts

```
x <- sample(LETTERS[1:5],
            size = 1000,
            replace = TRUE,
            prob = 6:10)
df7 <- data.frame(x=x)
ggplot(df7, aes(x)) +
  geom_bar(alpha=0.75, fill="lightblue") +
  xlab("")
```



Say we want to draw the graph based on percentages. Of course we could just calculate them and then do the graph. Here is another way:

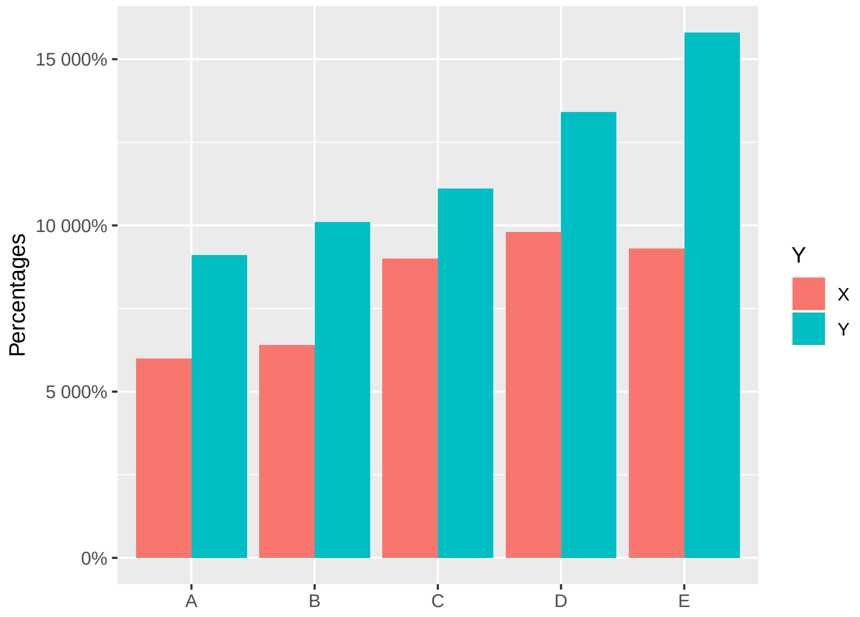
```r
ggplot(df7, aes(x=x)) +
  geom_bar(aes(y=(..count..)/sum(..count..)),
      alpha = 0.75,
      fill = "lightblue") +
  labs(x="", y="Percentages")
```



Notice how this works: in geom_bar we use a new aes, but the values in it are calculated from the old data frame.

Finally an example of a contingency table:

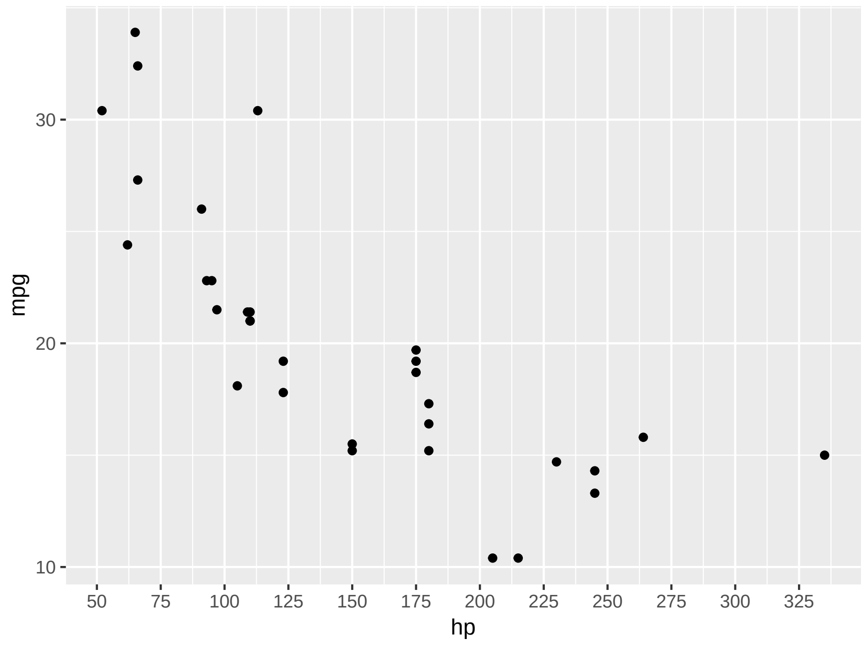
```r
df7$y <- sample(c("X", "Y"),
                size = 1000,
                replace = TRUE,
                prob = 2:3)
ggplot(df7, aes(x=x, fill = y)) +
  geom_bar(position = "dodge") +
    scale_y_continuous(labels=scales::percent) +
    labs(x="", y="Percentages", fill="Y")
```



1.6.6 Axis Ticks and Legend Keys

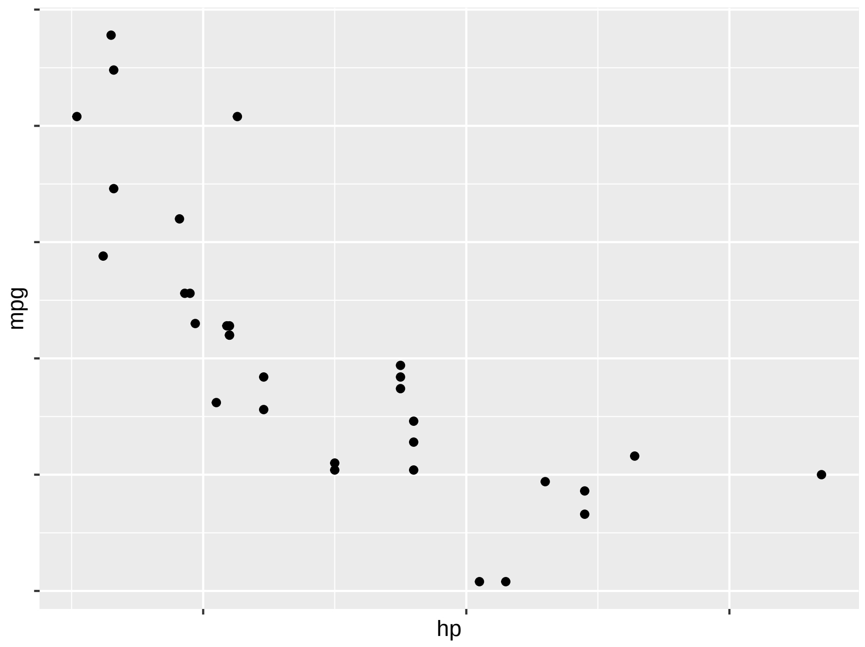
Let's return to the basic plot of mpg by hp. Let's say we want to change the axis tick marks:

```r
ggplot(mtcars, aes(hp, mpg)) +
  geom_point() +
  scale_x_continuous(breaks = seq(50, 350, by=25)) +
  scale_y_continuous(breaks = seq(0, 50, by=10))
```



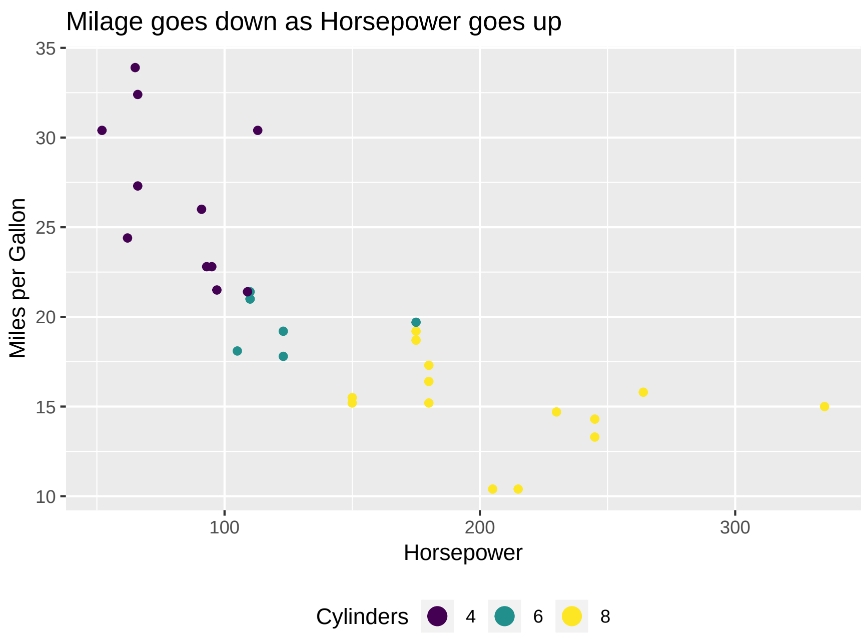
sometimes we want to do graphs without any tick labels. This is useful for example for maps and also for confidential data, so the viewer sees the relationship but can't tell the sizes:

```r
ggplot(mtcars, aes(hp, mpg)) +
  geom_point() +
  scale_x_continuous(labels = NULL) +
  scale_y_continuous(labels = NULL)
```



By default ggplot2 draws the legends on the right. We can however change that. We can also change the appearance of the legend. Recall that the basic graph is in *plt2*. Then

```
plt2 +
  theme(legend.position = "bottom") +
  guides(color=guide_legend(nrow = 1,
                            override.aes = list(size=4)))
```



1.6.7 Saving the graph

It is very easy to save a ggplot2 graph. Simply run

```
ggsave("myplot.pdf")
```

it will save the last graph to disc.

One issue is figure sizing. You need to do this so that a graph looks “good”. Unfortunately this depends on where it ends up. A graph that looks good on a webpage might look ugly in a pdf. So it is hard to give any general guidelines.

If you use R markdown, a good place to start is with the chunk arguments fig.with=6 and out.width=“70%”. In fact on top of every R markdown file I have a chunk with

```
library(knitr)
opts_chunk$set(fig.width=6,
               fig.align = "center",
               out.width = "70%",
               warning=FALSE,
               message=FALSE)
```

so that automatically every graph is sized that way. I also change the default behavior of the chunks to something I like better!

1.7 Important Commands

In the section I will list the most important commands in base R. The list is taken in large part from Hadley Wickham's book Advanced R. Most of them we already discussed. Those we have not you can read up on yourself.

1.7.1 The first functions to learn

? str

1.7.2 Important operators and assignment

%in%, match
=, <-, <<-
$, [, [[, head, tail, subset
with
assign, get

1.7.3 Comparison

all.equal, identical
!=, ==, >, >=, <, <=
is.na, complete.cases
is.finite

1.7.4 Random variables

(q, p, d, r) * (beta, binom, cauchy, chisq, exp, f, gamma, geom, hyper, lnorm, logis, multinom, nbinom, norm, pois, signrank, t, unif, weibull, wilcox, birthday, tukey)

1.7.5 Matrix algebra

crossprod, tcrossprod
eigen, qr, svd
%*%, %o%, outer
rcond
solve

1.7.6 Workspace

ls, exists, rm
getwd, setwd
q

source
install.packages, library, require

1.7.7 Help

help, ?
help.search
apropos
RSiteSearch
citation
demo
example
vignette

1.7.8 Debugging

traceback
browser
recover
options(error =)
stop, warning, message
tryCatch, try

1.7.9 Output

print, cat
message, warning
dput
format
sink, capture.output

1.7.10 Reading and writing data

data
count.fields
read.csv, write.csv
read.delim, write.delim
read.fwf
readLines, writeLines
readRDS, saveRDS
load, save
library

1.7.11 Files and directories

dir
basename, dirname, tools::file_ext
file.path
path.expand, normalizePath
file.choose
file.copy, file.create, file.remove, file.rename, dir.create
file.exists, file.info
tempdir, tempfile
download.file,

1.8 Introduction to ESMA 5015

1.8.1 Simulation

Modern computers allow us to experiment with data.

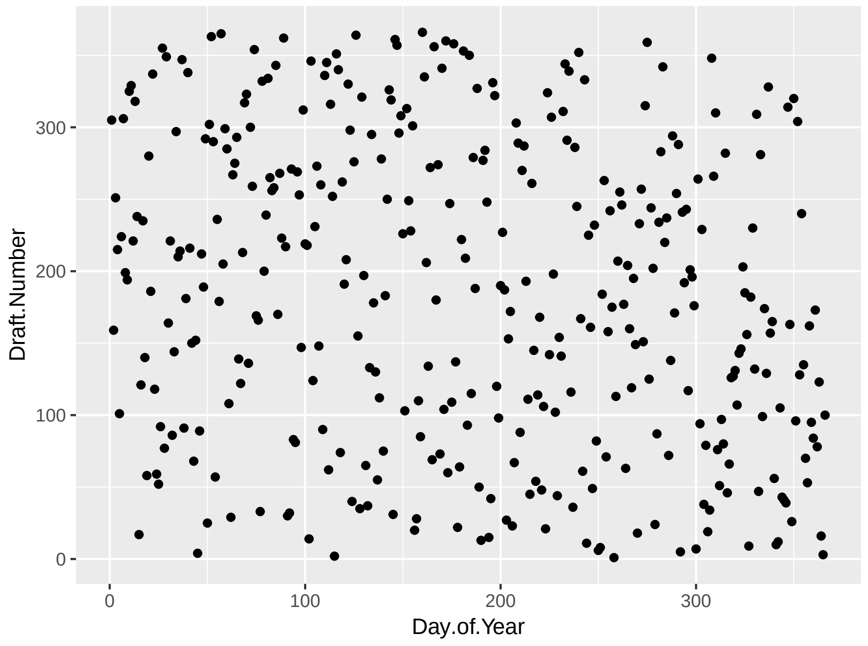
####**Example**

Consider the data set on the 1970's draft.

In R the dataset draft is organized as a matrix, with 366 rows and two columns. Just typing draft shows the content of the dataset.

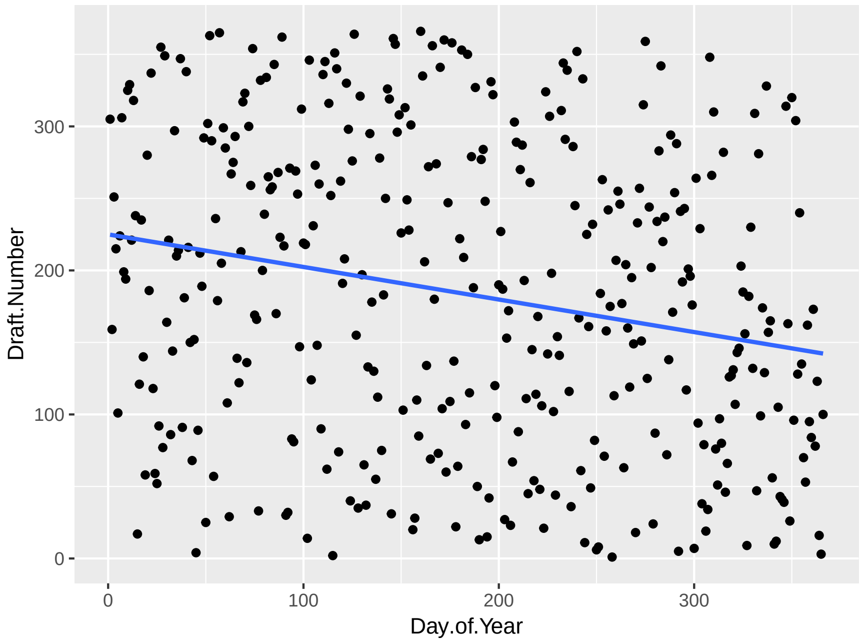
In Statistics we really like to look at pictures. For this type of data the standard one is called the **scatterplot** (really just the data plotted in a Cartesian coordinate system):

```
ggplot(data=draft, aes(Day.of.Year, Draft.Number)) +
  geom_point()
```



It certainly does not appear that there is a relationship between “Day of the Year” and “Draft Number”, but is this really true? As first hint that this may not be so let’s add the least squares regression line:

```r
ggplot(data=draft, aes(Day.of.Year, Draft.Number)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



which needs a few explanations:

- the least squares regression method is a special case of what is called the **linear model**, and in R the calculations are done with the **lm** command. It uses the **formula** structure y~x with the response (y) on the left and the predictor (x) on the right. So lm(Draft.Number~Day.of.Year) finds the least squares regression for y (=Draft.Number) vs x (=Day.of.Year)

Let’s see some details of this **fit**:

```r
summary(lm(Draft.Number~Day.of.Year, data=draft))
```

```
##
## Call:
## lm(formula = Draft.Number ~ Day.of.Year, data = draft)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -210.837  -85.629   -0.519   84.612  196.157
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 225.00922   10.81197  20.811  < 2e-16
## Day.of.Year  -0.22606    0.05106  -4.427 1.26e-05
##
## Residual standard error: 103.2 on 364 degrees of freedom
## Multiple R-squared:  0.05109,    Adjusted R-squared:  0.04849
## F-statistic:  19.6 on 1 and 364 DF,  p-value: 1.264e-05
```

Back to the draft. If there is no relationship between the x and the y variables, then the line should be flat. Ours seems to have a negative slope, so maybe there is a problem. Of course, the specific data we have depends on the **sample** we have drawn, and the line will never be perfectly flat. The question is, how much of a slope is to much? As a second way to look at the data we might find the **correlation coefficient r**

```
cor(draft$Draft.Number, draft$Day.of.Year)
```

```
## [1] -0.2260414
```

Recall some properties of the correlation coefficient:

- $-1 < r < 1$
- r close to 0 means very small or even no correlation (relationship)
- r close to -1 means strong negative relationship
- r close to +1 means strong positive relationship

So we have r = -0.226. But of course the question is whether -0.226 is close to 0, close enough to conclude that all went well. In effect we want to do a **hypothesis test**. This is a method that chooses one of two options. Here these are:

H_0: Draft was random vs. H_a: Draft was not random

We have already decided to use Pearson's correlation coefficient as a measure of "randomness" (or more precisely of "independence" of the two variables "Day of the Year" and "Draft Number". It comes in two versions:

- r: a **statistic**, that is a number computed from a **sample**
- ρ - a **parameter**, that is a number belonging to a **population**

ρ tells us something about the procedure that was used in 1970. If the procedure "worked" as intended we should have $\rho = 0$. r is the actual result of the draft as done in 1970.
We have found r = -0.226, but the real question is whether or not $\rho = 0$, so we can rewrite the hypotheses as follows:

H_0: $\rho = 0$ (= draft was random)
H_a: $\rho \neq 0$ (= draft was not random)

The "traditional" way to answer this question would be to find the **sampling distribution** of r. For example, if it can be assumed that the central limit theorem applies here (and it does), then a number closely related to r has a sampling distribution which is a t distribution. Then the value of this test statistic can be compare to a t table. All of this is implemented in the command **cor.test**:

```r
cor.test(draft$Draft.Number, draft$Day.of.Year)
```

```
##
##  Pearson's product-moment correlation
##
## data:  draft$Draft.Number and draft$Day.of.Year
## t = -4.4272, df = 364, p-value = 1.264e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.3211109 -0.1264617
## sample estimates:
##        cor
## -0.2260414
```

We see that p-value $= 1.3 \times 10^{-5}$, so the test rejects the null hypothesis for any reasonable type I error probability α, and we reject the null hypothesis. It appears that $\rho \neq 0$.

The above works perfectly fine, but in general there could be two problems:

- Just about every statistical method has **assumptions**, what do we do if these are either violated or hard to verify?

- What if we wish to use a test statistic with no known sampling distribution?

In these situations (and many others) we can try to do a simulation:

Doing a **simulation** means recreating the data on a computer under controlled conditions, and then comparing the result with the real-live data. For us this means generating an artificial version of Draft Number, calculate the correlation of this variable and Day and see how large these correlations are. We can do this as follows:

```r
cor(draft$Day.of.Year,
    sample(draft$Day.of.Year))
```

```
## [1] -0.04290002
```

But we need to do this many times, so let's automate the process:

```r
B <- 10000
z <- rep(0, B)
for(i in 1:B)
  z[i] <-cor(draft$Day.of.Year,
    sample(draft$Day.of.Year))
length(z[abs(z) > 0.226])
```

```
## [1] 0
```

and we see that in 10000 runs we NEVER got an r as large as 0.226, so we would conclude that the p value of the test is $< 1/10000$.

Another great feature of R is that we can write our own functions. In the terminology of hypothesis testing we would say that the test has a p-value less than 0.001, and so would reject the null hypothesis. We would conclude that the draft was **not** random.

So, something did go wrong!

1.9 An Example: Monopoly

1.10 The Game

Everybody (I hope) knows the game of Monopoly:

In a game a person's token moves along the board, mostly after the player throws two dice and moves as many fields as the sum of the dice. If he throws a "double" he goes again. Occasionally he lands on Chance or Community Chest and picks a card that may direct him to a specific field. Finally, if he throws three "doubles" in a row or lands on "Go to Jail" he does just that. If he lands on a field he can buy the property if it is not already owned or pay rent to the owner if it is.

After a while in the game a player might own all the properties of the same "color", and he is said to have a "monopoly". Then he can begin to build houses, up to a hotel, on this property, and get much more rent from the other players.

Much of the game is "automatic", that is the player really does not make any decisions. Even the buying of property is (almost) automatic because it is understood that the more property he has, the better off he is. The one time strategy comes into play is when two players have the properties to form two monopolies, but need to "trade" some of them. As an example say player "A" owns New York Ave, Tennessee Ave (orange) and Indiana Ave (red). "B" owns St. James Place (orange), Kentucky Ave and Illinois Ave (red). So if they exchanged St. James Place for Indiana Ave, both would have a monopoly and can build houses. The question is, should they?

One thing is clear: the "red" properties have a higher rent than the orange ones (\$950 vs \$1050). But rent is only paid if somebody steps on the property, so we also need to know the probability of that happening. This would be easy (all would be equally likely) except for those Chance and Community Chest cards, and the "Go to Jail" option. Notice that everytime somebody gets out of jail, the probability to step on an orange field is

P(6, 8 or 9) = (5+7+8)/36 = 0.55

So how can we find these probabilities? Really, the only way to do this is via a simulation.

1.11 The Simulation Setup

How do we set this up as a simulation? First we need to get the “board into R”. The board has 40 fields. Each field has a name (“Go”, " Mediterreanen Ave", ..) . Many, but not all have a color. Finally there is the rent to be paid (for a hotel). So, we will represent the board as a vector of length 40. In *monopoly* this setup is done at the beginning:

```
monopoly <- function (n = 1e+05) {
  fields <- c("Go", "Mediterranean", "Community Chest",
    "Baltic", "Income Tax", "Reading",  "Oriental",
    "Chance", "Vermont", "Connecticut", "Jail",
    "St. Charles", "Electic Company", "States",
    "Virginia", "Pennsylvania RR", "St. James",
    "Community Chest", "Tennessee", "New York",
    "Free Parking", "Kentucky", "Chance", "Indiana",
    "Illinois", "R&B", "Atlantic", "Ventnor",
    "Water Works", "Marvin Gardens", "Go to Jail",
    "Pacific", "North Carolina", "Community Chest",
    "Pennsylvania", "Short", "Chance",
    "Park", "Luxery Tax", "Boardwalk")
  colors <- c("Go", "Purple", "Community Chest",
    "Purple", "Income Tax", "RR", "Light Blue",
    "Chance", "Light Blue", "Light Blue", "Jail",
    "Wine", "Utility", "Wine", "Wine", "RR", "Orange",
    "Community Chest", "Orange", "Orange",
    "Free Parking", "Red", "Chance", "Red", "Red", "RR",
    "Yellow", "Yellow", "Utility", "Yellow",
    "Go to Jail", "Green", "Green", "Community Chest",
    "Green", "RR", "Chance", "Blue", "Luxery Tax","Blue")
  rent <- c(0, 250, 0, 450, 0, 200, 550, 0, 550, 600, 0,
    750, 10, 750, 900, 200, 950, 0, 950, 1000, 0, 1050,
    0, 1050, 1100, 200, 1150, 1150, 10, 1200, 0, 1275,
    1275, 0, 1400, 200, 0, 1500, 0, 2000)
  properties <- 1:10
  names(properties) <- c("Purple", "Light Blue", "Wine",
    "Orange", "Red", "Yellow", "Green", "Blue", "RR",
    "Utility")
  chancemove <- function(x) {
        z <- sample(1:16, 1)
        if (z == 1) {
            if (x < 3) {
                x <- x + 39 - 3
            }
            else {
                x <- x - 3
            }
        }
    if (z == 2)
        x <- 0
    if (z == 3)
        x <- 39
    if (z == 4)
        x <- 5
    if (z == 5)
        x <- 11
    if (z == 6)
        x <- 24
    if (z == 7)
        x <- 10
    if (z == 8) {
        x <- 12
        if (x > 21 & x < 39)
            x <- 28
    }
    if (z == 9) {
        if (x > 0 & x < 9)
            x <- 5
        if (x > 10 & x < 19)
            x <- 15
        if (x > 20 & x < 29)
            x <- 25
        if (x > 30 & x < 39)
            x <- 35
    }
    x
}
communitymove <- function(x) {
    z <- sample(1:15, 1)
    if (z == 1)
        x <- 0
    if (z == 2)
        x <- 10
    x
}
visits <- rep(0, 40)
rounds <- 0
jail.visit <- TRUE
double <- 0
x <- 0
for (i in 1:n) {
    oldx <- x
    dice <- sample(1:6, size = 2, replace = T)
    if (dice[1] == dice[2])
```

```
            double <- double + 1
        else double <- 0
        if (double == 3) {
            x <- 10
            jail.visit <- F
        }
        x <- (x + sum(dice))%%40
        if (x == 10) {
            jail.visit <- TRUE
        }
        else {
            if (fields[x + 1] == "Chance")
                x <- chancemove(x)
            if (fields[x + 1] == "Community Chest")
                x <- communitymove(x)
            if (x == 30)
                x <- 10
            if (x == 10)
                jail.visit <- F
        }
        if (x == 10 & !jail.visit) {
            for (j in 1:3) {
                visits[x + 1] <- visits[x + 1] + 1
                dice <- sample(1:6, size = 2, replace = T)
                if (dice[1] == dice[2])
                  break
            }
            x <- x + sum(dice)
        }
        visits[x + 1] <- visits[x + 1] + 1
        if (oldx >= x)
            rounds <- rounds + 1
    }
    print(paste("Number of Rounds:", rounds), quote=FALSE)
    names(visits) <- fields
    payed <- rent * visits
    payed[c(12, 28) + 1] <- 70 * visits[c(12, 28) + 1]
    p <- visits/sum(visits)
    print("Percentage of Visits:", quote=FALSE)
    print(round(sort(p, decreasing = T) * 100, 3), quote=FALSE)
    print(" ", quote=FALSE)
    for (i in names(properties)) properties[i] <- sum(payed[colors ==
        i])
    print("Mean Rent per Round by Monopoly:", quote=FALSE)
    print(properties/rounds, quote=FALSE)
    return(list(p = p, rent = properties/rounds))
```

```
}
```

How do we represent the position of a token on the board? One way would be as one of the numbers 1-40, but another option is to use 0-39 instead. An advantage of this is it makes the move counting easier, because R has a modulus function built in. So say we are on field 35, and roll (2, 4). Then the next field is

```
(35 + 2 + 4)%%40
```

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

which is Baltic Ave.

But there is also a problem: In R vectors cannot be indexed by 0, fields[0] would give an error message. fields[1] is “Go”, not “Baltic Ave”. So if we use x = 0-39, we need to use fields[x+1] to find out where we are.

So the basic move is done with the R commands:

```
dice <- sample(1:6, size=2,replace=TRUE) # Throw the dice
x <- (x+sum(dice))%%40 # Move along board
```

After a move we need to check whether we are on

- “Go to Jail” (if x=30)
- “Cummunity Chest” (fields[x+1]==“Community Chest”)
- “Chance” (fields[x+1]==“Chance”)

If we are on x=30 we go to Jail (x=10). If we are on “Community Chest” we pick a card and move as directed, which is done by the function *communitymove*. Similarly for “Chance”.

Note that there are 15 kinds of cards in the cummunity chest, but only two lead to a move of the token, Similarly 9 of the 16 chance cards lead to a move. Because we are interested in the probabilities of visiting streets we can ignore the other cards.

We need to include one more item in our simulation: if a player lands in jail he has two options: he can get out right away (paying $50 or using a card) or he can throw the dice. If they come up doubles he gets out, moving the sum of the doubles. On the third round he gets out for sure (paying $50). Of course, if he is just “visiting” the jail he moves on normaly.

What should a player do? Clearly in the early rounds he wants to get out of jail as quickly as possible (so he has a chance of buying more property) but in the later rounds it is much better to stay in jail (where rent is free). Because in our simulation we are interested in the later stages of a game, this will be our policy.

So the idea is to have one token start at Go (x=0) and let it move over the board according to the rules for many many rounds.

What should we keep track off? Obviously we need to know where we have been, which is done in

```r
visits <- rep(0, 40)
```

We also keep track of the "rounds", that is how often we moved around the board. This happens everytime x gets "reset" by the mod function, so if we call "oldx" the current position, "x" the next position then if oldx>x we have gone around once more.

Finally we keep track of the money paid, in

```r
rent <- rep(0, 40)
```

Here we assume that every property has a hotel, all the railroads are owned by the same person and all the "Utilities" are also owned by the same person.

What should we print out? We are interested in the fields that make money, and their probabilities, so we sort these by the probabilities and show the result. We also show the "rent per round", this for the different monopolies.

```r
tmp <- monopoly()
```

```
## [1] Number of Rounds: 20925
## [1] Percentage of Visits:
##             Jail        Illinois    Free Parking              Go
##           10.946           2.981           2.928           2.859
##        Tennessee Electic Company        New York       St. James
##            2.843           2.831           2.791           2.773
##              R&B         Reading         Indiana        Kentucky
##            2.771           2.696           2.657           2.595
##      St. Charles       Boardwalk         Ventnor         Pacific
##            2.549           2.544           2.541           2.530
##         Atlantic     Water Works  North Carolina  Marvin Gardens
##            2.522           2.506           2.480           2.429
## Community Chest        Virginia           Short    Pennsylvania
##            2.381           2.370           2.351           2.350
## Pennsylvania RR          States      Income Tax Community Chest
##            2.322           2.256           2.242           2.227
##          Vermont        Oriental     Connecticut          Baltic
##            2.202           2.173           2.146           2.073
##       Luxery Tax            Park   Mediterranean Community Chest
##            2.070           2.050           2.007           1.766
##           Chance          Chance          Chance      Go to Jail
##            1.342           0.955           0.945           0.000
## [1]
## [1] Mean Rent per Round by Monopoly:
##      Purple Light Blue       Wine     Orange        Red     Yellow
##    75.17802  193.61051  300.67384  425.90442  460.87933  457.95699
##       Green       Blue         RR    Utility
##   507.23656  427.81362  106.29391   19.57993
```

```
tmp
```

```
## $p
##                Go   Mediterranean Community Chest          Baltic
##       0.028586018     0.020069482     0.017662238     0.020726003
##        Income Tax         Reading        Oriental          Chance
##       0.022422015     0.026962952     0.021729021     0.009546909
##           Vermont     Connecticut            Jail     St. Charles
##       0.022020808     0.021464589     0.109456638     0.025494898
## Electic Company          States        Virginia Pennsylvania RR
##       0.028312468     0.022558791     0.023698584     0.023224430
##         St. James Community Chest       Tennessee        New York
##       0.027728893     0.023808004     0.028431006     0.027911260
##      Free Parking        Kentucky          Chance         Indiana
##       0.029279012     0.025950816     0.013422207     0.026570863
##          Illinois             R&B        Atlantic         Ventnor
##       0.029807876     0.027710657     0.025221348     0.025412833
##       Water Works  Marvin Gardens      Go to Jail         Pacific
##       0.025057218     0.024291276     0.000000000     0.025303413
##    North Carolina Community Chest    Pennsylvania           Short
##       0.024801904     0.022267003     0.023497980     0.023507099
##            Chance            Park      Luxery Tax       Boardwalk
##       0.009446608     0.020498044     0.020698648     0.025440188
##
## $rent
##      Purple Light Blue       Wine     Orange        Red     Yellow
##    75.17802  193.61051  300.67384  425.90442  460.87933  457.95699
##       Green       Blue         RR    Utility
##   507.23656  427.81362  106.29391   19.57993
```

Here are the streets sorted by the probabilities:

```
df <- data.frame(Property=names(tmp$p),
          Probability=round(as.numeric(tmp$p), 4))
df <- df[order(df$Probability, decreasing = TRUE), ]
row.names(df) <- NULL
kable.nice(df)
```

Property	Probability
Jail	0.1095
Illinois	0.0298
Free Parking	0.0293
Go	0.0286
Tennessee	0.0284
Electic Company	0.0283
New York	0.0279
St. James	0.0277
R&B	0.0277
Reading	0.0270
Indiana	0.0266
Kentucky	0.0260
St. Charles	0.0255
Ventnor	0.0254
Boardwalk	0.0254
Pacific	0.0253
Atlantic	0.0252
Water Works	0.0251
North Carolina	0.0248
Marvin Gardens	0.0243
Community Chest	0.0238
Virginia	0.0237
Pennsylvania	0.0235
Short	0.0235
Pennsylvania RR	0.0232
States	0.0226
Income Tax	0.0224
Community Chest	0.0223
Vermont	0.0220
Oriental	0.0217
Connecticut	0.0215
Baltic	0.0207
Luxery Tax	0.0207
Park	0.0205
Mediterranean	0.0201
Community Chest	0.0177
Chance	0.0134
Chance	0.0095
Chance	0.0094
Go to Jail	0.0000

How about the monopolies?

```r
df <- data.frame(Monopoly=names(tmp$rent),
          Rent=round(as.numeric(tmp$rent)))
df <- df[order(df$Rent, decreasing = TRUE), ]
row.names(df) <- NULL
kable.nice(df)
```

Monopoly	Rent
Green	507
Red	461
Yellow	458
Blue	428
Orange	426
Wine	301
Light Blue	194
RR	106
Purple	75
Utility	20

Notice that owning the four railroads is worth more than owning the "Purple" monopoly ($105 vs $76 per round).

1.11.1 Basic Question

So, how about our original question, should players "A" and "B" exchange St. James Place for Indiana Ave? If they do "A" then has the orange monoploy and "B" has the red one. So "A" should make about $425 per round and "B" should make about $458. On average in each round "B" makes $33 more than "A". So this looks like "A" should not trade at all!

Well, there are other considerations. For example, say the players agree to play for another 10 rounds, and "B" offers "A" an extra $330. Now it is a fair trade.

Another thing to consider is the amount of money "A" and "B" have. For example, if "B" is just about broke but "A" has plenty of money, "B" should not trade because he won't be able to build any houses and so he won't get any rent anyway.

Let's assume instead "A" has $a and "B" has $b. Then a trade would be fair if either of the two is equally likely to win at some point in the future. How can we find out what the probability of say "A" winning is? Well if "A" starts with $a then

- he has to pay for building the hotels, after which he has a-sum(cost[colors=="Orange"]) left

then after one round

- his wealth will increase by rent[“Orange”] with probability p[“Orange”] (if “B” lands on an orange field)
- his wealth will decrease by rent[“Red”] with probability p[“Red”] (if he lands on red)
- his wealth remains the same (if neither lands on the others property)

Similar of course for B.

Note that the construction costs are $250 per hotel in “row 1”, $500 in “row 2” and so on.

1.11.2 Routines

So now we can run a simulation, playing many games as above until one or the other is broke, in

```
monopoly1()
```

This routine is written to be easily understood, but it is a little slow. We need to run this a number of times, and so it is important to speed it up a bit. This is done in

```
monopoly2 <- function (a, b, A = "Orange", B = "Red",
                       N = 10000, Show = FALSE)
{
  fields <- c("Go", "Mediterranean", "Community Chest",
    "Baltic", "Income Tax", "Reading",  "Oriental",
    "Chance", "Vermont", "Connecticut", "Jail",
    "St. Charles", "Electic Company", "States",
    "Virginia", "Pennsylvania RR", "St. James",
    "Community Chest", "Tennessee", "New York",
    "Free Parking", "Kentucky", "Chance", "Indiana",
    "Illinois", "R&B", "Atlantic", "Ventnor",
    "Water Works", "Marvin Gardens", "Go to Jail",
    "Pacific", "North Carolina", "Community Chest",
    "Pennsylvania", "Short", "Chance",
    "Park", "Luxery Tax", "Boardwalk")
  colors <- c("Go", "Purple", "Community Chest",
    "Purple", "Income Tax", "RR", "Light Blue",
    "Chance", "Light Blue", "Light Blue", "Jail",
    "Wine", "Utility", "Wine", "Wine", "RR", "Orange",
    "Community Chest", "Orange", "Orange",
    "Free Parking", "Red", "Chance", "Red", "Red", "RR",
    "Yellow", "Yellow", "Utility", "Yellow",
    "Go to Jail", "Green", "Green", "Community Chest",
    "Green", "RR", "Chance", "Blue", "Luxery Tax","Blue")
  rent <- c(0, 250, 0, 450, 0, 200, 550, 0, 550, 600, 0,
    750, 10, 750, 900, 200, 950, 0, 950, 1000, 0, 1050,
    0, 1050, 1100, 200, 1150, 1150, 10, 1200, 0, 1275,
    1275, 0, 1400, 200, 0, 1500, 0, 2000)
  names(rent) = fields
p <- c(0.02929, 0.02059, 0.01787, 0.02088, 0.02236,
        0.02682, 0.0214, 0.00968, 0.02215, 0.0219,
        0.10723, 0.02568, 0.02817, 0.02178, 0.0248,
        0.02356, 0.02729, 0.02344, 0.02855, 0.02821,
        0.02852, 0.026, 0.01376, 0.02583, 0.02976,
        0.02719, 0.02556, 0.02581, 0.02524, 0.02499,
        0, 0.02547, 0.02472, 0.02213, 0.0236, 0.02409,
        0.00962, 0.02038, 0.02058, 0.02508)
  cost = rep(1:4 * 250, rep(10, 4))
  A.properties <- fields[colors == A]
  B.properties <- fields[colors == B]
  A.win <- 0
  a <- a - sum(cost[colors == A])
  b <- b - sum(cost[colors == B])
  if (Show) {
      print("Money after building Hotels:")
      print(c(a, b))
      print("Rent to be paid:")
      print(rent[c(A.properties, B.properties)])
  }
  if (a < 0 | b < 0) {
      print("No money to build!")
      return(NA)
  }
  a <- rep(a, N)
  b <- rep(b, N)
  repeat {
      n <- length(a)
      x <- sample(1:40, size=n, replace=TRUE, prob=p)
      for (j in B.properties) {
          visit <- ifelse(fields[x] == j, TRUE, FALSE)
          a[visit] <- a[visit] - rent[j]
          b[visit] <- b[visit] + rent[j]
      }
      y <- sample(1:40, size=n, replace=TRUE, prob=p)
      for (j in A.properties) {
          visit <- ifelse(fields[y] == j, TRUE, FALSE)
          a[visit] <- a[visit] + rent[j]
          b[visit] <- b[visit] - rent[j]
      }
      if (min(a) < 0 | min(b) < 0) {
          if (length(b[b < 0]) > 0)
              A.win <- A.win + length(b[b < 0])
      }
      still.playing <- ifelse(a>=0 & b>=0, TRUE, FALSE)
```

```
        if (sum(still.playing) == 0)
            break
        a <- a[still.playing]
        b <- b[still.playing]
    }
    A.win/N
}
```

Here we essentially run all N simulations simultaneously.

Playing around with this routine we can find out what a fair trade value is.

Say both have $3000, then

```
monopoly2(3000, 3000)
```

```
## [1] 0.6574
```

"A" wins with probability 66%. With a little bit of trial and error we can find that if as part of the trade "A" gives "B" $225, then they both have the same probability of going broke:

```
monopoly2(3000-225, 3000+225)
```

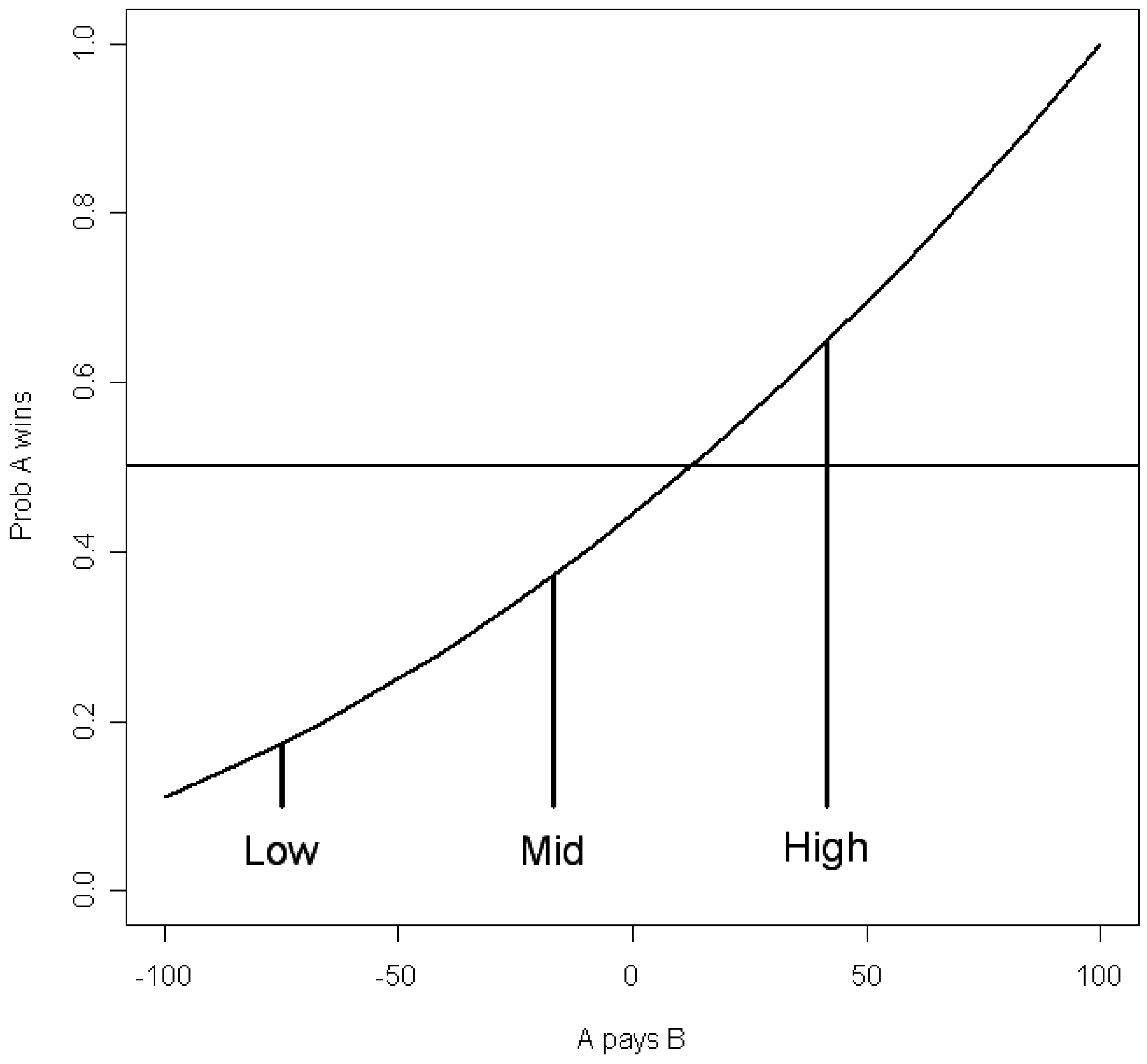
```
## [1] 0.5249
```

This is a very strange result: the orange properties are before the red ones, with lower rents, yet A should pay B? The explanation is this: B needs to pay more money ($750 more) to build his hotels, and so he can get broke by just a few visits to A. If both started out with $6000, it is B who needs to pay A, some $300.

In the example above we have used trial and error to find the right trade-value. Can we write a routine that does this for us? Here is an idea:

- find the probability of A winning without any money changing hands, call it p_0
- if $p_0 > 0.5$ A needs to pay B, if $p_0 < 0.5$ B needs to pay A.
- say $p_0 > 0.5$, let m be some amount of money (for example 10% of \$a, let p_m be the probability of A winning for a trade-value of m. If $p_m < 0.5$ the correct trade value is in [0, m], otherwise raise m and try again.
- once we have found an interval $[m_1, m_2]$ that contains the fair trade-value, check the midpoints and half the intervals.
- if $p_0 < 0.5$ do the same the other way around.
- stop when $p_m = 0.5$ (just about)

This is an example of one of my favorite algorithms, the **bisection** algorithm. here is the basic idea:



so we change low to mid and run again. This algorithm works great if the function is monotone. It is very slow but extrememly stable. In our case because we don't have an explicit expression for the function (and therefore no derivative) it's probably as good as we can do.

There is one difficulty: our routine is a simulation and will give slightly different values in different runs. Here is what we need to consider:

Let's say we do N = 10000 runs. On each run A either wins or looses. Let the rv $Z_i = 1$ if A wins, 0 otherwise. Then $Z_1, .., Z_N$ is a sequence of independent Bernoulli rv's with success probability p_m. Therefore

$$\hat{p} = \frac{1}{N}\sum_{i=1}^{N} Z_i$$

$$E[\hat{p}] = p_m$$

$$sd(\hat{p}) = \sqrt{\frac{p_m(1-p_m)}{N}} = \sqrt{\frac{0.5(1-0.5)}{10000}} = \frac{1}{200}$$

$$3sd(\hat{p}) = \frac{3}{200} = 0.015$$

so if we run a simulation that gives $p_m \in [0.485,0.515]$ the "true" p_m might just be 0.5 and the corresponding m is a fair-trade value.

This is implemented in

```
monopoly3(5000, 3000, Show=TRUE)
```

```
## [1] "m= 0   pm= 0.7688"
## [1] "m= 500 ,  pm= 0.6134"
## [1] "m= 1000 ,  pm= 0.5587"
## [1] "m= 1500 ,  pm= 0.4069"
## [1] "m= 1250 ,  pm= 0.4462"
## [1] "m= 1125 ,  pm= 0.5162"
## [1] "m= 1187.5 ,  pm= 0.4879"
## A should pay B: $ 1187.5
```

```
## [1] 1187.5
```

This routine is not yet very good. To see whether your routine has problems it is often a good idea to run "extreme" cases. For example, try

```
monopoly3(10000,3000,"Purple","Green", Show=T)
```

```
## [1] "m= 0   pm= 0.3424"
## [1] "No money to build!"
```

```
## Error in if (abs(pm - 0.5) < 0.015) {: missing value where TRUE/FALSE needed
```

but the error message makes no sense, to begin with A and B had enough money to build. Any idea what's wrong with my routine?

So, next time you play monopoly, bring your laptop and at the right time run our little routine.

If you want to know more about strategy in Monopoly, check out these websites:

[Amnesta] (http://www.amnesta.net/other/monopoly)

[Collins] (http://www.tkcs-collins.com/truman/monopoly/monopoly.shtml)

2 Probability

2.1 Introduction

The probability of rain tomorrow is 0.3. What does that mean?

We usually find probabilities in one of three ways:

- empirically through many repetitions of an experiment - relative frequency interpretation

- through reasoning about outcomes etc. - classical interpretation

- by using our intuition and experience - subjective interpretation

2.1.0.1 Example - coin tossing

what is the probability of getting "heads" when tossing a fair coin?

- relative frequency interpretation: take a coin and flip it! the South African mathematician Jon Kerrich, while in a German POW camp during WWII tossed a coin 10000 times. Result 5067 heads, for a probability of 0.5067

- classical interpretation: This experiment has two possible outcomes - heads and tails. Fair means they are equally likely, so p=P("heads")=P("tails")=0.5

- subjective interpretation: I think it's 1/2.

An **experiment** is a well-defined procedure that produces a set of outcomes. For example, "roll a die"; "randomly select a card from a standard 52-card deck"; "flip a coin" and "pick any moment in time between 10am and 12 am" are experiments.

A **sample space** is the set of outcomes from an experiment. Thus, for "flip a coin" the sample space is {H, T}, for "roll a die" the sample space is {1, 2, 3, 4, 5, 6} and for "pick any moment in time between 10am and 12 am" the sample space is [10, 12].

An **event** is a subset, say A, of a sample space S. For the experiment "roll a die", an event is "obtain a number less than 3". Here, the event is {1, 2}.

If all the outcomes of a sample space S are equally likely and if A is an event, then the probability of A is:

$$P(A) = \frac{N(A)}{N(S)} = \frac{\text{number of ways of "success"}}{\text{number of total ways}}$$

So, the probability of an event, say A, is the *ratio of success to total.*

2.1.0.2 Example

flipping a coin what is the probability of a heads?

The total number of outcomes is 2 and the number of ways to be successful is 1. Thus, P(heads) = 1/2.

2.1.0.3 Example

consider randomly selecting a card from a standard 52-card deck: what is the probability of getting a king?

the total number of outcomes is 52 and of these outcomes 4 would be successful. So, P(king) = 4/52.

2.1.0.4 Example

What is the probability of a sum of 8 when rolling two fair dice?

Solution 1: Sample space is

(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)
(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)

There are 5 pairs that have a sum of 8, so P(sum of 10)=5/36=0.1389

Solution 2: The sum can be any number from 2 to 12, the sample space is {2,3,4,..,11,12}. There are 11 numbers in the sample space, one of them is 8, so P(sum of 10)=1/11=0.091

Which is right?

Let's do a simulation to see which answer is correct. use command "sample" to randomly pick an element from a set

args(sample) shows you the correct syntax of the "sample command sample(1:6, 2, TRUE) picks two numbers from 1 to 6 **with repetition**

sum(sample(1:6, 2, TRUE)) finds their sum, just what we want

```
z <- rep(0, 10000) #generates a vector of length 10000
for(i in 1:10000)
  z[i] <- sum(sample(1:6, 2, TRUE)) #repeats our experiment 10000 times
length(z[z==8])/10000 #finds the proportion of "8's" in z
```

```
## [1] 0.1371
```

But why is it right?

2.1.1 Fundamentals

The definition above works well as long as S is finite but breaks down if S is infinite. Instead modern probability, like geometry, is built on a small set of basic rules called axioms, derived in the 1930's by Kolmogorov. They are:

$$\begin{aligned}
&\text{Axiom 1: } 0 \le P(A) \le 1\\
&\text{Axiom 2: } P(S) = 1\\
&\text{Axiom 3: } P(\cup_{i=1}^n A_i) = \sum_{i=1}^n P(A_i)
\end{aligned}$$

if $A_1, ..., A_n$ are mutually exclusive

Example : Derive the formula above (for a finite sample space) from these axioms.

Solutions: say we have a sample space $S = \{e_1, ..., e_n\}$ and an event $A = \{e_{k_1}, ..., e_{k_m}\}$. Then:

$$\begin{aligned}
&P(S) = P(\{e_1,...,e_n\}) = P(\cup_{i=1}^n \{e_i\}) = \textstyle\sum_{i=1}^n P(\{e_i\}) && \text{Axiom 3}\\
&= \textstyle\sum_{i=1}^n p = np = 1 \quad \Rightarrow \quad p = \frac{1}{n} && \text{Axiom 2}\\
&P(A) = P(\{e_{k_1},...,e_{k_m}\}) = P(\cup_{j=1}^m \{e_{k_j}\}) = \textstyle\sum_{j=1}^m P(\{e_{k_j}\}) && \text{Axiom 3}\\
&= \textstyle\sum_{j=1}^m = m\frac{1}{n} = \frac{m}{n}
\end{aligned}$$

### Some useful formulas

Complement: $P(A) = 1 - P(A^c)$

Example : A fair coin is tossed 5 times. What is the probability of at least one "Heads"?

Sample Space S={(H,H,H,H,H), (H,H,H,H,T), ... , (T,T,T,T,T)}

S has $2^5 = 32$ elements

P(at least one "Heads") =
1 - P("No Heads") =

1 - P({(T,T,T,T,T)}) =
1 - 1/36 = 35/36

Addition Formula: $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

2.1.1.1 Example

We roll two fair dice. What is the probability of a sum of 5 or 8, or highest number on either die is a 3?

Sample Space is above.

Event A = {(1,4), (2,3), (3,2), (4,1), (2,6), (3,5), (4,4), (5,3), (6,2)}, n(A) = 9

Event B = {(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)}, n(B) = 9

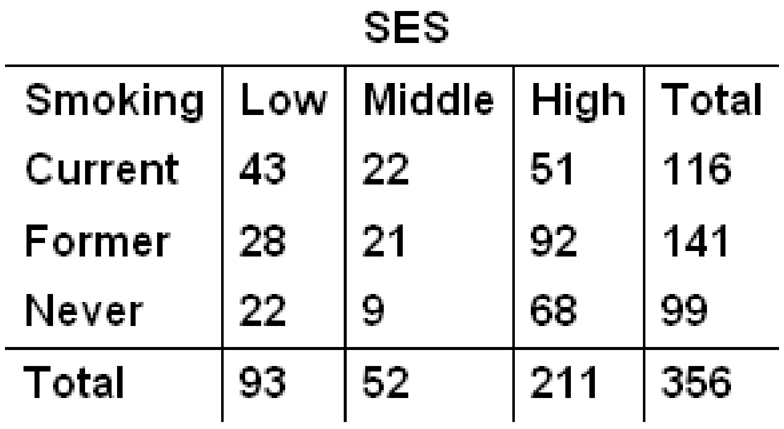
Event $A \cap B = \{(2,3),(3,2)\}$, $n(A \cap B) = 2$

$P(A \cup B) = P(A) + P(B) - P(A \cap B) =$
$9/36 + 9/36 - 2/36 = 16/36 = 4/9$

2.2 Conditional Probability and Independence

In a study of heart disease in male federal employees, researchers classified 356 volunteer subjects according to their socioeconomic status (SES - coded as Low, Middle, High) and their smoking status (Smoking - coded as Never, Former and Current).

Here is the data:



	SES			
Smoking	Low	Middle	High	Total
Current	43	22	51	116
Former	28	21	92	141
Never	22	9	68	99
Total	93	52	211	356

What is the probability that a randomly selected volunteer in this study is a former smoker with a high socioeconomic status?

Answer is easy (92/356) but let's do this slowly:

Event A = "Former Smoker" Event B = "high socioeconomic status"

$$P(A \cap B) = \frac{92}{356}$$

What is the probability that a randomly selected volunteer in this study is a former smoker?

$$P(A) = \frac{141}{356}$$

If we know that a randomly selected volunteer is a former smoker, what is the probability that he also has a high socioeconomic status?

Again the answer is clear: 92/141

This kind of probability is called a **conditional** probability. We use the notation P(high|former) = P(B|A). Note:

$$P(A|B) = \frac{92}{141} = \frac{92/356}{141/356} = \frac{P(A \cap B)}{P(B)}$$

In general we can find conditional probabilities using the formula

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Note: this only works if P(B)>0

2.2.1 Multiplication Rule

A simple manipulation of the equation above yields

$$P(A \cap B) = P(A|B)P(B)$$

2.2.1.1 Example

You draw two cards from a standard 52-card deck. What is the probability to draw 2 Aces?

Solution:

Let A = "First card drawn is an ace"
Let B = "Second card drawn is an ace"

Then

P(both cards drawn are aces) =

P(first card drawn is an ace and second card drawn is an ace) =

$$P(A \cap B) = P(A) \cdot P(B|A) = \frac{4}{52} \cdot \frac{3}{51}$$

It's easy to extend this to more than two events: What is the probability of drawing 4 aces when drawing 4 cards?

Let A_i = i^{th} card drawn is an ace"

Then

$$\begin{aligned}
&P(\text{all 4 cards drawn are aces}) = \\
&P(\text{first card drawn is an ace and second card drawn is an ace and} \\
&\text{third card drawn is an ace and fourth card drawn is an ace}) = \\
&P(A_1 \cap A_2 \cap A_3 \cap A_4) = \\
&P(A_1) \cdot P(A_2 \mid A_1) \cdot P(A_3 \mid A_1 \cap A_2) \cdot P(A_4 \mid A_1 \cap A_2 \cap A_3) = \\
&= \frac{4}{52} \cdot \frac{3}{51} \frac{2}{50} \cdot \frac{1}{49}
\end{aligned}$$

even a little more complicated: In most Poker games you get in the first round 5 cards (Later you can exchange some you don't like but we leave that out). What is the probability that you get 4 aces?

Again let $A_i = i^{th}$ card drawn is an ace"

Then

$$\begin{aligned}
&P(\text{4 of the 5 cards drawn are aces}) = \\
&P(\left(A_1 \cap A_2 \cap A_3 \cap A_4 \cap \overline{A}_5\right) \cup \left(A_1 \cap A_2 \cap A_3 \cap \overline{A}_4 \cap A_5\right) \cup \left(A_1 \cap A_2 \cap \overline{A}_3 \cap A_4 \cap A_5\right) \\
&\cup \left(A_1 \cap \overline{A}_2 \cap A_3 \cap A_4 \cap A_5\right) \cup \left(\overline{A}_1 \cap A_2 \cap A_3 \cap A_4 \cap A_5\right)) = \\
&\frac{4}{52} \cdot \frac{3}{51} \frac{2}{50} \cdot \frac{1}{49} \cdot \frac{48}{48} + \frac{4}{52} \cdot \frac{3}{51} \frac{2}{50} \cdot \frac{48}{49} \cdot \frac{1}{48} + \frac{4}{52} \cdot \frac{3}{51} \frac{48}{50} \cdot \frac{2}{49} \cdot \frac{1}{48} + \frac{4}{52} \cdot \frac{48}{51} \frac{3}{50} \cdot \frac{2}{49} \cdot \frac{1}{48} + \frac{48}{52} \cdot \frac{4}{51} \frac{3}{50} \cdot \frac{2}{49} \cdot \frac{1}{48} \\
&= 5 \cdot \frac{4}{52} \cdot \frac{3}{51} \frac{2}{50} \cdot \frac{1}{49}
\end{aligned}$$

2.2.2 Law of Total Probability and Bayes Rule

A set of events $\{A_i\}$ is called a **partition** of the sample space if

$$A_i \cap A_j = \emptyset \text{ if } i \neq j$$
$$\bigcup_{i=1}^{n} A_i = S$$

2.2.2.1 Example

a student is selected at random from all the undergraduate students at the Colegio

A$_1$ = "Student is female", A$_2$ = "Student is male"

or maybe

A$_1$ = "Student is freshman", .., A$_4$ = "Student is senior"

Let B be any event, then the law of total probability says

$$P(B) = \sum_{i=1}^{n} P(B|A_i)$$

2.2.2.2 Example

A company has 452 employees, 210 men and 242 women. 15% of the men and 10% of the women have a managerial position. What is the probability that a randomly selected person in this company does **not** have a managerial position?

Let A$_1$ = "person is female", A2 = "person is male"

Let B = "person has a managerial position"

Then

$$P(A_1) = \frac{242}{452}, \quad P(A_2) = \frac{210}{452}$$

$$P(B|A_1) = 0.1, \quad P(B|A_2) = 0.15$$

$$P(B) = P(B|A_1) \cdot P(A_1) + P(B|A_2) \cdot P(A_2) =$$

$$0.1 \cdot \frac{242}{452} + 0.15 \cdot \frac{210}{452} = 0.123$$

$$P(\overline{B}) = 1 - P(B) = 1 - 0.123 = 0.877$$

This is also part of Bayes' Rule:

$$P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{i=1}^{n} P(B|A_i)}$$

Notice that the denominator is just the law of total probability.

2.2.2.3 Example

In the company above a person is randomly selected, and that person is in a managerial position. What is the probability the person is female?

$$P(A_1|B) = \frac{P(B|A_1)\cdot P(A_1)}{P(B)} = \frac{0.1\cdot\frac{242}{452}}{0.123} = 0.434$$

####**Example**

A company has received three shipments, one each from three different suppliers. The shipment from company 1 contained 37 parts, the one from company 2 had 25 parts and the one from company 3 had 20 parts. An employee randomly selected one part from each shipment and tested it. It turned out one of them was bad. Unfortunately he did not pay attention which part came from which company. From previous experience we know that a part made by company 1 is faulty with probability 0.043. For company 2 the probability is 0.033 and for company 3 it is 0.027. What is the probability that the bad part came from company 2?

Let A_i = "part was made by company i"

B = "part is bad"

$$P(A_2|B) = \frac{P(B|A_2)\cdot P(A_2)}{\sum_{i=1}^{3} P(B|A_i)\cdot P(A_i)}$$
$$= \frac{0.033\cdot 25/82}{0.043\cdot 37/82+0.033\cdot 25/82+0.027\cdot 20/82} = 0.279$$

Bayes' Rule plays a very important role in Statistics and in Science in general. It provides a natural method for updating you knowledge based on data.

2.2.3 Independence

Sometimes knowing that one event occured does not effect the probability of another event. For example if you throw a red and a blue die, knowing that the red die shows a "6" will not change the probability that the blue die shows a "2".

Formally we have

$$P(A|B) = P(A)$$

or using the multiplication rule we get the better formula for two independent events

$$P(A\cap B) = P(A)P(B)$$

####**Example**

Say you flip a fair coin 5 times. What is the probability of 5 "heads"?

Let A_i = i^{th} flip is heads

Now it is reasonable to assume that the A_i's are independent and so

$$P(A_1 \cap A_2 \cap A_3 \cap A_4 \cap A_5) =$$

$$P(A_1) \cdot P(A_1) \cdot P(A_2) \cdot P(A_3) \cdot P(A_4) \cdot P(A_5)$$

$$= \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = 2^{-5} = 0.03125$$

2.3 Random Variables

A **random variable** (r.v.) X is set-valued function from the sample space into the real numbers.

2.3.0.1 Example 1

We roll a fair die, X is the number shown on the die

2.3.0.2 Example 2

We roll a fair die, X is 1 if the die shows a six, 0 otherwise.

2.3.0.3 Example 3

We roll a a fair die until the the first "6", X is the number of rolls needed.

2.3.0.4 Example 4

We randomly pick a time between 10am and 12 am, X is the minutes that have passed since 10am.

There are two basic types of r.v.'s:

- If X takes countably many values, X is called a **discrete** r.v.
- If X takes uncountably many values, X is called a **continuous** r.v.

There are also mixtures of these two.

Aboves examples 1, 2 and 3 above X is discrete, example 4 X is continuous.

There are some technical difficulties when defining a r.v. on a sample space like $\mathbb{R}$, it turns out to be impossible to define it for every subset of $\mathbb{R}$ without getting logical contradictions. The solution is to define a **σ-algebra** on the sample space and then define X only on that σ-algebra. We will ignore these technical difficulties.

Almost everything to do with r.v.'s has to be done twice, once for discrete and once for continuous r.v.'s. This separation is only artificial, it goes away once a more general definition of "integral" is used (Rieman-Stilties or Lebesgue)

2.3.1 (Commulative) Distribution Function

The distribution function of a r.v. X is defined by

$F(x) = P(X \leq x) \; \forall x \in \mathbb{R}$

2.3.1.1 Example 1

say x=2.2, then

$F(2.2) = P(X \leq 2.2) = P(1, 2) = 2/6 = 1/3$

2.3.1.2 Example 4

say x=67.5, then

$F(67.5) = P(X \leq 67.5) =$ P(we chose a moment between 10am and 11h7.5min am) $= 67.5/120 = 0.5625$

Some features of cdf's:

1. cdf's are standard functions on $\mathbb{R}$
2. $0 \leq F(x) \leq 1$
3. cdf's are non-decreasing
4. cdf's are right-continuous
5.
$$\begin{aligned} &F(x) \to 0 \text{ as } x \to -\infty \\ &F(x) \to 1 \text{ as } x \to \infty \end{aligned}$$

2.3.1.3 Example :

find the cdf F of the random variable X in example 3 above.

Solution: note $X \in \{1, 2, 3, ...\}$

let A_i be the event "a six on the i^{th} roll", i=1,2,3, ... Then

$$P(X = k) = P(\{A_1^c, A_2^c, .., A_{k-1}^c, A_k\}) =$$

$$P(\{A_1^c\}) \cdot P(\{A_2^c\}) \cdot ... \cdot P(\{A_{k-1}^c\}) \cdot P(\{A_k\}) = \left(\frac{5}{6}\right)^{k-1} \cdot \frac{1}{6}$$

and

$$P(X \le k) = \sum_{i=1}^{k} P(X = \text{i}) = \sum_{i=1}^{k} \left(\frac{5}{6}\right)^{i-1} \frac{1}{6} =$$

$$\frac{1}{6} \sum_{j=0}^{k-1} \left(\frac{5}{6}\right)^{j} = \frac{1}{6} \frac{1-\left(\frac{5}{6}\right)^{(k-1)+1}}{1-\frac{5}{6}} = 1 - \left(\frac{5}{6}\right)^{k}$$

so for $k \le x < k + 1$ we have $F(x) = 1 - (5/6)^k$

2.3.2 Probability Density Function (pdf)

The probability density function of a discrete r.v. X is defined by $f(x) = P(X = x)$

Note:

$f(x) = P(X = x) = P(X \le x) - P(X \le x - 1) = F(x) - F(x - 1)$

####**Example**

the pdf of X in example 3 is given by

$f(x) = 1/6 * (5/6)^{x-1}$ if $x \in \{1, 2, ..\}$, 0 otherwise.

Note that it follows from the definition and the axioms that for any density f we have

$$f(x) \ge 0$$
$$\sum_x f(x) = 1$$

f is the density of a continuous random variable with cdf F if

$F(x) = \int_{-\infty}^{x} f(t)dt$

Again it follows from the definition and the axioms that for any pdf f we have

$$f(x) \geq 0$$
$$\int_{\infty}^{\infty} f(x)dx = 1$$

####**Example**

Show that $f(x) = \lambda \exp(-\lambda x)$ if x>0, 0 otherwise defines a pdf, where $\lambda > 0$.

clearly $f(x) \geq 0$ for all x.

$$\int_{\infty}^{\infty} f(x)dx =$$
$$\int_{0}^{\infty} \lambda \exp(-\lambda t)dt =$$
$$-\exp(-\lambda t)|_0^\infty = 0 - (-1) = 1$$

This r.v. X is called an *exponential* r.v. with rate λ.

2.3.3 Random Vectors

A random vector is a multi-dimensional random variable.

####**Example**

we roll a fair die twice. Let X be the sum of the rolls and let Y be the absolute difference between the two roles. Then (X,Y) is a 2-dimensional random vector. The joint density of (X,Y) is given by:

	0	1	2	3	4	5
2	1	0	0	0	0	0
3	0	2	0	0	0	0
4	1	0	2	0	0	0
5	0	2	0	2	0	0
6	1	0	2	0	2	0
7	0	2	0	2	0	2
8	1	0	2	0	2	0
9	0	2	0	2	0	0
10	1	0	2	0	0	0
11	0	2	0	0	0	0
12	1	0	0	0	0	0

where every number is divided by 36.

All definitions are straightforward extensions of the one-dimensional case.

####**Example**

for a discrete random vector we have the density $f(x, y) = P(X = x, Y = y)$.

Say above

f(4,0) =
P(X=4, Y=0) =
P({(2,2)}) = 1/36

or

f(7,1) =
P(X=7,Y=1) =
P({(3,4),(4,3)}) = 1/18

####**Example**

Say $f(x, y) = cxy, 0 \le x < y \le 1$ is a pdf. Find c.

$$\iint_{R^2} f(x,y)d(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)dydx = \int_0^1\int_x^1 xydydx =$$

$$\int_0^1 x\left(\int_x^1 ydy\right)dx = \int_0^1 x(\tfrac{1}{2}y^2|_x^1 dx = \int_0^1 x\tfrac{1}{2}(1-x^2)dx = \tfrac{1}{2}\cdot\int_0^1 x(1-x^2)dx =$$

$$\tfrac{1}{2}\cdot\int_0^1 x - x^3 dx = \tfrac{1}{2}\cdot(\tfrac{1}{2}x^2 - \tfrac{1}{4}x^4|_0^1 = \tfrac{1}{2}(\tfrac{1}{2} - \tfrac{1}{4} - 0) = \tfrac{1}{2}\tfrac{1}{4} = \tfrac{1}{8}$$

so c=8.

Say (X,Y) is a discrete (continuous) r.v. with joint density (pdf) f. Then the **marginal** density (pdf) f_X is given by

$$f_X(x) = \sum_y f(x,y) \text{ if (X,Y) are discrete}$$
$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)dy \text{ if (X,Y) are continuous}$$

####**Example** For the discrete example above we find

$f_X(2) = f(2, 0) + f(2, 1) + .. + f(2, 5) = 1/36$

or

$f_Y(3) = 6/36$

####**Example**

Say $f(x, y) = 8xy, 0 \le x < y \le 1$, find $f_Y(y)$

$$f_Y(y) = \int_{-\infty}^{\infty} f(x,y)dx = \int_0^y 8xydx = 4yx^2|_0^y = 4y^3, 0 \le y \le 1$$

Note that f_Y is s proper pdf: $f_Y(y) \ge 0$ and

$$\int_{-\infty}^{\infty} f_Y(y)dy = \int_0^1 4y^3 dy = y^4|_0^1 = 1$$

2.3.4 Conditional R.V.'s

let (X,Y) be a discrete r.v. with joint density f(x,y) and marginals density f_X and f_Y. For any x such that $f_X(x) > 0$ the conditional density $f_{Y|X=x}(y|x)$ is defined by

$$f_{Y|X=x}(y|x) = \frac{f(x,y)}{f_Y(y)}$$

####**Example**

find $f_{X|Y=5}(7|5)$ and $f_{Y|X=3}(7|3)$

$$f_{X|Y=5}(7|5) = \frac{f(7,5)}{f_Y(5)} = \frac{2/36}{2/36} = 1$$

$$f_{Y|X=7}(3|7) = \frac{f(7,3)}{f_X(7)} = \frac{2/36}{6/36} = 1/3$$

For continous r.v. everything works the same:

####**Example** Find $f_{X|Y=y}(x|y)$

$$f_{X|Y=y}(x|y) =$$
$$\frac{f(x,y)}{f_Y(y)} =$$
$$\frac{8xy}{4y^3} = \frac{2x}{y^2}$$

for $0 \leq x \leq y$.

Here y is a fixed number!

Again, note that a conditional pdf is a proper pdf:

$$\int_{-\infty}^{\infty} f_{X|Y=y}(x|y)dx = \int_0^y 2x/y^2 dx = x^2/y^2|_0^y = 1$$

Note that a conditional density (pdf) requires a specification for a value of the random variable on which we condition, something like $f_{X|Y=y}$. An expression like $f_{X|Y}$ is not defined!

2.3.5 Independence

Two r.v. X and Y are said to be independent iff

$f_{X,Y}(x,y) = f_X(x)f_Y(y)$

####**Example**

in the example above we found $f_{X,Y}(7,1) = 1/18$ but $f_X(7)f_Y(1) = 1/6 * 10/36 = 5/108$, so X and Y are not independent

Mostly the concept of independence is used in reverse: we assume X and Y are independent (based on good reason!) and then make use of the formula:

Say we use the computer to generate 10 independent exponential r.v's with rate λ. What is the probability density function of this random vector?

We have $f_{X_i}(x_i) = \lambda \exp(-\lambda x_i)$ for i=1,2,..,10 so

$$f_{(X_1,..,X_{10})}(x_1,..,x_{10}) =$$
$$\prod_{i=1}^{10} f_{X_i}(x_i) =$$
$$\prod_{i=1}^{10} \lambda \exp(-\lambda x_i) =$$
$$\lambda^{10} \exp(-\lambda \sum_{i=1}^{10} x_i) =$$

Notation: we will use the notation $X \perp Y$ if X and Y are independent.

2.4 Expectation

2.4.1 Expectations of Random Variables

The *expectation* (or expected value) of a random variable g(X) is defined by

$$\sum_x g(x)f(x) \text{ if X discrete}$$
$$\int_{-\infty}^{\infty} g(x)f(x)dx \text{ if X continuous}$$

We use the notation Eg(X)

####**Example** we roll fair die until the first time we get a six. What is the expected number of rolls?

We saw that f(x) = 1/6*(5/6)^x-1 if

Here we just have g(x)=x, so

$$EX = \sum_{i=1}^{\infty} g(x_i)f(x_i) = \sum_{x=1}^{\infty} xf(x) = \sum_{x=1}^{\infty} x\frac{1}{6}\left(\frac{5}{6}\right)^{x-1}$$

How do we compute this sum? Here is a “standard” trick:

$$\sum_{x=1}^{\infty} x\tau^{x-1} = \sum_{x=1}^{\infty} \frac{d}{dt}(t^x)|_{t=\tau} =$$
$$\frac{d}{dt}(\sum_{x=1}^{\infty} t^x)|_{t=\tau} = \frac{d}{dt}(\sum_{x=0}^{\infty} t^x - 1)|_{t=\tau} =$$
$$\frac{d}{dt}(\frac{1}{1-t})|_{t=\tau} = \frac{1}{(1-t)^2}|_{t=\tau} = \frac{1}{(1-\tau)^2}$$

and so we find

$$EX = \frac{1}{6}\cdot\frac{1}{(1-\frac{5}{6})^2} = \frac{1}{6}\cdot\frac{1}{(\frac{1}{6})^2} = \frac{1}{\frac{1}{6}} = 6$$

####**Example**

X is said to have a *uniform* [A,B] distribution if f(x)=1/(B-A) for A<x<B, 0 otherwise.

Find EX^k (this is called the k^th moment of X).

$$EX^k = \int_{-\infty}^{\infty} x^k \cdot f(x)dx = \int_A^B x^k \cdot \frac{1}{B-A}dx = \frac{1}{B-A}\cdot(\frac{1}{k+1}x^{k+1}|_A^B = \frac{B^{k+1}-A^{k+1}}{(k+1)(B-A)}$$

---

some special expectations are the **mean** of X defined by $\mu = EX$ and the **variance** defined by $\sigma^2 = V(X) = E(X-\mu)^2$. Related to the variance is the **standard deviation** σ, the square root of the variance.

Here are some formulas for expectations:

$$E(aX+b) = aEX+b$$
$$E(X+Y) = EX+EY$$
$$V(aX+b) = a^2V(X)$$
$$V(X) = EX^2-\mu^2$$

the last one is a useful formula for finding the variance and/or the standard deviation.

####**Example** find the mean and the standard deviation of a uniform [A,B] r.v.

$$\mu = EX = \frac{B^{1+1}-A^{1+1}}{(1+1)(B-A)} = \frac{B^2-A^2}{2(B-A)} = \frac{(B+A)(B-A)}{2(B-A)} = \frac{A+B}{2}$$

$$EX^2 = \frac{B^{2+1}-A^{2+1}}{(2+1)(B-A)} = \frac{B^3-A^3}{3(B-A)} = \frac{\left(B^2+BA+A^2\right)(B-A)}{3(B-A)} = \frac{A^2+AB+B^2}{3}$$

$$\sigma^2 = EX^2 - \mu^2 = \frac{A^2+AB+B^2}{3} - \left(\frac{A+B}{2}\right)^2 = \frac{4A^2+4AB+4B^2-3A^2-6AB-3B^2}{12} =$$

$$\frac{A^2-2AB+B^2}{12} = \frac{(A-B)^2}{12}$$

and so $\sigma = (B-A)/\sqrt{12}$

####**Example** Find the mean and the standard deviaiton of an exponential rv with rate λ.

$$\mu = EX = \int_{-\infty}^{\infty} xf(x)dx = \int_0^{\infty} x\lambda e^{-\lambda x}dx =$$

$$(x \cdot (-e^{-\lambda x})|_0^{\infty} - \int_0^{\infty} -e^{-\lambda x}dx = -\frac{1}{\lambda}e^{-\lambda x}|_0^{\infty} = \frac{1}{\lambda}$$

$$EX^2 = \int_{-\infty}^{\infty} x^2 f(x)dx = \int_0^{\infty} x^2\lambda e^{-\lambda x}dx =$$

$$(x^2 \cdot (-e^{-\lambda x})|_0^{\infty} - \int_0^{\infty} -2xe^{-\lambda x}dx = 2\int_0^{\infty} xe^{-\lambda x}dx =$$

$$\frac{2}{\lambda}\int_0^{\infty} x\lambda e^{-\lambda x}dx = \frac{2}{\lambda}\mu = \frac{2}{\lambda^2}$$

and so

$$\sigma = \sqrt{V(X)} = \sqrt{EX^2 - \mu^2} = \sqrt{\frac{2}{\lambda^2} - \frac{1}{\lambda^2}} = \frac{1}{\lambda}$$

One way to "link" probabilities and expectations is via the indicator function I_A defined as

$$I_A(x) = \begin{cases} 1 \text{ if } x \in A \\ 0 \text{ if } x \notin A \end{cases}$$

because with this we have for a continuous r.v. X with density f:

$$EI_A = \int_{-\infty}^{\infty} I_A(x)f(x)dx = \int_A f(x)dx = P(A)$$

2.4.2 Expectations of Random Vectors

The definition of expectation easily generalizes to random vectors:

####**Example**

Let (X,Y) be a discrete random vector with
$f(x,y) = (1/2)^{x+y}, x \geq 1, y \geq 1$
Find $E[XY^2]$

$$E[XY^2] = \sum_{x=1}^{\infty}\sum_{y=1}^{\infty} xy^2\left(\tfrac{1}{2}\right)^{(x+y)} =$$

$$\sum_{x=1}^{\infty} x\left(\tfrac{1}{2}\right)^{x} \cdot \sum_{y=1}^{\infty} y^2\left(\tfrac{1}{2}\right)^{y} =$$

now

$$\sum_{x=1}^{\infty} x\left(\tfrac{1}{2}\right)^{x} = \tfrac{1}{2}\sum_{x=1}^{\infty} x\left(\tfrac{1}{2}\right)^{x-1} = \tfrac{1}{2}\sum_{x=1}^{\infty} \tfrac{d}{dt}t^x|_{t=1/2} =$$

$$\tfrac{1}{2}\tfrac{d}{dt}\sum_{x=1}^{\infty} t^x|_{t=1/2} = \tfrac{1}{2}\tfrac{d}{dt}\left(\sum_{x=0}^{\infty} t^x \text{ -1}\right)|_{t=1/2} = \tfrac{1}{2}\tfrac{d}{dt}\left(\tfrac{1}{1-t}\text{ -1}\right)|_{t=1/2} =$$

$$\tfrac{1}{2}\tfrac{1}{(1-t)^2}|_{t=1/2} = \tfrac{1}{2}\tfrac{1}{(1-1/2)^2} = 2$$

Also

$$\sum_{x=1}^{\infty} x(x-1)\left(\tfrac{1}{2}\right)^{x} = \left(\tfrac{1}{2}\right)^2\sum_{x=2}^{\infty} x(x-1)\left(\tfrac{1}{2}\right)^{x-2} = \tfrac{1}{4}\sum_{x=2}^{\infty}\tfrac{d^2}{dt^2}t^x|_{t=1/2} =$$

$$\tfrac{1}{4}\tfrac{d^2}{dt^2}\sum_{x=2}^{\infty} t^x|_{t=1/2} = \tfrac{1}{4}\tfrac{d^2}{dt^2}\left(\sum_{x=0}^{\infty} t^x \text{ -1-t}\right)|_{t=1/2} = \tfrac{1}{4}\tfrac{d^2}{dt^2}\cdot\left(\tfrac{1}{1-t}\text{ -1-t}\right)|_{t=1/2} =$$

$$\tfrac{1}{4}\tfrac{2}{(1-t)^3}|_{t=1/2} = \tfrac{1}{4}\tfrac{2}{(1-1/2)^3} = 4$$

but

$$\sum_{x=1}^{\infty} x(x-1)\left(\tfrac{1}{2}\right)^{x} = \sum_{x=1}^{\infty} x^2\left(\tfrac{1}{2}\right)^{x} - \sum_{x=1}^{\infty} x\left(\tfrac{1}{2}\right)^{x}$$

and so

$$\sum_{x=1}^{\infty} x^2\left(\tfrac{1}{2}\right)^{x} = \sum_{x=1}^{\infty} x(x-1)\left(\tfrac{1}{2}\right)^{x} + \sum_{x=1}^{\infty} x\left(\tfrac{1}{2}\right)^{x} = 4+2 = 6$$

finally

$$E[XY^2] = 2\cdot 6 = 12$$

2.4.3 Covariance and Correlation

The covariance of two r.v. X and Y is defined by $cov(X,Y) = E[(X-\mu_X)(Y-\mu_Y)]$

The correlation of X and Y is defined by

$cor(X, Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$

Note cov(X,X) = V(X)

As with the variance we have a simpler formula for actual calculations:

$cov(X, Y) = E(XY) - (EX)(EY)$

####**Example** take the example of the sum and absolute value of the difference of two rolls of a die. What is the covariance of X and Y?

So we have

$\mu_X = EX = 2 * 1/36 + 3 * 2/36 + ... + 12 * 1/36 = 7.0$
$\mu_Y = EY = 0 * 6/36 + 1 * 12/36 + ... + 5 * 2/36 = 70/36$
$EXY = 0 * 2 * 1/36 + 1 * 2 * 0/36 + .2 * 2 * 0/36.. + 5 * 12 * 0/36 = 490/36$

and so

$cov(X, Y) = EXY - EXEY = 490/36 - 7.0 * 70/36 = 0$

Note that we previously saw that X and Y are **not** independent, so we here have an example that a covariance of 0 does **not** imply independence! It does work the other way around, though:

Theorem: If X and Y are independent, then cov(X,Y) = 0 (= cor(X,Y))

proof (in the case of X and Y continuous):

$$EXY = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xy f_{(X,Y)}(x,y)\,dx\,dy = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xy f_X(x) f_Y(y)\,dx\,dy =$$
$$\left(\int_{-\infty}^{\infty} x f_X(x)\,dx\right) \cdot \left(\int_{-\infty}^{\infty} y f_Y(y)\,dy\right) = EX \cdot EY$$

and so cov(X,Y) = EXY-EXEY = EXEY - EXEY = 0

####**Example** Consider again the example from before: we have continuous rv's X and Y with joint density

$f(x, y) = 8xy, 0 \le x < y \le 1$

Find the covariance and the correlation of X and Y.

We have seen before that $f_Y(y) = 4y^3, 0 < y < 1$, so

$E[Y] = \int_{-\infty}^{\infty} y f_Y(y) dy = \int_0^1 y 4y^3 dy = 4/5 y^5|_0^1 = 4/5$

Now

$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)dy = \int_x^1 8xydy = 4xy^2|_x^1 = 4(x-x^3), 0 < x < 1$$

so

$$E[X] = \int_0^1 x4(x-x^3)dx = \tfrac{4}{3}x^3 - \tfrac{4}{5}x^5|_0^1 = \tfrac{4}{3} - \tfrac{4}{5} = \tfrac{8}{15}$$

and

$$E[XY] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} xyf(x,y)dydx = \int_0^1\int_x^1 8x^2y^2dydx = \int_0^1 \tfrac{8}{3}x^2y^3|_x^1 dx =$$
$$\int_0^1 \tfrac{8}{3}(x^2-x^5)dx = \tfrac{8}{9}x^3 - \tfrac{4}{9}x^5|_0^1 = \tfrac{4}{9}$$

and so cov(X,Y)=4/9-8/15 $\cdot$ 4/5 = 12/675

Also

$$EX^2 = \int_0^1 x^2 4(x-x^3)dx = x^4 - \tfrac{2}{3}x^6|_0^1 = 1 - \tfrac{2}{3} = \tfrac{1}{3}$$

so $\sigma_X^2 = EX^2 - \mu_x^2 = \frac{1}{3} - \left(\frac{8}{15}\right)^2 = \frac{11}{225}$

$$EY^2 = \int_0^1 y^2 4y^3 dy = \tfrac{2}{3}y^6|_0^1 = \tfrac{2}{3}$$

so $\sigma_Y^2 = EY^2 - \mu_Y^2 = \frac{2}{3} - \left(\frac{4}{5}\right)^2 = \frac{2}{75}$

$$cor(X,Y) = \frac{cov(X,Y)}{\sigma_X\sigma_Y} = \frac{\frac{12}{675}}{\sqrt{\frac{11}{225}\frac{2}{75}}} = 0.492$$

We saw above that E(X+Y) = EX + EY. How about V(X+Y)?

$$V(X+Y) = E(X+Y)^2 - (E(X+Y))^2 =$$
$$E\left[X^2 + 2XY + Y^2\right] - (EX+EY)^2 =$$
$$EX^2 + 2EXY + EY^2 - \left((EX)^2 + 2EXEY + (EY)^2\right) =$$
$$\left(EX^2 - (EX)^2\right) + \left(EY^2 - (EY)^2\right) + 2 \cdot (EXY - EXEY) =$$
$$V(X) + V(Y) + 2Cov(X,Y)$$

and if $X \perp Y$ we have V(X+Y) = VX + VY

2.4.4 Conditional Expectation and Variance

Say X|Y=y is a conditional r.v. with density (pdf) f. Then the conditional expectation of X|Y=y is defined by

$$E[h(X)|Y=y] = \sum_x h(x) f_{X|Y=y}(x,y) \quad \text{if X is discrete}$$
$$E[h(X)|Y=y] = \int_{-\infty}^{\infty} h(x) f_{X|Y=y}(x,y)dx \quad \text{if X is continuous}$$

Let E[X|Y] denote the function of the random variable Y whose value at Y=y is given by E[X|Y=y]. Note then Z=E[X|Y] is itself a random variable.

####**Example** An urn contains 2 white and 3 black balls. We pick two balls from the urn. Let X be denote the number of white balls chosen. An additional ball is drawn from the remaining three. Let Y equal 1 if the ball is white and 0 otherwise.

For example

$f(0,0) = P(X=0, Y=0) = 3/5 * 2/4 * 1/3 = 1/10$.

The complete density is given by:

	x=0	x=1	x=2
y=0	0.1	0.4	0.1
y=1	0.2	0.2	0.0

The marginals are given by

x	P(X=x)
x=0	0.3
x=1	0.6
x=2	0.1

y	P(Y=y)
y=0	0.6
y=1	0.4

The conditional distribution of X|Y=0 is

x	P(X=x|Y=0)
0	1/6
1	2/3
2	1/6

and so $E[X|Y=0] = 0*1/6 + 1*2/3 + 2*1/6 = 1.0$.

The conditional distribution of X|Y=1 is

x	P(X=x|Y=1)
0	1/2
1	1/2
2	0

and so $E[X|Y=1] = 0*1/2 + 1*1/2 + 2*0 = 1/2$.

Finally the conditional r.v. Z = E[X|Y] has density

z	P(Z=z)
1	3/5
1/2	2/5

with this we can find $E[Z] = E[E[X|Y]] = 1*3/5 + 1/2*2/5 = 4/5$.

How about using simulation to do these calculations? - program **urn1**

```
urn1 <- function (n = 2, m = 3, draws = 2, B = 10000) {
   u <- c(rep("w", n), rep("b", m))
   x <- rep(0, B)
   y <- x
   for (i in 1:B) {
        z <- sample(u, draws + 1)
        y[i] <- ifelse(z[draws + 1] == "w", 1, 0)
        for (j in 1:draws)
          x[i] <- x[i] + ifelse(z[j] == "w", 1, 0)
    }
    print("Joint pmf:")
    print(round(table(y, x)/B, 3))
    print("pmf of X:")
    print(round(table(x)/B, 3))
    print("pmf of Y:")
    print(round(table(y)/B, 3))
    print("pmf of X|Y=0:")
    x0 <- table(x[y == 0])/length(y[y == 0])
    print(round(x0, 3))
    print("E[X|Y=0]:")
    print(sum(c(0:draws) * x0))
    print("pmf of X|Y=1:")
    x1 <- table(x[y == 1])/length(y[y == 1])
    print(round(x1, 3))
    print("E[X|Y=1]:")
    print(sum(c(0:1) * x1))

 }
urn1()
```

```
## [1] "Joint pmf:"
##    x
## y       0     1     2
##   0 0.104 0.395 0.104
##   1 0.201 0.196 0.000
## [1] "pmf of X:"
## x
##     0     1     2
## 0.305 0.592 0.104
## [1] "pmf of Y:"
## y
##     0     1
## 0.603 0.397
## [1] "pmf of X|Y=0:"
##
##     0     1     2
## 0.173 0.655 0.172
## [1] "E[X|Y=0]:"
## [1] 0.9990046
## [1] "pmf of X|Y=1:"
##
```

```
##     0     1
## 0.505 0.495
## [1] "E[X|Y=1]:"
## [1] 0.494713
```

####**Example**

Consider again the example from before: we have continuous rv's X and Y with joint density $f(x, y) = 8xy, 0 \le x < y \le 1$. We have found $f_Y(y) = 4y^3, 0 < y < 1$, and $f_{X|Y=y}(x|y) = 2x/y^2, 0 \le x \le y$. So

$$E[X|Y = y] = \int_{-\infty}^{\infty} x f_{X|Y=y}(x|y)dx = \int_0^y x\frac{2x}{y^2}dx = \frac{2}{3y^2}x^3|_0^y = \frac{2y^3}{3y^2} = \frac{2}{3}y$$

Throughout this calculation we treated y as a constant. Now, though, we can change our point of view and consider $E[X|Y = y] = 2y/3$ as a function of y:

$g(y) = E[X|Y = y] = 2y/3$

What are the values of y? Well, they are the observations we might get from the rv. Y, so we can also write

$g(Y) = E[X|Y = Y] = 2Y/3$

but Y is a rv, then so is 2Y/3, and we see that we can define a rv Z=g(Y)=E[X|Y].

Recall that the expression $f_{X|Y}$ does not make sense. Now we see that on the other hand the expression E[X|Y] makes perfectly good sense!

There is a very useful formula for the expectation of conditional r.v.s:

$$E[E[X|Y]] = E[X]$$

$E[X] = 0 * 3/10 + 1 * 3/5 + 2 * 1/10 = 4/5.$

There is a simple explanation for this seemingly complicated formula!

Here is a corresponding formula for the variance:

$$V(X) = E[V(X|Y)] + V[E(X|Y)]$$

####**Example**

let's say we have a continuous bivariate random vector with the joint pdf $f(x, y) = c(x + 2y)$ if $0 < x < 2$ and $0 < y < 1$, 0 otherwise.

Find c:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)dx = \int_0^1\int_0^2 c(x+2y)dxdy =$$

$$c\int_0^2 \frac{1}{2}x^2 + 2yx|_0^2 dy = c\int_0^2 2+4ydy =$$

$$c(2y+2y^2|_0^1 = c(2+2) = 4c = 1 \Rightarrow c = \frac{1}{4}$$

Find the marginal distribution of X

$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)dx = \int_0^1 \frac{1}{4}\cdot(x+2y)dy =$$

$$\frac{1}{4}\cdot(xy+y^2|_0^1 = \frac{1}{4}\cdot(x+1) \quad \text{for } 0 < x < 2$$

Find the marginal distribution of Y

$$f_Y(y) = \int_{-\infty}^{\infty} f(x,y)dx = \int_0^2 \frac{1}{4}\cdot(x+2y)dx =$$

$$\frac{1}{4}\cdot(\frac{1}{2}x^2+2yx|_0^2 = \frac{1}{4}\cdot(2+4y) =$$

$$\frac{1}{2}+y \text{ for } 0 < y < 1$$

Find the conditional pdf of Y|X=x

$$f_{Y|X=x}(y|x) = \frac{f(x,y)}{f_X(x)} = \frac{\frac{1}{4}(x+2y)}{\frac{1}{4}(x+1)} =$$

$$\frac{x+2y}{x+1} \text{ for } 0 < y < 1$$

Note: this is a proper pdf for any fixed value of x

Find E[Y|X=x]

$$E[Y|X=x] = \int_{-\infty}^{\infty} y f_{Y|X=x}(y|x)dy = \int_0^1 y\frac{x+2y}{x+1}dy =$$

$$\frac{1}{x+1}(\tfrac{1}{2}xy^2 + \tfrac{2}{3}y^3|_0^1 = \frac{1}{x+1}(\tfrac{1}{2}x + \tfrac{2}{3})$$

for $0 < x < 2$

Let Z=E[Y|X]. Find E[Z]

Solution 1: $E[Z] = E[E[Y|X]] = \int_{-\infty}^{\infty} E[Y|X=x]f_X(x)dx =$

$$\int_0^2 \frac{1}{x+1}\cdot\left(\tfrac{1}{2}x + \tfrac{2}{3}\right)\tfrac{1}{4}(x+1)dx = \int_0^2\left(\tfrac{1}{2}x + \tfrac{2}{3}\right)\tfrac{1}{4}dx =$$

$$\int_0^2\left(\tfrac{1}{8}x + \tfrac{1}{6}\right)dx = (\tfrac{1}{16}x^2 + \tfrac{1}{6}x|_0^2 = \tfrac{1}{16}4 + \tfrac{1}{6}2 = \tfrac{7}{12}$$

Solution 2: $E[Z] = E[E[Y|X]] = E[Y] = \int_{-\infty}^{\infty} yf_Y(y)dy =$

$$\int_0^1 y\left(\tfrac{1}{2} + y\right)dy = (\tfrac{1}{4}y^2 + \tfrac{1}{3}y^3|_0^1 = \tfrac{1}{4} + \tfrac{1}{3} = \tfrac{7}{12}$$

2.5 Inequalities and Limit Theorems

2.5.1 Two very useful inequalities

Markov's Inequality

If X takes on only nonnegative values, then for any a>0

$$P(X \geq a) \leq \frac{EX}{a}$$

proof:

$$P(X \geq a) = \int_a^\infty f(x)dx \leq \int_a^\infty \frac{x}{a} f(x)dx =$$
$$\frac{1}{a}\int_a^\infty xf(x)dx \leq \frac{1}{a}\int_0^\infty xf(x)dx =$$
$$\frac{1}{a}\int_{-\infty}^\infty xf(x)dx = \frac{1}{a}E[X]$$

Chebyshev's Inequality:

If X is a r.v. with mean mu and variance σ^2, then for any k>0:

$$P(|X-\mu| \geq k\sigma) \leq 1/k^2$$

proof:

$$\begin{aligned} &P(|X-\mu| \geq k\sigma) = \\ &P((X-\mu)^2 \geq k^2\sigma^2) \leq \\ &\frac{E(X-\mu)^2}{k^2\sigma^2} = \frac{\sigma^2}{k^2\sigma^2} = 1/k^2 \end{aligned}$$

####**Example** Consider the uniform random variable with f(x) = 1 if $0 < x < 1$, 0 otherwise. We already know that $\mu = 0.5$ and $\sigma = 1/\sqrt{12} = 0.2887$. Now Chebyshev says

$P(|X-0.5| > k0.2887) \leq 1/k^2$

For example

$P(|X-0.5| > 0.2887) \leq 1$ (rather boring!)

or

$P(|X-0.5| > 3 \times 0.2887) \leq 1/9$

actually $P(|X-0.5| > 0.866) = 0$, so this is not a very good upper bound.

2.5.2 (Weak) Law of Large Numbers

Let $X_1, X_2, ...$ be a sequence of independent and identically distributed (iid) r.v.'s having mean μ. Then for all $\epsilon > 0$

$$P(|\frac{1}{n}\sum X_i - \mu| > \epsilon) \to 0$$

proof (assuming in addition that $V(X_i) = \sigma^2 < \infty$

$$
\begin{aligned}
&E[\frac{1}{n}\sum X_i] = \frac{1}{n}\sum E[X_i] = \mu \\
&V[\frac{1}{n}\sum X_i] = \frac{1}{n^2}\sum V[X_i] = \frac{\sigma^2}{n} \\
&P(|\frac{1}{n}\sum X_i - \mu| > \epsilon) = \\
&P(|\frac{1}{n}\sum X_i - \mu| > \frac{\epsilon}{\sigma/\sqrt{n}}\sigma/\sqrt{n}) \le \\
&1/(\frac{\epsilon}{\sigma/\sqrt{n}}) = \frac{\sigma}{\epsilon\sqrt{n}} \to 0
\end{aligned}
$$

This theorem forms the bases of (almost) all simulation studies: say we want to find a parameter θ of a population. We can generate data from a random variable X with pdf (density) $f(x|\theta)$ such that $Eh(X) = \theta$. Then by the law of large numbers

$$\frac{1}{n}\sum h(X_i) \to \theta$$

####**Example**

in a game a player rolls 5 fair dice. He then moves his game piece along k fields on a board, where k is the smallest number on the dice + largest number on the dice. For example if his dice show 2, 2, 3, 5, 5 he moves 2+5 = 7 fields. What is the mean number of fields θ a player will move?

To do this analytically would be quite an excercise. To do it via simulation is easy:

Let X be an independent random vector of length 5, with $X[j] \in 1,..,6$ and $P(X[j] = k) = 1/6$. Let $h(x) = min(x) + max(x)$, then $Eh(X) = \theta$.

Let $X_1, X_2, ..$ be iid copies of X, then by the law of large numbers

```
B <- 1e5
z <- rep(0, B)
for (i in 1:B) {
  x <- sample(1:6, size = 5, replace = TRUE)
  z[i] <- min(x)+max(x)
}
mean(z)
```

```
## [1] 7.00259
```

####**Example** A company has a system (website, computers, telephone bank, machine, ...) that is mission-critical, and so they have two identical systems with the second one taking over automatically when the first one fails. From experience they know that each systems failure time has an exponential distribution with mean 84.6 hours. Once a system is down its repair time has a U[1,5] distribution. What is the probability that both systems are down simultaneously?

Note: there is the possibility that both systems go down, system 1 gets fixed but breaks again before system 2 is fixed, so that they both are down immediately. The probability of this happening is so small though that we will ignore this.

Let T_i, i=1,2 be the failure time of system i. Let R be the repair time of system 1. Then the probability of both systems being down is

$P(R > T_2) = P(R - T_2 > 0) = E[I_{(0,\infty)})(R - T_2)]$

So we have

$h(R, T_2) = I_{(0,\infty)})(R - T_2)$

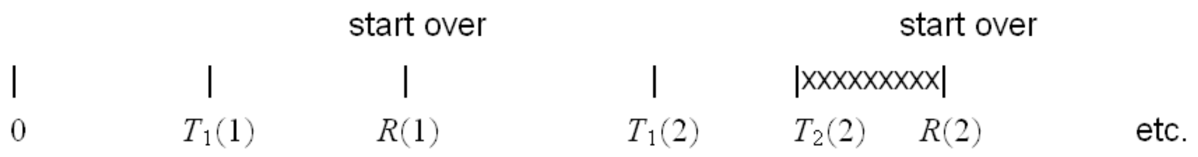
```
exsystems1 <- function(B=1000)
{
  T2 <- rexp(B, 1/84.6)
  R <- runif(B, 1, 5)
  z <- ifelse(R - T2 > 0, 1, 0)
  mean(z)
}
exsystems1()
```

```
## [1] 0.038
```

What is the mean time per year when both systems are down?

Let $T_i(k)$ be the failure times of system i in order with k=1 the first time, k=2 the second time etc.. Let R(k) be the same for the repair time. So we are looking at a sequence

$T_1(1), R(1), T_1(2), T_2(2)$ etc.



```
exsystems2 <- function (B=1000, u=5) {
  yearhours <- 24*365
  z <- rep(0, B)
  for(j in 1:B) {
    T1 <- rexp(1000, 1/84.6)
    T2 <- rexp(1000, 1/84.6)
    R <- runif(1000, 1, u)
    totaltime <- 0
    downtime <- 0
    for (i in 1:1000) {
      totaltime <- totaltime + T1[i] + R[i]
      if (T2[i] < R[i])
        downtime <- downtime + (R[i] - T2[i])
      if (totaltime > yearhours)
```

```
        break
    }
    if (totaltime < yearhours)
    print(paste("not enough time"))
    z[j] <- downtime
  }
  z
}
z <- exsystems2()
mean(z)
```

```
## [1] 6.002364
```

Say the company has the option to contract another repair man, which would lower the repair time such that then R~U[1,3]. It costs the company $8950 per hour when the system is down. How much can they pay this new repair man so it is a good idea to contract him?

We found that without the new guy we have a downtime of about 6 hours per year for a yearly cost of $53700. If we higher the new guy we have an annual downtime of about 2.5 hours for a total cost of $22400. So if we should pay him at most $31300 per year.

Say we have a contract with our main customer that specifies that our downtime can not exceed 10 hours per year, otherwise we have to pay a fine. We decide we are willing to accept a 10% chance that we exceed the time limit. Should we proceed with these conditions?

```
quantile(z, 0.9)
```

```
##      90%
## 11.09243
```

We see that there is about a 15% chance of the failure time exceeding 10 hours, so we should not proceed.

2.5.3 Central Limit Theorem

This is one of the most famous theorems in all of mathematics / statistics. Without it, Statistics as a science would not have existed until very recently:

We first need the definition of a normal (or Gaussian) r.v.:

A random variable X is said to be normally distributed with mean μ and standard deviation σ if it has density:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

If $\mu = 0$ and $\sigma = 1$ we say X has a standard normal distribution.

We use the symbol Φ for the distribution function of a standard normal r.v.

Let $X_1, X_2, ..$ be an iid sequence of r.v.'s with mean μ and standard deviation σ. Let $\bar{X} = \frac{1}{n}\sum X$. Then

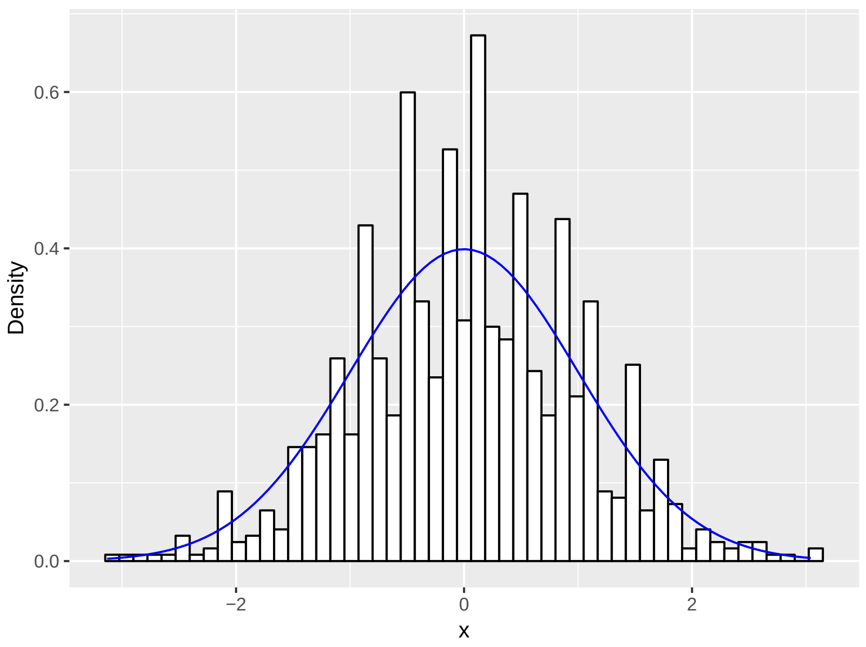
$$P(\frac{\bar{X} - \mu}{\sigma/\sqrt{n}} \le z) \to \Phi(z)$$

###**Example**

Let's do a simulation to illustrate the CLT: we will use the most basic r.v. of all, called a Bernoulli r.v. which has $P(X = 0) = 1 - p$ and $P(X = 1) = p$. (Think indicator function for the coin toss}. So we sample n Bernoulli r.v. with "success paramater p" and find their sample mean. Note that

$$E(X) = p$$
$$V(X) = p(1 - p)$$

```
cltexample1 <- function (p, n, B=1000) {
  xbar <- rep(0, n)
  for (i in 1:B) {
    xbar[i] <- mean(sample(c(0, 1), n,
                TRUE, prob=c(1-p,  p)))
    }
  df <- data.frame(x=sqrt(n)*(xbar-p)/sqrt(p*(1-p)))
  bw <- diff(range(df$x))/50
  ggplot(df, aes(x)) +
    geom_histogram(aes(y = ..density..),
      color = "black",
      fill = "white",
      binwidth = bw) +
      labs(x = "x", y = "Density") +
    stat_function(fun = dnorm, colour = "blue",
                  args=list(mean=0, sd=1))

}
cltexample1(0.5, 500)
```



2.5.4 Approximation Methods

Say we have a r.v. X with density f, a function h and we want to know V(h(X)). Of course we have the definitions but sometimes these integrals (sums) are very difficult to evaluate. In this section we discuss some methods for approximating the variance.

Recall: If a function h(x) has derivatives of order r, that is if $h^{(r)}(x)$ exists, then for any constant a the **Taylor polynomial** of order r is defined by

$$T_r(x) = \sum_0^r \frac{h^r(a)}{n!}(x-a)^n$$

One of the most famous theorems in mathematics called Taylor's theorem states that the remainder of the approximation $h(x) - T_r(x)$ goes to 0 faster than the highest order term:

Taylor's theorem

$$\lim_{x \to a} \frac{h(x) - T_r(x)}{(x-a)^r} = 0$$

There are various formulas for the remainder term, but we won't need them here.

####**Example**

say $h(x) = log(x+1)$ and we want to approximate h at x=0. Then we have

$$h(x)|_{x=0} = h(0) = 0 \qquad h'(x)|_{x=0} = \frac{1}{x+1}|_{x=0} = 1$$

$$h''(x)|_{x=0} = -\frac{1}{(x+1)^2}|_{x=0} = -1 \qquad h^{(3)}(x)|_{x=0} = \frac{2}{(x+1)^3}|_{x=0} = 2$$

$$h^{(r)}(x)|_{x=0} = (-1)^{r+1}\frac{(r-1)!}{(x+1)^r}|_{x=0} = (-1)^{r+1}(r-1)!$$

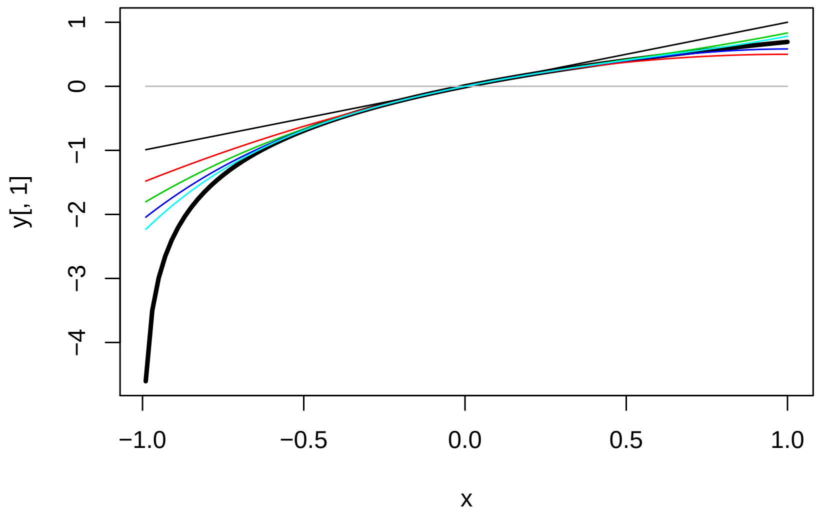
$$T_0(x) = h(0) = 0$$

$$T_1(x) = T_0(x) + h'(0) \cdot (x-0) = 0 + 1 \cdot (x-0) = x$$

$$T_2(x) = T_1(x) + \frac{h''(0)}{2}(x-0)^2 = x + \frac{-1}{2} \cdot (x-0)^2 = x - x^2/2$$

$$T_3(x) = T_2(x) + \frac{h^{(3)}(0)}{6}(x-0)^3 = x - x^2/2 + \frac{2}{6}(x-0)^3 =$$
$$x - x^2/2 + x^3/3$$

```r
taylor <- function(a=0, r=5, xrange=c(-0.99, 1)) {
  x <- seq(xrange[1], xrange[2], length = 100)
  h <- rep(0, r + 1)
  h[1] <- log(a + 1)
  for (n in 1:r) h[n + 1] <- (-1)^(n + 1)/n/(a + 1)^n
  y <- matrix(0, 100, r + 2)
  for (k in 1:100) {
    y[k, 1] <- log(x[k] + 1)
    y[k, 2] <- log(a + 1)
    for (n in 1:r)
      y[k, n+2] <- y[k, n+1] + h[n+1]*(x[k]-a)^n
  }
  plot(x, y[, 1], type = "l",
       ylim = c(min(y), max(y)), lwd = 3)
  for (n in 2:(r + 2)) lines(x, y[, n], col = n + 6)

}
taylor()
```



For our purposes we will need only first-order approximations (that is using the first derivative) but we will need a multivariate extension as follows: say $X_1, .., X_n$ are r.v. with means $\mu_1, .., \mu_n$ and define $\mathbf{X} = (X_1, .., X_n)$ and $\mu = (\mu_1, .., \mu_n)$. Suppose there is a differentiable function $h(\mathbf{x})$ for which we want an approximate estimate of the variance. Define

$$h_i'(\mu) = \frac{\partial}{\partial t_i} h(\mathbf{t})|_{t_i=\mu_i}$$

The first order Taylor expansion of h about μ is

$$h(\mathbf{t}) = h(\mu) + \sum_{i=1}^{n} h_i'(\mu)(t_i - \mu_i) + \text{Remainder}$$

Forgetting about the remainder we have

$$Eh(\mathbf{X}) \approx E\left[h(\boldsymbol{\mu}) + \sum_{i=1}^{n} h_i'(\boldsymbol{\mu})(\mathbf{X}_i - \mu_i) \right] = h(\boldsymbol{\mu}) + \sum_{i=1}^{n} h_i'(\boldsymbol{\mu})(E\mathbf{X}_i - \mu_i) = h(\boldsymbol{\mu})$$

and

$$Vh(\mathbf{X}) \approx E\left[(h(\mathbf{X}) - h(\boldsymbol{\mu}))^2\right] \approx E\left[\left(\sum_{i=1}^n h_i'(\boldsymbol{\mu})(\mathbf{X}_i - \mu_i)\right)^2\right]$$
$$= E\left[\sum_{i,j=1}^n \left(h_i'(\boldsymbol{\mu})(\mathbf{X}_i - \mu_i) \cdot h_j'(\boldsymbol{\mu})(\mathbf{X}_j - \mu_j)\right)\right] =$$
$$E\left[\sum_{i=1}^n (h_i'(\boldsymbol{\mu}))^2(\mathbf{X}_i - \mu_i)^2\right] + 2 \cdot E\left[\sum_{i<j}^n \left(h_i'(\boldsymbol{\mu})(\mathbf{X}_i - \mu_i) \cdot h_j'(\boldsymbol{\mu})(\mathbf{X}_j - \mu_j)\right)\right] =$$
$$\sum_{i=1}^n (h_i'(\boldsymbol{\mu}))^2 E(\mathbf{X}_i - \mu_i)^2 + 2 \cdot \sum_{i<j}^n \left(h_i'(\boldsymbol{\mu})h_j'(\boldsymbol{\mu})E[(\mathbf{X}_i - \mu_i)(\mathbf{X}_j - \mu_j)]\right) =$$
$$\sum_{i=1}^n (h_i'(\boldsymbol{\mu}))^2 V\mathbf{X}_i + 2 \cdot \sum_{i<j}^n h_i'(\boldsymbol{\mu})h_j'(\boldsymbol{\mu})Cov(\mathbf{X}_i, \mathbf{X}_j)$$

####**Example** say we have a sample $X_1, .., X_n$ from a Bernoulli r.v. with success parameter p. One popular measure of the probability of winning a game is the odds p/(1-p). For example when you roll a fair die the odds of getting a six are (1/6)/(1-(1/6) = 1:5.

An obvious estimator for p is $\hat{p}$, the sample mean, or here the proportion of "successes" in the n trials.

Then an obvious estimator for the odds is $\frac{\hat{p}}{1-\hat{p}}$. The question is, what is the variance of this estimator?

Using the above approximation we get the following: let $h(p) = p/(1-p)$, so $h'(p) = 1/(1-p)^2$ and

$$V\left(\frac{\hat{p}}{1-\hat{p}}\right) \approx [h'(p)]^2 V(\hat{p}) =$$
$$\left[\frac{1}{(1-p)^2}\right]^2 \frac{p(1-p)}{n} = \frac{p}{n(1-p)^3}$$

```
p <- 0.5
n <- 100
B <- 10000
phat <- rep(0, B)
for (i in 1:B)
  phat[i] <- mean(rbinom(n, 1, p))
odds <- phat/(1 - phat)
c(var(odds), p/n/(1 - p)^3)
```

```
## [1] 0.04303315 0.04000000
```

####**Example** We have a rv $X \sim U[0,1]$, and a rv $Y|X = x \sim U[0,x]$. Find an approximation of $V[Y/(1+Y)]$.

Note: this is called a hierarchical model.

We have:

1) $f_X(x) = 1$ if 0<x<1, 0 otherwise
2) $f_{Y|X=x}(y|x) = 1/x$ if 0<y<x, 0 otherwise

Now

$h(t) = \frac{t}{1+t}$, so $h'(t) = \frac{1}{(1+t)^2}$

therefor $V\left(\frac{Y}{1+Y}\right) \simeq \frac{1}{(1+EY)^4} VY$

but what are EY and VY?

$f_{Y|X=x}(y|x) = \frac{f(x,y)}{f_X(x)}$ so

$f(x,y) = f_{Y|X=x}(y|x) f_X(x) = \frac{1}{x}, \; 0 < y < x < 1$

$f_Y(y) = \int_y^1 \frac{1}{x} dx = \log(x)|_y^1 = -\log(y), \; 0 < y < 1$

$EY = \int_0^1 y(-\log(y))dy = -\frac{y^2}{2}\log(y)|_0^1 - \int_0^1 -\frac{y^2}{2}\frac{1}{y}dy =$

$\int_0^1 \frac{y}{2}dy = \frac{y^2}{4}|_0^1 = \frac{1}{4}$

$EY^2 = \int_0^1 y^2(-\log(y))dy = -\frac{y^3}{3}\log(y)|_0^1 - \int_0^1 -\frac{y^3}{3}\frac{1}{y}dy =$

$\int_0^1 \frac{y^2}{3}dy = \frac{y^3}{9}|_0^1 = \frac{1}{9}$

$VY = \frac{1}{9} - \left(\frac{1}{4}\right)^2 = \frac{7}{144}$

and so

$V\left(\frac{Y}{1+Y}\right) \simeq \frac{1}{(1+EY)^4} VY = \frac{1}{\left(1+\frac{1}{4}\right)^4}\frac{7}{144} = \frac{4^4}{5^4}\frac{7}{144} = 0.0199$

```
x <-  runif(B)
y <-  runif(B, 0, x)
c(mean(y), var(y), var(y/(y+1)))
```

```
## [1] 0.24655469 0.04704202 0.01630806
```

####**Example** let's consider the random vector with joint pdf $f(x, y) = 6x$, $0 < x < y < 1$. Say we want to find $V(X/Y)$. Then if we consider the function h(x,y) = x/y we have

$$\frac{\partial}{\partial x}h(x,y) = \frac{1}{y} \text{ and } \frac{\partial}{\partial y}h(x,y) = -\frac{x}{y^2}$$

$$Eh(X,Y) = E(X/Y) \approx \frac{\mu_X}{\mu_Y}$$

$$Vh(X,Y) = V(X/Y) \approx \left(\frac{1}{\mu_Y}\right)^2 V(X) + \left(-\frac{\mu_X}{\mu_Y^2}\right)^2 V(Y) + 2\cdot\left(\frac{1}{\mu_Y}\right)\left(-\frac{\mu_X}{\mu_Y^2}\right)Cov(X,Y) =$$

$$\frac{1}{\mu_Y^2}V(X) + \frac{\mu_X^2}{\mu_Y^4}V(Y) - 2\cdot\frac{\mu_X}{\mu_Y^3}Cov(X,Y)$$

Now we need to find $\mu_X = E[X]$, V[X], $\mu_Y = E[Y]$, V[Y] and cov(X,Y):

$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)dy = \int_x^1 6x dy = 6xy|_x^1 = 6x(1-x) \quad 0 < x < 1$$

$$f_Y(y) = \int_{-\infty}^{\infty} f(x,y)dx = \int_0^y 6x dx = 3x^2|_0^y = 3y^2 \quad 0 < y < 1$$

$$EX = \int_{-\infty}^{\infty} f_X(x)dx = \int_0^1 x6x(1-x)dx = 2x^3 - \frac{3}{2}x^4|_0^1 = \frac{1}{2}$$

$$EY = \int_{-\infty}^{\infty} f_Y(y)dy = \int_0^1 y3y^2dy = \frac{3}{4}y^4|_0^1 = \frac{3}{4}$$

$$EX^2 = \int_0^1 x^2 6x(1-x)dx = \frac{3}{2}x^4 - \frac{6}{5}x^5|_0^1 = \frac{3}{10} \text{ and so } V(X) = \frac{3}{10} - \left(\frac{1}{2}\right)^2 = \frac{1}{20}$$

$$EY^2 = \int_0^1 y^2 3y^2 dy = \frac{3}{5}y^5|_0^1 = \frac{3}{5} \text{ and so } V(Y) = \frac{3}{5} - \left(\frac{3}{4}\right)^2 = \frac{3}{80}$$

$$EXY = \int_0^1\int_0^y xy6xdxdy = \int_0^1 (2x^3y|_0^y dy = \int_0^1 2y^4 dy = \frac{2}{5}y^5|_0^1 = \frac{2}{5}$$

$$\text{and so } Cov(X,Y) = EXY - EX\cdot EY = \frac{2}{5} - \frac{1}{2}\cdot\frac{3}{4} = \frac{1}{40}$$

$$V(X/Y) \approx \frac{1}{(3/4)^2}\frac{1}{20} + \frac{(1/2)^2}{(3/4)^4}\frac{3}{80} - 2\frac{(1/2)}{(3/4)^3}\frac{1}{40} = 0.058$$

Doing this via simulation has to wait until we learn how to simulate from such a random vector!

2.6 Functions of a R.V. - Transformations

####**Example**

say $X \sim U[0,1]$ and $\lambda > 0$. What is the pdf (density?) of the random variable $Y = -\lambda\log(X)$?

Solution: we first find the cdf and then the pdf as follows:

$$F_Y(y) = P(Y \le y) = P(-\lambda \log X \le y) =$$
$$P(\log X \ge -\tfrac{y}{\lambda}) = P(X \ge e^{-\frac{y}{\lambda}}) =$$
$$1 - P(X \le e^{-\frac{y}{\lambda}}) = 1 - e^{-\frac{y}{\lambda}}$$
$$f_y(y) = \frac{d}{dy} F_Y(y) = \frac{1}{\lambda} e^{-\frac{y}{\lambda}}$$

if y>0.

For y<0 note that P(-logX<y) = 0 because 0<X<1, so logX<0, so -logX>0 always.

This is an example of a function (or transformation) of a random variable. These transformations play a major role in probability and statistics. We will see how to find their pdf's (density's) on a few examples.

###**Example** Say X is the number of roles of a fair die until the first six. We have already seen that $P(X = x) = 1/6 * (5/6)^{x-1}$, x=1,2,.. Let Y be 1 if X is even, 0 otherwise. Find the density of Y.

Note: here both X and Y are discrete.

let's do this a little more general, with p instead of 1/6. Also let q=1-p=5/6. Then

$$P(Y = 1) = P(X \in \{2,4,6,..\}) = \sum_{k=1}^{\infty} pq^{2k-1} =$$
$$p(q + q^3 + q^5 + ..) = pq(1 + q^2 + q^4 + ..) =$$
$$pq\sum_{k=0}^{\infty} q^{2k} = pq\sum_{k=0}^{\infty} (q^2)^k = pq\frac{1}{1-q^2} =$$
$$pq\frac{1}{(1-q)(1+q)} = \frac{q}{1+q}$$
$$\text{and for } p = \tfrac{1}{6} \text{ we get } P(Y = 1) = \frac{5/6}{1+5/6} = \frac{5}{11}$$

and P(Y=0) = 1 - P(Y=1) = 6/11.

####**Example** Say we have a fair coin. We flip the coin until the first "Heads". What is the probability this will happen on an even-numbered flip?

Now we have the same as above, with p=0.5, so

P(Y=1)=0.5/(1+0.5)=1/3.

Is there a loaded coin with probability of heads p so that the probability of “first heads on even-numbered flip” is 1/2?

Now P(Y=1)=q/(1+q)=1/2, so 2q=1+q or q=1 or p=0, but if p=0 we never get “heads”, so no such coin exists!

###**Example** say X is a continuous r.v with pdf $f_X(x) = 1/2\exp(-|x|)$. This is called a double exponential. Let $Y = I_{[-1,1]}(X)$. Find the density of Y.

Note: here X is continuous and Y is discrete.

$$P(Y=1) = P(-1 \le X \le 1) = \int_{-1}^{1} \frac{1}{2} e^{-|x|} dx =$$

$$\int_0^1 e^{-x} dx = -e^{-x}\Big|_0^1 = -(e^{-1}-1) = 0.632$$

$$P(Y=0) = 1 - 0.632 = 0.368$$

####**Example** again let X have pdf $f_X(x) = 1/2\exp(-|x|)$. Let $Y = X^2$. Then for y<0 we have P(Y≤y) = 0. So let y>0. Then

$$F_Y(y) = P(Y \le y) = P(X^2 \le y) = P\left(-\sqrt{y} \le X \le \sqrt{y}\right) =$$

$$\int_{-\sqrt{y}}^{\sqrt{y}} \frac{1}{2} e^{-|x|} dx = \int_0^{\sqrt{y}} e^{-x} dx = -e^{-x}\Big|_0^{\sqrt{y}} = -\left(e^{-\sqrt{y}} - 1\right) = 1 - e^{-\sqrt{y}}$$

$$\text{and so } f_Y(y) = \frac{d}{dy} F_Y(y) = \frac{1}{2\sqrt{y}} e^{-\sqrt{y}},\ y \ge 0$$

Next up some examples of functions of random vectors:

####**Example** say (X,Y) is a bivariate standard normal r.v, that is it has joint density given by

$$f(x,y) = \frac{1}{2\pi} e^{-(x^2+y^2)}$$

Let the r.v. (U,V) be defined by U = X+Y and V = X-Y. Find the joint pdf of (U,V)

To start let’s define the functions $g_1(x,y) = x+y$ and $g_2(x,y) = x-y$, so that $U = g_1(X,Y)$ and $V = g_2(X,Y)$.

For what values of u and v is $f_{(U,V)}(u,v)$ positive? Well, for any values for which the system of 2 linear equations in two unknowns u=x+y and u=x-y has a solution.

These solutions are

$$x = h_1(u,v) = (u+v)/2$$
$$y = h_2(u,v) = (u-v)/2$$

From this we find that for any (u,v) there is a unique (x,y) such that u=x+y and v=x-y. So the transformation $(x,y) \rightarrow .png)(u,v)$ is one-to-one and therefore has a Jacobian given by

$$J = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial y} \end{vmatrix} = \begin{vmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{vmatrix} = -\frac{1}{2}$$

Now from multivariable calculus we have the following:

$$f_{U,V}(u,v) = f_{X,Y}(h_1(x,y), h_2(x,y)) \cdot |J| =$$
$$\frac{1}{2\pi} \exp\left(-\frac{1}{2}\left(\left(\frac{u+v}{2}\right)^2 + \left(\frac{u-v}{2}\right)^2\right)\right) \cdot \left|-\frac{1}{2}\right| =$$
$$\frac{1}{4\pi} \exp\left(-\frac{1}{2}\left(\frac{u^2+2uv+v^2+u^2-2uv+v^2}{4}\right)\right) = \frac{1}{4\pi} \exp\left(-\frac{1}{2}\left(\frac{u^2+v^2}{2}\right)\right) =$$
$$\left(\frac{1}{\sqrt{2\pi}\sqrt{2}} e^{-\frac{1}{2}\frac{u^2}{2}}\right) \cdot \left(\frac{1}{\sqrt{2\pi}\sqrt{2}} e^{-\frac{1}{2}\frac{v^2}{2}}\right)$$

Note that the density factors into a function of u and a function of v. This is not only a necessary but also a sufficient condition for U and V to be independent.

####**Example** say X and Y are independent standard normal r.v.'s. Let Z = X + Y. Find the pdf of Z.

Note: Z = X + Y = U in the example above, so the pdf of Z is just the marginal of U and we find

$$f_Z(z) = f_U(z) = \int_{-\infty}^{\infty} f_{U,V}(z,v)dv =$$

$$\int_{-\infty}^{\infty}\left(\frac{1}{\sqrt{2\pi}\sqrt{2}}e^{-\frac{1}{2}\frac{z^2}{2}}\right)\cdot\left(\frac{1}{\sqrt{2\pi}\sqrt{2}}e^{-\frac{1}{2}\frac{v^2}{2}}\right))dv =$$

$$\left(\frac{1}{\sqrt{2\pi}\sqrt{2}}e^{-\frac{1}{2}\frac{z^2}{2}}\right)\cdot\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sqrt{2}}e^{-\frac{1}{2}\frac{v^2}{2}}dv =$$

$$\frac{1}{\sqrt{2\pi}\sqrt{2}}e^{-\frac{1}{2}\frac{z^2}{2}}$$

Say X and Y are two continuous independent r.v with pdf f's f_X and f_Y, and let Z = X+Y. If we repeat the above calculations we can show that in general the pdf of Z is given by

$$f_z(z) = \int_{-\infty}^{\infty} f_X(t)f_Y(z-t)dt$$

This is called the convolution formula.

There is a second method for deriving the convolution formula which is useful. It uses a continuous analog to the law of total probability:

In the setup from above we have

$$F_{X+Y}(z) = P(X+Y\le z) = \int_{-\infty}^{\infty} P(X+Y\le z|Y=y)f_Y(y)dy =$$
$$\int_{-\infty}^{\infty} P(X\le z-y)f_Y(y)dy = \int_{-\infty}^{\infty} F_X(z-y)f_Y(y)dy$$
$$f_{X+Y}(z) = \frac{d}{dz}\int_{-\infty}^{\infty} F_X(z-y)f_Y(y)dy = \int_{-\infty}^{\infty}\frac{d}{dz}F_X(z-y)f_Y(y)dy = \int_{-\infty}^{\infty} f_X(z-y)f_Y(y)dy$$

The tricky part of this is the interchange of the derivative and the integral. Working with densities and cdfs usually means they are ok.

####**Example** Say $X_1,..,X_n$ are iid U[0,1]. Let $M = \max\{X_1,..,X_n\}$. Find f_M.

$$F_M(x) = P(M \le x) = P(\max\{X_1,..,X_n\} \le x) =$$

$$P(X_1 \le x,..,X_n \le x) = \prod_{i=1}^{n} P(X_i \le x) = [P(X_1 \le x)]^n = x^n$$

$$f_M(x) = \frac{\partial}{\partial x} F_M(x) = \frac{\partial}{\partial x} x^n = nx^{n-1}$$

3 Generating Random Variables

3.1 General Methods

3.1.1 Random Numbers

Everything starts with generating X_1, X_2, .. iid U[0,1]. There are simply called random numbers. There are some ways to get these:

- random number tables
- numbers taken from things like the exact (computer) time
- quantum random number generators
- ...

The R package random has the routine randomNumbers which gets random numbers from a web site which generates them based on (truely random) admospheric phenomena.

```
require(random)
randomNumbers(20, 0, 100)
```

```
##      V1 V2 V3 V4  V5
## [1,] 40 34 12 96  55
## [2,] 79 37  5 42  43
## [3,] 81 66 58 35 100
## [4,] 81 35  1 36  17
```

3.1.2 Pseudo-Random Numbers

These are numbers that look random, smell random ...

Of course a computer can not do anything truly random, so all we can do is generate X_1, X_2, .. that **appear** to be iid U[0,1], so-called pseudo-random numbers.

Luckily, some people are really good at that!

####**Example** A *linear congruential generator* works as follows:

start with a **seed** X_0, calculate

$X_{n+1}=(aX_n+c) \bmod m$

where a, c and m are chosen such that

- c and m are relatively prime
- a-1 is divisible by all prome factors of m
- a-1 is a multiple of 4 if m is a multiple of 4

A well known algorithm called PRNG in Numerical Recipes in C uses a=1664525, c=1013904223 and $m=2^{32}$

Some computer programs (like R) have them already built in, most general computer languages (like C) do not. There are many excellent ones availabe on the Internet.

Some issues to be aware of:

- All pseudo random number generators are cyclic, that is there is an N such that $X_1=X_N$, $X_2=X_{N+1}$ etc. For any decent method we have N in the billions. For the one above $N=m=2^{32}$
- All pseudo random number generators have a SEED, usually an integer. If you want to generate the same sequence you can do this by specifying this SEED.

in R if you want to generate the same sequence again then use the command set.seed(SEED) where SEED is an integer.

```
sample(1:3, size=10, replace=TRUE)
```

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

```
sample(1:3, size=10, replace=TRUE)
```

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

```
set.seed(1111)
sample(1:3, size=10, replace=TRUE)
```

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

```
set.seed(1111)
sample(1:3, size=10, replace=TRUE)
```

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

This can be very useful when writing a simulation and getting an error every 10000 or so runs!

- There are often subtle differences between compilers so don't expect the same program to generate the same sequence on different computers.

3.1.3 Generating Discrete Random Variables

- **The Inverse Transform Method**

Say we want to generate a random variable X from a distribution with density

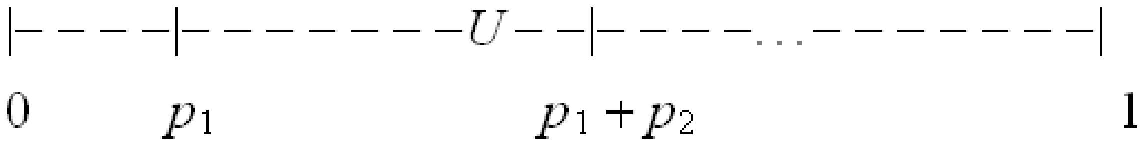
$$f(x_j) = P(X = x_j) = p_j$$

j = 1, 2, 3, ..

Here is a simple algorithm to do this:

Step 1: generate $U \sim U[0,1]$, set j = 1, p = p_1
Step 2: if U < p, set X = x_j, **done**
Step 3: set j = j+1, p = p+p_j, goto Step 2

Why this works:



Here we have $p_1 < U < p_1 + p_2$, so we set X = x_2

The routine **gendisc1** runs this algorithm:

```
gendisc1 <- function (n, x, p) {
  y <- rep(0, n)
  m <- length(x)
  p <- p/sum(p)
  cdf <- cumsum(p)
  for(i in 1:n) {
    U <- runif(1)
    for(j in 1:m) {
        if(U < cdf[j]) {
            y[i] <- x[j]
            break
        }
    }
  }
  y
}
table(gendisc1(n=1000, x=letters[1:5], p=1:5))/1000
```

```
##
##     a     b     c     d     e
## 0.063 0.134 0.189 0.265 0.349
```

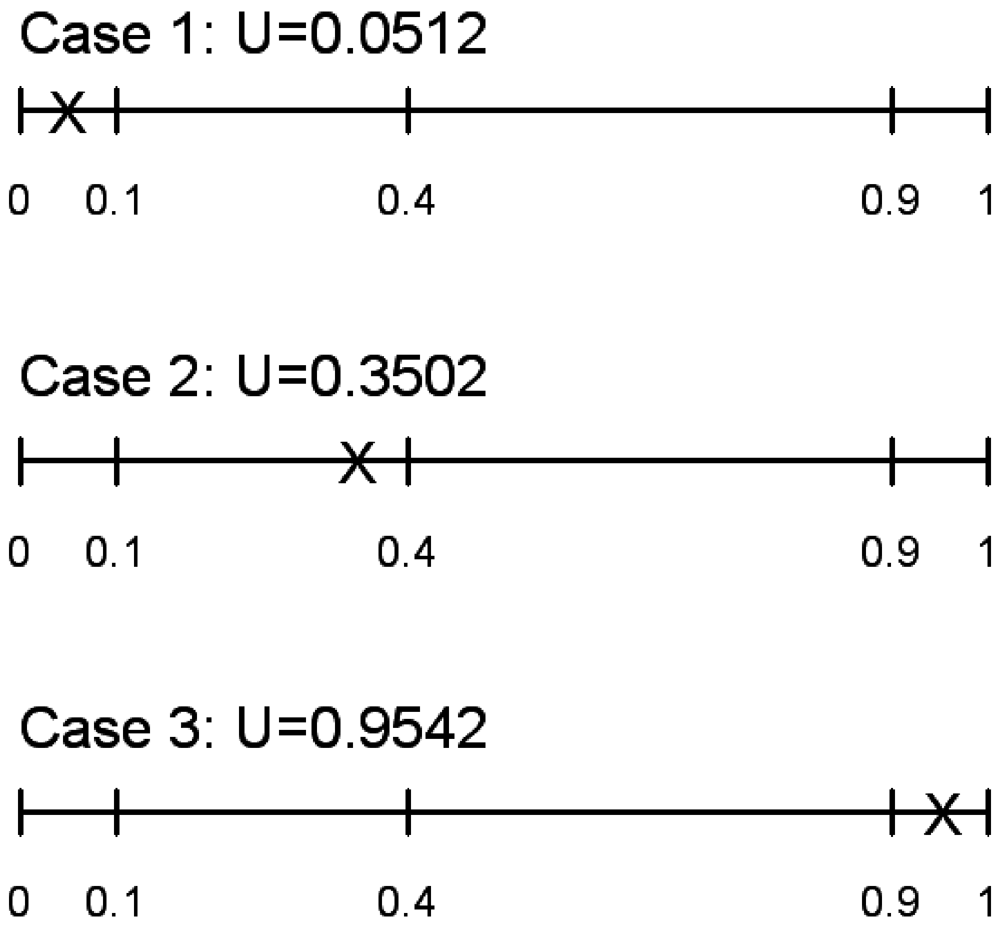
How this works:

generate data from the following rv:

P(X = 0) = 0.1, P(X = 1) = 0.3, P(X = 2) = 0.5 and P(X = 4) = 0.1

Now we generate U~U[0,1] and we get:

Case 1: U=0.0512, then we have the following:



so U < p_1, and we set X = x_1 = 0

Case 2: U=0.3502, then p_1 < U < p_1+p_2, and we set X = x_2 = 1

Case 3: U = 0.9542, then U > $p_1+p_2+..+p_{k-1}$, and we set X = x_4 = 4

Notice that the values of X (x_1,..) are almost irrelevant, in fact we can just generate data with values 1, 2, .., and in the end change the “labels”: “1”→x_1, “2”→x_2 etc.

Theorem the above algorithm generates the required rv.
proof
Remember that $U \sim U[0,1]$, so P(U < x) = x if 0 < x < 1

$P(X = x_1) = P(U < p_1) = p_1$

say k>1, then

$P(X = x_k) = P(p_1+..+p_{k-1} < U < p_1+..+p_k) =$
$(p_1+..+p_k) - (p_1+..+p_{k-1}) = p_k$

####**Example** Generate 100000 observations from $X \sim Bin(10, 0.65)$

Of course there is a routine in R built in to do this:

```
rbinom(10000, 10, 0.65)
```

You can check that the correct data was generated by comparing

```
tmp <- proc.time()
x <- table(rbinom(100000, 10, 0.65))
proc.time() - tmp
```

```
##    user  system elapsed
##    0.02    0.00    0.01
```

```
y <- round(100000*dbinom(0:10, 10, 0.65))
rbind(x, y)
```

```
##   0  1   2    3    4     5     6     7     8    9   10
## x 2 51 461 2196 7070 15130 23679 25173 17619 7253 1366
## y 3 51 428 2120 6891 15357 23767 25222 17565 7249 1346
```

Or you can use our routine above:

```
tmp <- proc.time()
x <- table(gendisc1(100000, 0:10, dbinom(0:10, 10, 0.65)))
proc.time() - tmp
```

```
##    user  system elapsed
##    0.48    0.00    0.50
```

```
rbind(x, y)
```

```
##   0  1   2    3    4     5     6     7     8    9   10
## x 2 46 447 2140 6949 15370 23840 25138 17530 7217 1321
## y 3 51 428 2120 6891 15357 23767 25222 17565 7249 1346
```

So this works but is a bit slow. Can we speed it up? Consider this:

$\text{dbinom}(0, 10, 0.65) = 2.76 \times 10^{-5}$, so we almost never choose 0, but we check it in the computer program every single time.

dbinom(6,10,0.65) - dbinom(5,10,0.65) = 0.2522 is the biggest interval and about 25% of the time U is in there, but in order to get there our program first needs to check 0, 1, 2, ,3 , 4 and 5.

This give us an idea for a slight improvement:

order p and x by decreasing size of p. Here is the table of x and p:

x	p
0	0.000
1	0.001
2	0.004
3	0.021
4	0.069
5	0.154
6	0.238
7	0.252
8	0.176
9	0.072
10	0.013

Here is the same table, ordered by the p's:

x	p
7	0.252
6	0.238
8	0.176
5	0.154
9	0.072
4	0.069
3	0.021
10	0.013
2	0.004
1	0.001
0	0.000

so now if U < 0.252 we set x $= 7$, if $0.252 <$ U $< 0.252+0.238 = 0.49$ set x $= 6$ and so on

The routine **gendisc2** does this.

```
gendisc2 <- function (n, x, p) {
  y <- rep(0, n)
  m <- length(x)
  x <- x[order(p, decreasing = TRUE)]
  p <- sort(p, decreasing = TRUE)
  print(rbind(x, p))
  p <- cumsum(p)
  for (i in 1:n) {
      U <- runif(1)
      for (j in 1:m) {
          if (U < p[j]) {
              y[i] <- x[j]
              break
          }
      }
  }
  y
}
```

Check

```
tmp <- proc.time()
x <- table(gendisc2(100000, 0:10,
                    dbinom(0:10, 10, 0.65)))
```

```
##        [,1]      [,2]     [,3]      [,4]       [,5]      [,6]       [,7]
## x 7.0000000 6.0000000 8.000000 5.0000000 9.00000000 4.0000000 3.00000000
## p 0.2522196 0.2376685 0.175653 0.1535704 0.07249169 0.0689098 0.02120302
##          [,8]        [,9]        [,10]        [,11]
## x 10.00000000 2.000000000 1.0000000000 0.000000e+00
## p  0.01346274 0.004281378 0.0005123017 2.758547e-05
```

```
proc.time() - tmp
```

```
##    user  system elapsed
##    0.81    0.00    0.82
```

```
rbind(x, y)
```

```
##   0  1   2    3    4     5     6     7     8    9   10
## x 3 51 417 2120 6904 15334 23775 25327 17487 7180 1402
## y 3 51 428 2120 6891 15357 23767 25222 17565 7249 1346
```

rbinom is still much faster. Another way in R to speed things up is to "vectorize" the program.

####**Example** Generate observations from $X \sim G(0.01)$.

Here we have the additional problem that the vector p is infinite.

Computers cannot handle infinitly large objects, so we need to "truncate" p. Here is one way to do this:

1. Find x_1 and x_2 such that $P(x_1 < X < x_2) = 0.999999$

```r
qgeom(c(0.0000005, 0.9999995), 0.01) + 1
```

```
## [1]    1 1444
```

If U < 0.0000005 + P(X = 1) = 0.0100005, set x = 1

If U > 0.9999995, set x = 1444

otherwise do as above.

- **The Accept-Reject Algorithm**

Suppose we have a method for generating a random variable having density $\{q_j, j = 1, 2, ..\}$ and we want to generate r.v. from a distribution with density $\{p_j, j = 1, 2, ..\}$. We can do this by first simulating a r.v. Y from $\{q_j\}$ and then "accepting" this simulated value with probability proportional to p_Y/q_Y.

Specifically, let c be a constant such that $p_j/q_j \le c$ for all j such that $p_j > 0$. Then

Step 1: generate Y from density $\{q_j\}$

Step 2: generate $U \sim U[0, 1]$

Step 3: If $U < p_Y/(cq_Y)$, set X = Y and stop. Otherwise go back to 1

Notation Y is often called an *auxiliary* variable. In a somewhat different context later it will also be called a *proposal density*, and sometimes either of these terms is used.

####**Example** Say we want to generate a r.v X with values x in {1, 3, 5, 7} and probabilities p = (0.1, 0.5, 0.1, 0.3).

First we need a r.v. Y which is easy to generate and takes 4 values (not necessarily the same as in X though!). We can use for this the r.v. that chooses a number from 1 to 4 at random, using the sample(1:4, 1) command. This has density q = (1/4, 1/4, 1/4, 1/4), so

p/q = (0.4, 2, 0.4, 1.2)

and if we set c = 2 we have

$p_j/q_j \le$ c for all j.

with this the accept-reject algorithm for this problem is implemented in **gendisc3**:

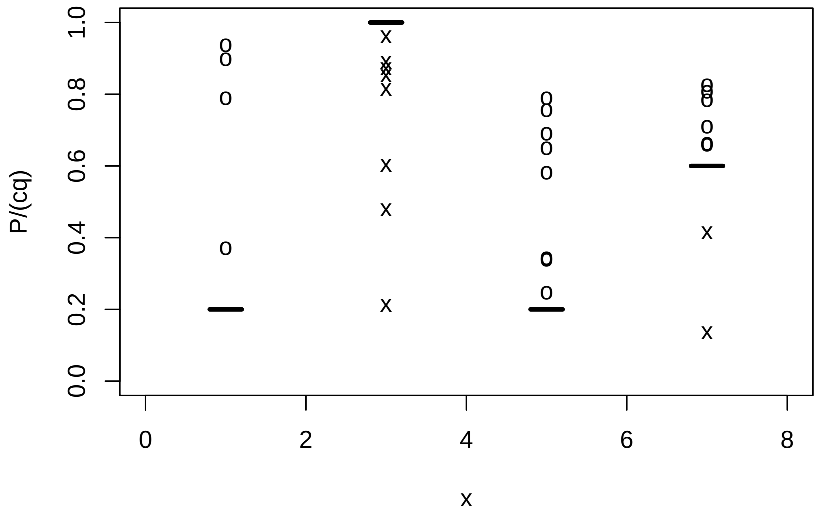
```r
gendisc3 <- function (n, x, p) {
  z <- rep(0, n)
  m <- length(x)
  q <- rep(1/4, 4)
  for (i in 1:n) {
      for (j in 1:100) {
          U <- runif(1)
          Y <- sample(1:4, 1)
          if (U < (p[Y]/(2*q[Y]))) {
              z[i] <- x[Y]
              break
          }
      }
  }
  z
}
table(gendisc3(n=10000,
      x=c(1, 3, 5, 7),
      p=c(0.1, 0.5, 0.1, 0.3)))/10000
```

```
##
##      1      3      5      7
## 0.0992 0.5018 0.1010 0.2980
```

Why this works: if our "candidate" r.v Y picks a value y which has a high probability in p, it will often be accepted and we get many of these. If on the other hand Y picks a value which has a low probability in p, it will rarely be accepted and we get only a few of those. The method is illustrated in **accrej.ill**.

```
accrej.ill <- function (n) {
  x <- c(1, 3, 5, 7)
  p <- c(0.1, 0.5, 0.1, 0.3)
  q <- rep(1/4, 4)
  X <- rep(0, n)
  plot(c(0, 8), c(0, 1), type = "n",
       xlab = "x", ylab = "P/(cq)")
  segments(x - 0.2, 2 * p, x + 0.2, 2 * p, lwd = 3)
  for (i in 1:n) {
      for (j in 1:100) {
          Y <- sample(1:4, 1)
          U <- runif(1)
          if (U < p[Y]/(2 * q[Y])) {
              X[i] <- x[Y]
              points(x[Y], U, pch = "x")
              break
          }
          else {
              points(x[Y], U, pch = "o")
          }
      }
  }
  X
}
```

```
accrej.ill(10)
```



```
##  [1] 3 3 3 3 7 3 3 3 3 7
```

Theorem

The accept-reject algorithm generates a r.v. X such that

$P(X = x_j) = p_j$

In addition, the number of iterations of the algorithm needed until X is found is a geometric r.v. with mean c.

proof

$$
\begin{aligned}
&P(Y=j, Y \text{ is accepted}) = \\
&P(Y=j)P(Y \text{ is accepted}|Y=j) = \\
&q_j \frac{p_j}{cq_j} = \frac{p_j}{c} \\
&P(Y \text{ is accepted}) = \\
&\sum_{j=1}^{\infty} P(Y=j, Y \text{ is accepted}) = \\
&\sum_{j=1}^{\infty} \frac{p_j}{c} = \frac{1}{c}
\end{aligned}
$$

Now each iteration is a Bernoulli trial with success probability 1/c, and successive trials are independent. Therefore the number of trials needed until the first success is a geometric r.v. with mean c. Also

$$P(X = x_j) =$$
$$\sum_n P(j \text{ is accpeted at } n^{th} \text{ trial}) =$$
$$\sum_n (1-1/c)^{n-1}\frac{p_j}{c} = p_j$$

####**Example** say we want to generate a rv X with P(X = k) = a_k, k = 1, 2, .., N for some fixed and known N > 1. (Of course the sample command with the argument prob = .. will do just that, but let's write our own routine).

Note that we here we know the p_j's only up to a constant.

For the rv Y we will simply use U[1, 2, .., N], that is

P(Y = k) = 1/N

Now

$$P(X=j)=a_j, j=1,..,N; a_j \ge 0 p_j = \frac{a_j}{\sum a_i} P(Y=j) = \frac{1}{N}, j=1,..,N \frac{p_j}{q_j} = \frac{Na_j}{\sum a_i} \max\{\frac{p_j}{q_j}\} = \max\{\frac{Na_j}{\sum a_i}\}$$

so we get that we don't actually need to find $\sum a_i$! In general we don't need to have the probabilities "normalized", that is sum up to 1.

Now for the routine:

```
N <- 6
a <- c(1, 3, 1, 6, 10, 4)
n <- 10000
x <- rep(0, n)
counter <- 0
for (i in 1:n) {
  repeat {
    counter <- counter+1
    y <- sample( 1:N, 1)
    if (runif(1) <= a[y]/max(a)) {
       x[i] <- y
       break
    }
  }
}
out <- data.frame( sim=c(table(x)), calc=a/sum(a)*n)
rownames(out) <- 1:N
out
```

```
##    sim calc
## 1  400  400
## 2 1176 1200
## 3  365  400
## 4 2407 2400
## 5 4002 4000
## 6 1650 1600
```

```
c(counter/n, N*max(a)/sum(a))
```

```
## [1] 2.3885 2.4000
```

Also the theorem says that a new X is generated every c = 2.4 tries, and the routine shows this to be true.

####**Example** say we want to generate r.v.'s X such that

$P(X=k)=6/(\pi^2k^2), k=1,2,...$

Recall

$\sum_{k=1}^{\infty}\frac{1}{k^2}=\frac{\pi^2}{6}$

We need a r.v Y on {1,2,..} which we can generate. There are two discrete rv's we know which have infinitely many values, the geometric and the Poisson. But there is a problem with those two:

Poisson:

$$\frac{p_j}{q_j} = \frac{6/\pi^2 j^2}{\frac{\lambda^j}{j!}e^{-\lambda}} = k\frac{j!}{j^2\lambda^j} \rightarrow \infty \text{ as } j \rightarrow \infty$$

$$\text{so } c = \max\left\{\frac{p_j}{q_j}\right\} = \infty$$

Geometric:

$$\frac{p_j}{q_j} = \frac{6/\pi^2 j^2}{p(1-p)^j} = k\frac{1}{j^2(1-p)^j} \rightarrow \infty \text{ as } j \rightarrow \infty$$

$$\text{so } c = \max\left\{\frac{p_j}{q_j}\right\} = \infty$$

We do know, though, a continuous r.v. that goes to 0 very slowly, namely the Cauchy. Actually, its density $f(x)=1/(\pi(1+x^2))$ already has the right "size" for our problem. We can do this, then by "discretizing" the Cauchy: Let $Z\sim$ Cauchy and define

Y = i($I_{[-i,-i+1]}$(Z)+$I_{[i-1,i]}$(Z)) for i=1,2,..

So

if |Z|<1 → Y=1

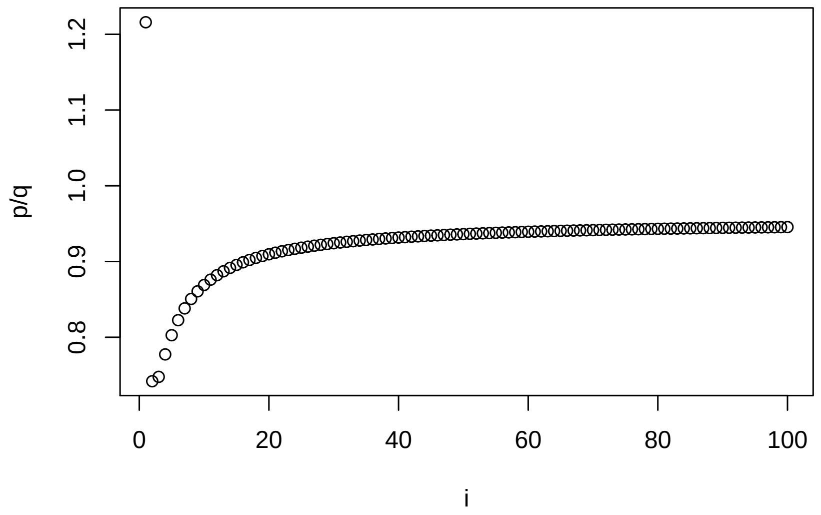
if 1<|Z|<2 → Y=2

and so on. Now

$$\begin{aligned}
&q_i = P(Y = i) = \\
&P(-i < Z < -i + 1 \text{ or } i - 1 < Z < i) = \\
&2P(i - 1 < Z < i) = \\
&2\int_{i-1}^{i} \frac{1}{\pi(1 + z^2)} dz = \\
&\frac{2}{\pi}\left(\arctan(i) - \arctan(i - 1)\right)
\end{aligned}$$

Now we need max{p_j/q_j}. Doing this via calculus would be quite difficult because we would end up with a nonlinear equation which we would need to solve numerically. A different approach is to just plot j vs. p_j/q_j and see what it looks like:

```
gendisc4 <- function (n, findc = FALSE) {
  if (findc) {
      i <- 1:n
      p <- 6/pi^2/i^2
      q <- 2/pi * (atan(i) - atan(i - 1))
      plot(i, p/q)
      return(max(p/q))
  }
  x <- rep(0, n)
  const <- 3/pi/1.216
  for (i in 1:n) {
      for (j in 1:100) {
          z <- rcauchy(1)
          y <- floor(abs(z)) + 1
          u <- runif(1)
          if (u <= const/(y^2*(atan(y)-atan(y-1)))) {
              x[i] <- y
              break
          }
      }
  }
  table(x)/n
}
gendisc4(100, TRUE)
```



```
## [1] 1.215854
```

We see that p_j/q_j appears to approach a limit a little below 1 as j goes to infinity, but it has a value of 1.216 at j=1, so we can reasonably guess that c = 1.216 should work for us. (Of course we can verify all that by taking derivatives and using de L'Hospital's rule).

With this we can generate these r.v.'s, again using **gendisc4**:

```
tmp <- gendisc4(10000)
head(tmp)
```

```
## x
##      1      2      3      4      5      6
## 0.6094 0.1563 0.0632 0.0390 0.0223 0.0176
```

```
tail(tmp)
```

```
## x
##    733   1280   1392   1872   6892 180337
##  1e-04  1e-04  1e-04  1e-04  1e-04  1e-04
```

Alternatively we could of course have used the inverse transform method, but notice that in this r.v. occasionally we see very large values, and so the inverse transform method would be quite slow.

In the theorem above it says that the mean number of trials until the first success, which is the number of "tries" until we find an acceptable candidate, has a geometric rv with mean c. Obviously the smaller c is, the faster we find a Y that is accepted, the faster we generate our observations.

####**Example** say we want to generate data from a discrete rv f with $P(X = x) = c\exp(-x^{3/2})$, x = 1, 2, … Let's assume we know how to generate data from a G(0.5). Then

$$\frac{\exp(-x^{3/2})}{(1/2)^x} =$$
$$\frac{\exp(-x^{3/2})}{\exp(-x\log(2))} =$$
$$\exp\left(x\log(2) - x^{3/2}\right)$$
$$\frac{d}{dx}\left\{\exp\left(x\log(2) - x^{3/2}\right)\right\} =$$
$$\exp\left(x\log(2) - x^{3/2}\right)\left(\log(2) - \frac{3}{2}\sqrt{x}\right) < 0$$

so max{ r_j/q_j } = r_1/ q_1 = exp(-1)/0.5 = 0.7357589

So the algorithm is

```
repeat {

  if ( runif(1) < $2^y$exp(-$y^{3/2}$)/0.7357589 ) { x<- y ; break }
}
```

3.1.4 Continuous Distributions

Accept-Reject Algorithm

This is very similar (actually the same) as the method for discrete r.v. Assume want to generate a r.v. X with density f. We have a way to generate a r.v. Y with density g. Let c be a constant such that

f(x)/g(x) $\le$ c for all x.

Then the accept-reject algorithm is as follows:

Step 1: generate Y from pdf g
Step 2: generate $U \sim U[0,1]$
Step 3: If U < f(Y)/(cg(Y)), set X = Y and stop. Otherwise go back to 1

We have the same theorem as for the discrete case:

Theorem

The accept-reject algorithm generates a r.v. X with density f.

In addition, the number of iterations of the algorithm needed until X is found is a geometric r.v. with mean c.

proof same as above.

Note: we do have to be careful because of course g(x) can be 0.

This is ok as long as f(x) is 0 as well but not if f(x) > 0.

Basically we need Y to live on the same set as X. (We say X and Y have the same *support*)

####**Example** generate a r.v. X with density f(x) = 6x(1-x) 0 < x < 1.

Here we can use $Y \sim U[0,1]$, with g(x) = 1.

Let's find max{f(x)/g(x) | 0 < x < 1}. Taking derivatives we find

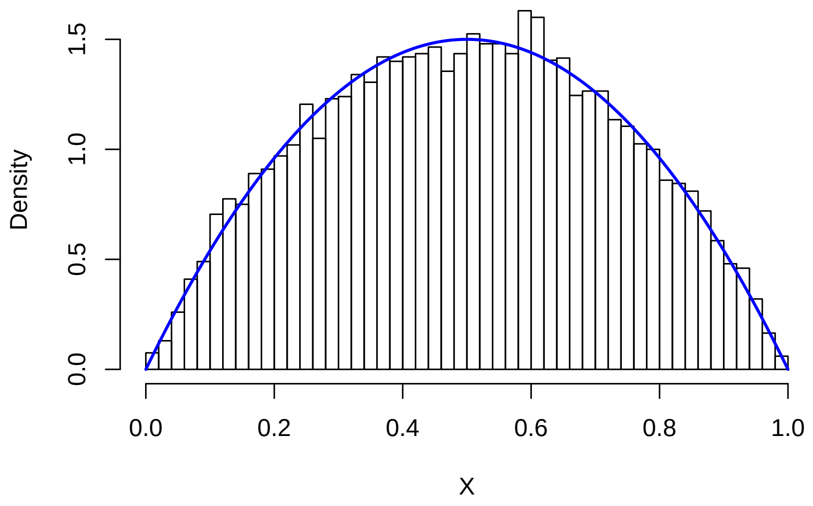
d/dx {6x(1-x)} = 6-12x = 0, x = 1/2.

This is a maximum (?) so we have

$$c = \max\left\{\frac{f(x)}{g(x)}; 0 < x < 1\right\} = f(\frac{1}{2}) = \frac{3}{2}\frac{f(x)}{cg(x)} = \frac{6x(1-x)}{3/2*1}/ = 4x(1-x)$$

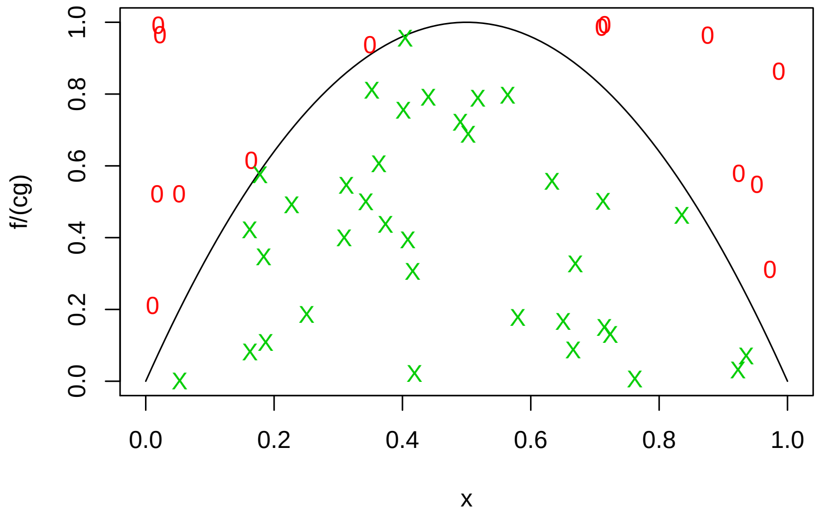
So here is the routine:

```r
gencont1 <- function(n, findc = FALSE, Show = FALSE) {
  if (findc) {
      x <- seq(0, 1, length = 100)
      plot(x, 6 * x * (1 - x), type = "l")
      return(max(6 * x * (1 - x)))
  }
  X <- rep(0, n)
  for (i in 1:n) {
      for (j in 1:100) {
          Y <- runif(1)
          if (runif(1) <= 4 * Y * (1 - Y)) {
              X[i] <- Y
              break
          }
      }
  }
  if (Show) {
      hist(X, 50, freq = FALSE, main="")
      x <- seq(0, 1, length=250)
      lines(x, 6*x*(1-x),
            lwd=2, col="blue")
  }
  X
}
tmp <- gencont1(10000, Show=TRUE)
```



why the accept-reject algorithm works is illustrated in **accrej1.ill**

```
accrej1.ill <- function(n) {
  f <- function(x) 6*x*(1-x)
  g <- function(x) 1
  c <- 3/2
  x <- seq(0, 1, length = 100)
  plot(x, f(x)/(c * g(x)),
       xlab = "x", ylab = "f/(cg)", type = "l",
        xlim = c(0, 1), ylim = c(0, 1))
  for(i in 1:n) {
    Y <- runif(1)
    U <- runif(1)
    if (U <= f(Y)/(c * g(Y))) {
      points(Y, U, pch = "X", col = 3)
    }
    else {
      points(Y, U, pch = "0", col = 2)
    }
  }

}
accrej1.ill(50)
```



####**Example** generate a r.v. $X \sim \text{Gamma}(3/2, 1)$.

Now f(x) $>$ 0 for x $>$ 0, so we need a Y that "lives" on $(0, \infty)$ and that we already know how to generate. Let's say we know how to generate $Y \sim Exp(1/\lambda)$.

What value of λ should we use? Note that

EX $= 3/2$

EY $= \lambda$

so maybe $\lambda = 3/2$ is a good idea.

with this we need to find c. Again we will try to find max{f(x)/g(x)}. We have

$$f(x) = \frac{1}{\Gamma(3/2)1^{3/2}} x^{3/2-1} e^{-x/1} = \frac{2}{\sqrt{\pi}} \sqrt{x}\, e^{-x} \text{ and } g(x) = \frac{2}{3} e^{-2x/3}$$

$$\frac{f(x)}{g(x)} = \frac{3}{\sqrt{\pi}} \sqrt{x}\, e^{-x/3} \text{ and } \frac{d}{dx}\left\{\frac{f(x)}{g(x)}\right\} = \frac{d}{dx}\left\{\frac{3}{\sqrt{\pi}} \sqrt{x}\, e^{-x/3}\right\} =$$

$$\frac{3}{\sqrt{\pi}}\left(\frac{1}{2\sqrt{x}} e^{-x/3} + \sqrt{x}\, e^{-x/3} \cdot (-1/3)\right) = 0$$

$$\Rightarrow \frac{1}{2\sqrt{x}} e^{-x/3} = \sqrt{x}\, e^{-x/3}/3 \Rightarrow 3 = 2x$$

$$\Rightarrow x = 3/2$$

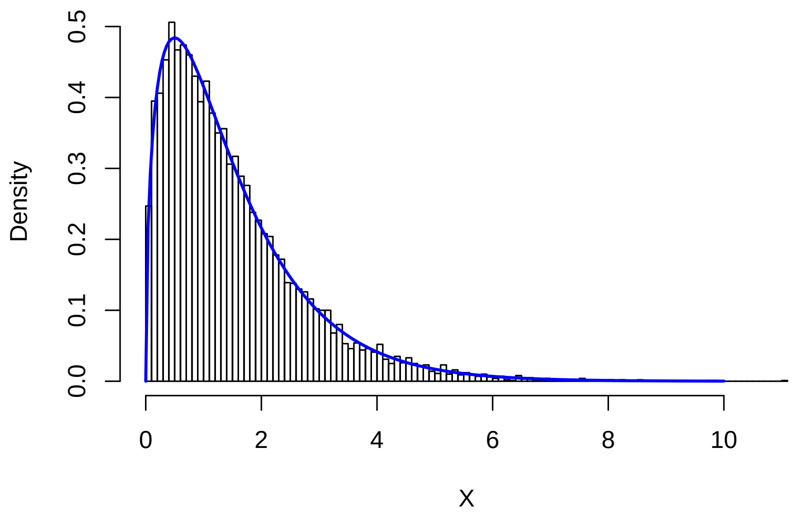
and so

$$c = \max\left\{\frac{f(x)}{g(x)}, x > 0\right\} = \frac{f(3/2)}{g(3/2)} = \frac{\frac{2}{\sqrt{\pi}}\sqrt{3/2}\,e^{-3/2}}{\frac{2}{3}e^{-2(3/2)/3}} = \frac{3^{3/2}}{\sqrt{2\pi e}}$$

$$\text{and } \frac{f(x)}{cg(x)} = \sqrt{\frac{2ex}{3}}\,e^{-x/3} = 1.34617\sqrt{x}\,e^{-x/3}$$

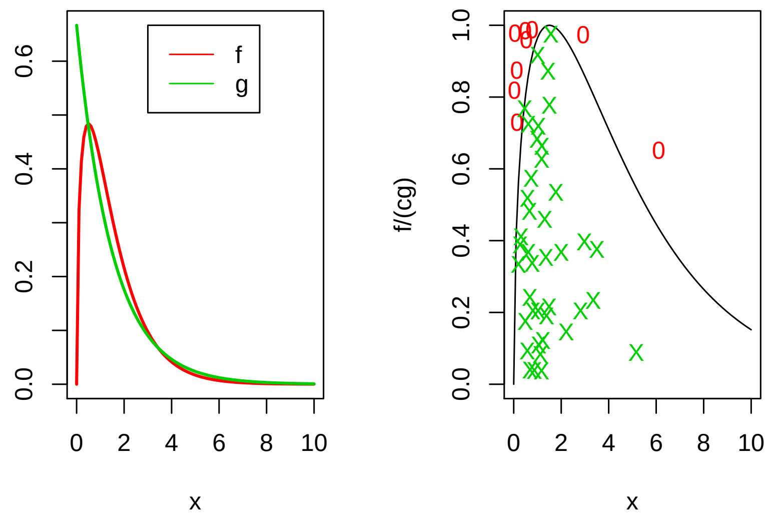
The routine is implemented in **gencont2**:

```
gencont2 <- function (n, findc = F, Show = FALSE) {
  if (findc) {
    x <- seq(0, 10, length = 100)
    plot(x, 3/sqrt(pi)*sqrt(x)*exp(-x/3), type = "l")
    return(max(3/sqrt(pi) * sqrt(x) * exp(-x/3)))
  }
  X <- rep(0, n)
  for (i in 1:n) {
    for (j in 1:100) {
      Y <- rexp(1, 2/3)
      if(runif(1)<=1.34617*sqrt(Y)*exp(-Y/3)) {
          X[i] <- Y
          break
      }
   }
  }
  if (Show) {
        hist(X, breaks=100, freq=FALSE, main="")
        x <- seq(0, 10, length=250)
        lines(x, dgamma(x, 3/2, 1),
            lwd=2, col="blue")
    }
    X
}
tmp <- gencont2(10000, Show = TRUE)
```



why the accept-reject algorithm works for this example is illustrated on this example in

```r
accrej2.ill <- function (n) {
  f <- function(x) dgamma(x, 3/2, 1)
  g <- function(x) 2/3*exp(-2/3*x)
  c <- 3^(3/2)/sqrt(2*pi*exp(1))
  x <- seq(0, 10, length = 100)
  par(mfrow = c(1, 2))
  plot(x, f(x),
       xlab = "x", ylab = "", type = "l",
       ylim = c(0, 2/3), col = 2, lwd = 2)
  lines(x, g(x), col = 3, lwd = 2)
  legend(3, 2/3, c("f", "g"), lty=c(1, 1), col=c(2, 3))
  plot(x, f(x)/(c * g(x)),
       xlab = "x", ylab = "f/(cg)", type = "l")
  for (i in 1:n) {
      Y <- rexp(1, 2/3)
      U <- runif(1)
      if (U <= f(Y)/(c * g(Y))) {
          points(Y, U, pch = "X", col = 3)
      }
      else {
          points(Y, U, pch = "O", col = 2)
      }
  }

}
accrej2.ill(50)
```



Above we picked $\lambda = 3/2$ because this matched up the means of X and Y, a reasonable choice. But is it an optimal choice? Let's see:

Optimal here means a choice of λ that minimizes c. Now if we repeat the above calculation with λ instead of 3/2 we find

$$\frac{f(x)}{g(x)} = \frac{2\lambda}{\sqrt{\pi}}\sqrt{x}\exp\left(-\frac{\lambda-1}{\lambda}x\right)$$

$$\frac{d}{dx}\left\{\frac{f(x)}{g(x)}\right\} = \frac{2\lambda}{\sqrt{\pi}}\left(\frac{1}{2\sqrt{x}} + \sqrt{x}\left(-\frac{\lambda-1}{\lambda}\right)\right)\exp\left(-\frac{\lambda-1}{\lambda}x\right) = 0$$

$$x = \frac{\lambda}{2(\lambda-1)}$$

$$c(\lambda) = \frac{f\left(\frac{\lambda}{2(\lambda-1)}\right)}{g\left(\frac{\lambda}{2(\lambda-1)}\right)} = \frac{\lambda}{\sqrt{2e\pi}}\sqrt{\frac{\lambda}{\lambda-1}}$$

$$c'(\lambda) = \frac{\lambda}{\sqrt{2e\pi}}\,\frac{1}{2}\sqrt{\frac{\lambda-1}{\lambda}}\,\frac{2\lambda-3}{(\lambda-1)^2} = 0 \text{ if } \lambda = \frac{3}{2}$$

This is the minimum (?) and so our choice was in fact optimal.

3.1.5 Generating Random Vectors

as one might expect, generating data from random vectors is generally harder than for one dimensional random variables. To begin with though, at least for the case of finite rv's there is nothing new:

say we have X = (X_1, .., X_n) where X_j takes values

$$x_{j1} \quad \dots \quad x_{jn_j}$$

Then all we need to do is generate data from the random variable X' with values

$$x_{11} \quad x_{1n_1} \quad x_{21} \quad \dots \quad x_{nn_n}$$

and their respective probabilities.

Example generate data from the random vector (X,Y) with density

	y=0	y=1
x=0	0.1	0.5
x=1	0.2	0.2

Instead generate data from X' with values 1-4 and probabilities (0.1, 0.5, 0.2, 0.2). Then

if X' = 1 set X = 0, Y = 0
if X' = 2 set X = 0, Y = 1
if X' = 3 set X = 1, Y = 0
if X' = 4 set X = 1, Y = 1

```r
n <- 1e6
xprime <- sample(1:4, size=n,
        replace=TRUE, prob=c(0.1, 0.5, 0.2, 0.2))
round(table(xprime)/n, 4)
```

```
## xprime
##      1      2      3      4
## 0.0993 0.5000 0.2004 0.2003
```

```r
x <- rep(0, n)
y <- x
x[xprime == 3 | xprime == 4] <- 1
y[xprime == 2 | xprime == 4] <- 1
round(table(x,y)/n, 4)
```

```
##    y
## x        0      1
##   0 0.0993 0.5000
##   1 0.2004 0.2003
```

####**Example** generate data from the random vector (X,Y,Z) with density

$f(x, y, z) = (x + y + z)/162, x, y, z \in \{1, 2, 3\}$

Note that there are 27 different combinations of values of (x, y, z), so we begin by generating data from a random variable X' with values 1 - 27 and probabilities as above.

```
gendisc5 <- function(i, j, k, n=10000) {
    xyz <- matrix(0, 27, 3)
    xyz[, 1] <- rep(1:3, 9)
    xyz[, 2] <- rep(1:3, each = 3)
    xyz[, 3] <- rep(1:3, rep(9, 3))
    p <- apply(xyz, 1, sum)/162
    xprime <- sample(1:27, size=n,
                  replace=TRUE, prob=p)
    x <- xyz[, 1][xprime]
    y <- xyz[, 2][xprime]
    z <- xyz[, 3][xprime]
    a <- 1:n
    a1 <- a[x == i]
    a2 <- a[y == j]
    a3 <- a[z == k]
    a <- table(c(a1, a2, a3))
    a <- as.numeric(names(a[a == 3]))
    c(length(a)/n, (i + j + k)/162)
}
gendisc5(1, 1, 2)
```

```
## [1] 0.02630000 0.02469136
```

For infinite discrete and for continous random vectors we still have the Accept-Reject algorithm:

####**Example** say we want to generate data from a continuous rv (X, Y, Z) with

$f(x, y, z) = 4xy; 0 \leq x, y, z \leq 1.$

Here we can generate data from (U_1, U_2, U_3) = 1 on $[0,1]^3$. Now

c = max{f(x,y,z)/g(x,y,z)} = max{4xy/1} = 4

One problem in the multivariate case is to make sure that our program generates the correct data. One idea is to check the histograms of the marginals, but of course this is not sufficient proof that there is no mistake. Here the marginals are given by

$$f_X(x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y,z)dzdy = \int_0^1\int_0^1 4xydzdy = \int_0^1 4xydy = 2xy^2|_0^1 = 2x, \ \ 0 < x < 1$$

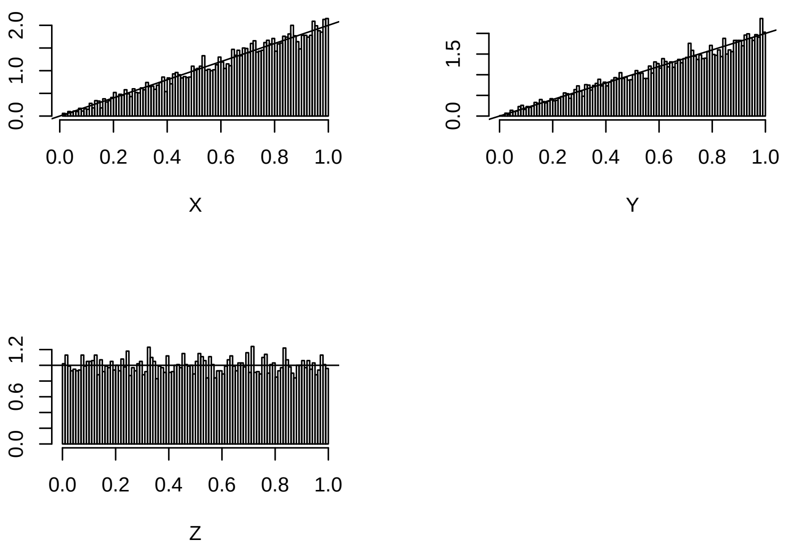
$$f_Y(y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y,z)dzdx = \int_0^1\int_0^1 4xydzdx = \int_0^1 4xydx = 2yx^2|_0^1 = 2y, \ \ 0 < y < 1$$

$$f_Z(z) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y,z)dxdy = \int_0^1\int_0^1 4xydxdy = \int_0^1 2y \cdot (x^2|_0^1)dy = 2xy^2|_0^1 =$$

$$\int_0^1 2ydy = y^2|_0^1 = 1, \ \ 0 < z < 1$$

The algorithm is in **gen_xyz(1)**.

```
gen_xyz <- function (which = 1, n = 10000) {
  xyz <- matrix(0, n, 3)
  if(which==1) {
      for (i in 1:n) {
         repeat {
            u <- runif(3)
            if (runif(1) <= u[1] * u[2])
               break
          }
          xyz[i, ] <- u
      }
  }
  if(which==2) {
    for (i in 1:n) {
      repeat {
        u <- runif(1)
        if (runif(1) <= u)
          break
      }
      xyz[i, 1] <- u
      repeat {
        u <- runif(1)
        if(runif(1)<=u)
          break
      }
      xyz[i, 2] <- u
    }
    xyz[, 3] <- runif(n)
  }
  par(mfrow = c(2, 2))
  hist(xyz[, 1], breaks = 100,
       xlab = "X", ylab = "", freq = FALSE,
        main = "")
  abline(0, 2)
  hist(xyz[, 2], breaks = 100,
       xlab = "Y", ylab = "", freq = FALSE,
        main = "")
  abline(0, 2)
  hist(xyz[, 3], breaks = 100,
       xlab = "Z", ylab = "", freq = FALSE,
        main = "")
  abline(h = 1)

}
gen_xyz(1)
```



In this case of course X,Y and Z are independent because

$f(x, y, z) = f_X(x)\ f_Y(y)\ f_Z(z)$

so we can just generate data from the marginals, see **gen_xyz(2)**.

Another idea is to generate the data from conditional distributions:

####**Example** Say (X, Y) is a discrete rv with joint density

$f(x, y) = (1-p)^2 p^x$, $x,y \in \{0, 1, ..\}, y \le x$, and $0 < p < 1$.

Note that

$$f_Y(y) = \sum_x f(x,y) = \sum_{x=y}^{\infty} (1-p)^2 p^x = (1-p)^2 \left(\sum_{x=0}^{\infty} p^x - \sum_{x=0}^{y-1} p^x \right) =$$

$$(1-p)^2 \left(\frac{1}{1-p} - \frac{1-p^y}{1-p} \right) = (1-p)p^y, \ \ y = 0, 1, 2, ..$$

and so

$$f_{X|Y=y}(x|y) = \frac{f(x,y)}{f_Y(y)} = \frac{(1-p)^2 p^x}{(1-p)p^y} = (1-p)p^{x-y}, \ \ x = y, y+1 ..$$

so we have that

Y = G-1

and

X|Y=y = G+y-1

where $G \sim G(1-p)$. The method is implemented in **gen_px**.

```
gen_px <- function(p, n = 10000) {
  y <- rgeom(n, 1 - p)
  x <- rgeom(n, 1 - p) + y
  z <- table(x, y)/n
  xvals <- sort(unique(x))
  yvals <- sort(unique(y))
  z1 <- matrix(0, length(xvals), length(yvals))
  dimnames(z1) <- list(xvals, yvals)
  for(i in 1:length(xvals))
    for (j in 1:length(yvals))
      if(yvals[j]<= xvals[i])
        z1[i, j] <- (1-p)^2*p^xvals[i]
  print(round(z1, 3))
  print(round(z, 3))
  z
}
gen_px(p=0.2)
```

```
##       0     1     2     3     4 5 6
## 0 0.640 0.000 0.000 0.000 0.000 0 0
## 1 0.128 0.128 0.000 0.000 0.000 0 0
## 2 0.026 0.026 0.026 0.000 0.000 0 0
## 3 0.005 0.005 0.005 0.005 0.000 0 0
## 4 0.001 0.001 0.001 0.001 0.001 0 0
## 5 0.000 0.000 0.000 0.000 0.000 0 0
## 6 0.000 0.000 0.000 0.000 0.000 0 0
##    y
## x       0     1     2     3     4     5     6
##   0 0.648 0.000 0.000 0.000 0.000 0.000 0.000
##   1 0.126 0.128 0.000 0.000 0.000 0.000 0.000
##   2 0.024 0.023 0.025 0.000 0.000 0.000 0.000
##   3 0.004 0.004 0.006 0.006 0.000 0.000 0.000
##   4 0.002 0.001 0.001 0.001 0.000 0.000 0.000
##   5 0.000 0.000 0.000 0.000 0.000 0.000 0.000
##   6 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```
##    y
## x        0      1      2      3      4      5      6
##   0 0.6476 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   1 0.1258 0.1279 0.0000 0.0000 0.0000 0.0000 0.0000
##   2 0.0239 0.0230 0.0252 0.0000 0.0000 0.0000 0.0000
##   3 0.0043 0.0041 0.0057 0.0064 0.0000 0.0000 0.0000
##   4 0.0015 0.0008 0.0012 0.0008 0.0005 0.0000 0.0000
##   5 0.0002 0.0002 0.0002 0.0001 0.0001 0.0002 0.0000
##   6 0.0000 0.0001 0.0000 0.0000 0.0001 0.0000 0.0001
```

####**Example** say we have a 10-dimensional rv with joint pdf

$$f(x_1,..,x_10) = c\prod x_i, 0 < x_1 < x_2 < ... < x_{10} < 1$$

For the methods we know sofar we need c:

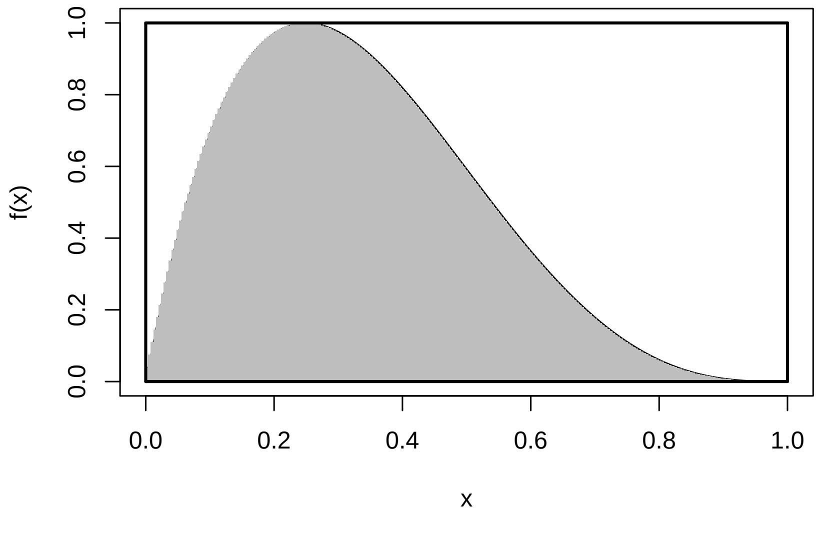
$$1 = \int_0^1\int_{x_1}^1\int_{x_2}^1 \cdots \int_{x_9}^1 c\prod_{i=1}^{10} x_i dx_{10}dx_9..dx_1 = c\int_0^1 x_1\int_{x_1}^1 x_2\int_{x_2}^1 \cdots \int_{x_9}^1 x_{10}dx_{10}dx_9..dx_1 =$$

$$c\int_0^1 x_1\int_{x_1}^1 x_2\int_{x_2}^1 \cdots \int_{x_8}^1 x_9(1-\frac{x_9^2}{2})dx_9..dx_1 = \ldots$$

so this is ugly. Doable, but ugly. Of course, if we needed this for 100 instead of 10… It turns out, though, that we can actually generate such data even without knowing c, but this discussion has to wait a bit.

3.2 The Fundamental Theorem of Simulation

Let's take another look at the Accept-Reject algorithm. Let's say we have a density f which has finite support on some interval [A,B] and we want to generate data from f. So we use Y~U[A,B] and we accept an observation if U < f(Y) . If we draw a graph similar to the **accrej.ill** routines above but with many runs it would look like this



There is another way to think of this: consider the rectangle and generate a pair of uniform rvs on it. A point is accepted if it falls into the gray region. This idea leads to the following:

First we write

$$f(x) = \int_0^{f(x)} du$$

Next consider the random vector (X, U) which has a joint density that is uniform on the set {(x,u): 0 < u < f(x)}. Now f is the marginal density of (X,U)!

It does not seem that we have gained much by this rewriting of the accept - reject algorithm, but in fact this way of thinking has a great many consequences. First of all it is much more general than it seems, for example f need not be a univariate density but can be a density of a random vector on an arbitrary space!

Also it is clear from this description that in general the auxiliary variable can be chosen to be a uniform, although in practice then some transformations might be needed. That this is more important than it seems at first is clear because of the

Theorem (Fundamental Theorem of Simulation)

Simulating

$X \sim F$

is equivalent to simulating

$(X, U) \sim U\{(x,u) : 0 < u < f(x)\}$

proof trivial

One thing that is made clear by this theorem is that we can generate X in three ways:

- first generate $X \sim F$, and then U|X=x, but this is useless because we already have X and don't need U
- first generate U, and then X|U=u, that is just the accept - reject algorithm
- generate (X, U) jointly, which will eventually turn out be the smart idea because it allows us to generate data on a larger set were simulation is easier, and then to use the pair if the constraint is satisfied.

The full generality of this approach will have to wait a bit, but here is a simple case: Say X has support [A, B], and f(x) < M for all x. So we generate a pair $(Y, U) \sim U\{0 < u < M\}$ by simulating $Y \sim U[a, b]$ and $U|Y = y \sim U[0, m]$ and take the pair iff u<f(y).

This works because

$$P\left(X \le x\right) = P(Y \le x|U < f(Y)) = \frac{P(Y \le x, U < f(Y))}{P(U < f(Y))} = \frac{\int_a^x \int_0^{f(y)} dudy}{\int_a^b \int_0^{f(y)} dudy} = \frac{\int_a^x f(y)dy}{\int_a^b f(y)dy} = \int_a^x f(y)dy$$

####**Example** Say $X \sim \text{Beta}(\alpha, \beta)$ with $\alpha, \beta > 1$.

Now take $Y \sim U[0, 1]$ and we find m by

$$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$$

$$g(x) = \log\{f(x)\} =$$

$$\log(\Gamma(\alpha+\beta)) - \log(\Gamma(\alpha)) - \log(\Gamma(\beta)) + (\alpha-1)\log x + (\beta-1)\log(1-x)$$

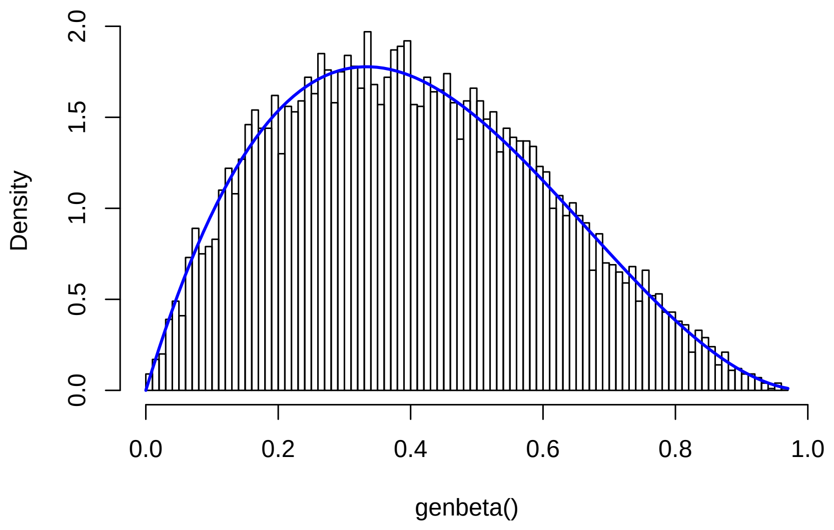
$$\frac{dg}{dx} = \frac{\alpha-1}{x} - \frac{\beta-1}{1-x} = 0$$

$$(\alpha-1)(1-x) - (\beta-1)x = 0$$

so $U \sim U[0, f((\alpha-1)/(\alpha+\beta-2))]$

this is implemented in **genbeta**:

```r
genbeta <- function(n=10000, alpha=2, beta=3) {
  x <- (alpha-1)/(alpha+beta-2)
  M <- dbeta(x, alpha, beta)
  xu <- matrix(0, n, 2)
  for(i in 1:n) {
    repeat {
      Y <- runif(1)
      U <- runif(1, 0, M)
      if(U<dbeta(Y, alpha, beta))
        {xu[i, ] <- c(Y, U); break}
    }
  }
  xu[, 1]
}
hist(genbeta(), 100, freq=FALSE, main="")
curve(dbeta(x, 2, 3),
      lwd=2, col="blue", add=TRUE)
```



The biggest restriction on the usefulness of this idea is that we need (X, U) to be in a box, so we can simulate from uniforms. This can be overcome if we allow simulating from a larger set on which uniform simulation is possible. Let's say this larger set is of the form

$$L = \{(y, u) : 0 < u < m(x)\}$$

then the constraint is of the form $m(x) < f(x)$.

Obviously because $m(x) < f(x)$ m(x) will not be a density (except if m(x)=f(x), where we are back at accept-reject) but that

$$\int m(x)dx = M < \infty$$

(if $\int m(x)dx = \infty$ uniform simulation from L would not be possible)

and so we can define $g(x) = m(x)/M$ and g is a density.

If uniform simulation on L is possible we can then use the third bullet above, that is generate $Y \sim F$ and then $U|Y = y \sim U(0, m(x))$. Now if we only accept y's with $u < f(y)$ we have

$$P(X \in A) =$$

$$P(Y \in A | U < f(Y)) =$$

$$\frac{P(Y \in A, U < f(Y))}{P(U < f(Y))} =$$

$$\frac{\int_A \left(\int_0^{f(y)} \frac{1}{m(y)} du \right) g(y) dy}{\int \left(\int_0^{f(y)} \frac{1}{m(y)} du \right) g(y) dy} =$$

$$\int_A f(y) dy$$

and we now have a generlization of the fundamental theorem:

Corollary Let $X \sim F$ and let g be a density that satisfies

$$f(x) < Mg(x)$$

for all x and some constant $M \geq 1$. Then, to simulate from f it is sufficient to generate from

Y~g

and

$U|Y = y \sim U(0, Mg(y))$

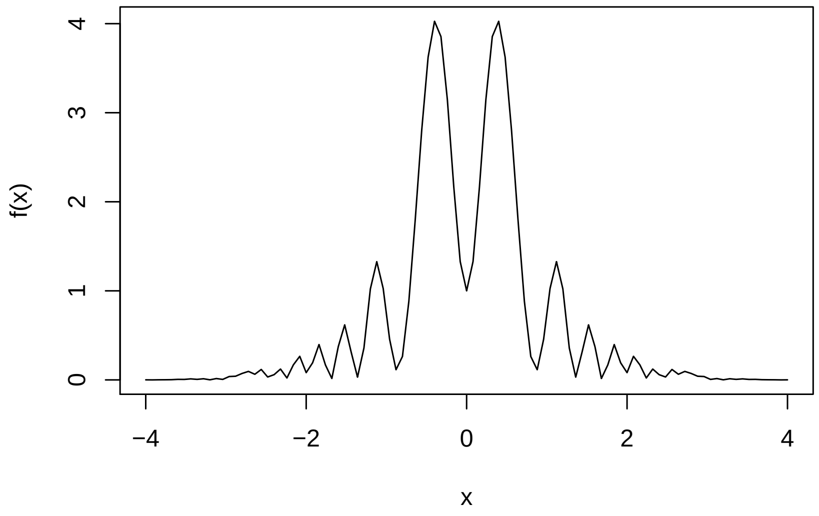
until 0 < u < f(y)

As with the basic accept-reject algorithm, it can be shown that the number of trials until acceptance has an exponential distribution with mean 1/M , so the smaller M can be chosen, the quicker the sample is generated.

####**Example**

say we want to simulate fromm the density

```r
f <- function(x) exp(-x^2/2)*(sin(6*x^2) + 3*cos(x)^2*sin(4*x)^2 + 1)
curve(f, -4, 4)
```



Notice that in order to use accept-reject we would need to find $\int f(x)dx$, already a no-trivial problem even using a numerical method.

In order to use the corollary, we need a density g with $f(x) < Mg(x)$ for some $M \geq 1$. Obviously

$$\sin(6x^2) + 3\cos(x)^2\sin(4x)^2 + 1 \leq 5$$

so if we use the standard normal density for g we have

$$\sin(6x^2) + 3\cos(x)^2\sin(4x)^2 + 1 \leq 5$$

$$f(x) = \exp(-x^2/2)[\sin(6x^2) + 3\cos(x)^2\sin(4x)^2 + 1] =$$

$$\frac{1}{\sqrt{2\pi}}\exp(-x^2/2)\sqrt{2\pi}\,[\sin(6x^2) + 3\cos(x)^2\sin(4x)^2 + 1] \leq$$

$$5\sqrt{2\pi}\,g(x)$$

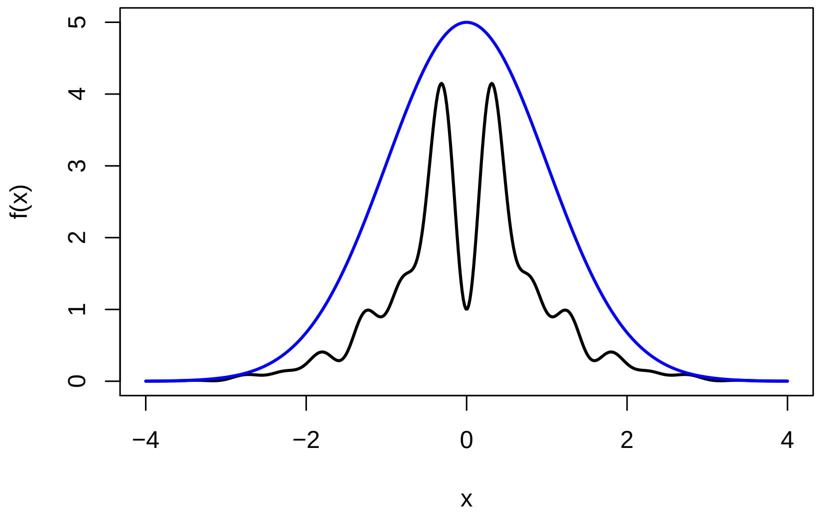
and so $M = 5\sqrt{(2\pi)} = 12.54$.

```r
fundex <- function (which = 1) {
    f <- function(x)
```

```r
      exp(-x^2/2)*(sin(6*x)^2+3*cos(x)^2*
                         sin(4*x)^2+1)
    if(which==1)
        curve(f, -4, 4, lwd = 2, n = 500)
    M <- 5*sqrt(2*pi)
    if(which==2) {
        curve(f, -4, 4, lwd=2, n=500, ylim=c(0, 5))
        x <- seq(-4, 4, length=200)
        lines(x, M*dnorm(x), lwd = 2, col = "blue")
    }
    if(which == 3) {
        n <- 1e+05
        x <- rep(0, n)
        for (i in 1:n) {
            repeat {
                y <- rnorm(1)
                u <- runif(1, 0, M * dnorm(y))
                if (u < f(y)) {
                  x[i] <- y
                  break
                }
            }
        }
        hist(x, breaks = 250, freq = FALSE, main="")
    }

}
```

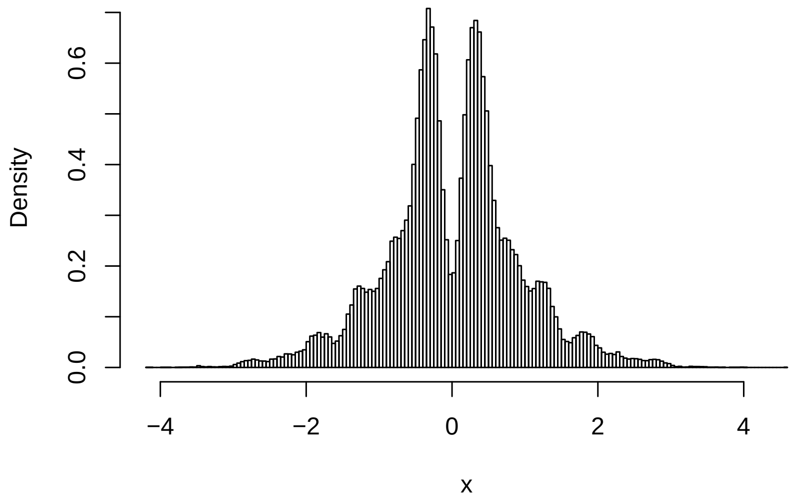
Here are f and g:

```r
fundex(2)
```



and here is the simulation:

```r
fundex(3)
```



####**Example**

say we want to generate standard normal variates. As g we will use the double exponential distribution, that is

$$g(x;\lambda) = \frac{1}{2}\lambda\exp(-\lambda|x|)$$

which, as we will see shortly, can be generated with

```r
sample(c(-1, 1), size=n)*lambda*\log(U)
```

Now

$$\frac{f(x)}{g(x;\lambda)} = \frac{\frac{1}{\sqrt{2\pi}}\exp(-x^2/2)}{\frac{\lambda}{2}\exp(-\lambda|x|)} =$$

$$\frac{2}{\lambda\sqrt{2\pi}}\exp(\lambda|x|-x^2/2)$$

$\lambda|x| - x^2/2$ is symmetric in x, and $\lambda|x| - x^2/2$ has a maximum at $x = \lambda$, so we have

$$f(x)/g(x;\lambda) \le \sqrt{2/\pi}\frac{1}{\lambda}exp(\lambda^2/2)$$

we are free to choose any λ, so it makes sense to choose it to minimize M, that is

$$\frac{d}{d\lambda}\frac{f(x)}{g(x;\lambda)} = \frac{d}{d\lambda}\sqrt{\frac{2}{\pi}}\,\frac{1}{\lambda}\exp(\lambda^2/2) =$$

$$\sqrt{\frac{2}{\pi}}\,(-\frac{1}{\lambda^2})\exp(\lambda^2/2) + \sqrt{\frac{2}{\pi}}\,\frac{1}{\lambda}\exp(\lambda^2/2)\lambda =$$

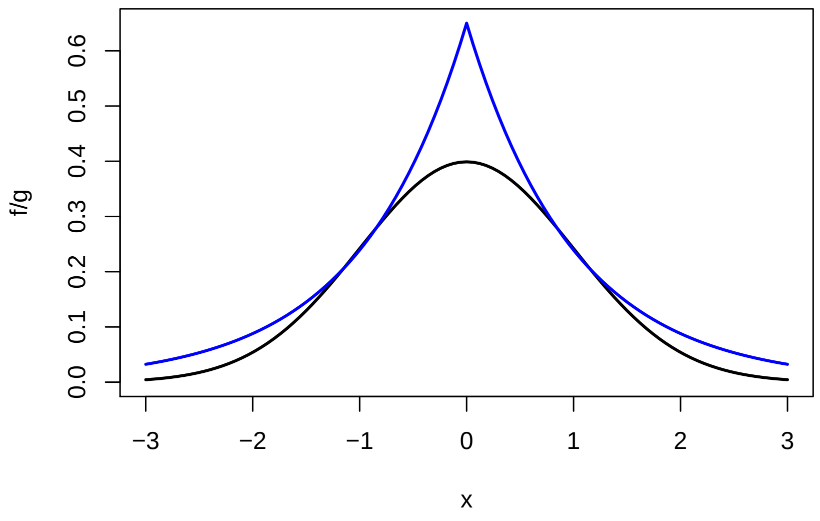
$$\sqrt{\frac{2}{\pi}}\exp(\lambda^2)\left(-\frac{1}{\lambda^2}+1\right) = 0$$

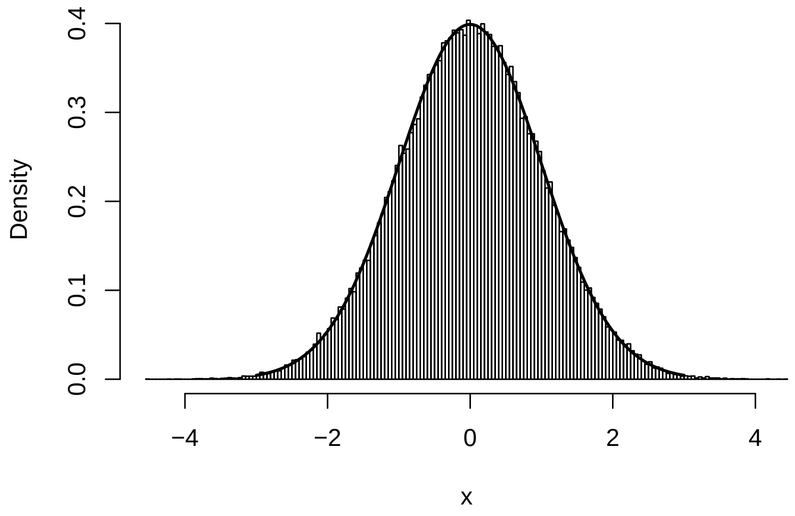
and we find $\lambda = 1$ is optimal. With it we have

$M = \sqrt{2/\pi}/\exp(1/2) = 1.3$

fundex1() draws the curve for f and M*g and does the simulation:

```r
fundex1 <- function (n=1e+05) {
    curve(dnorm(x), -3, 3,
          ylim=c(0, 0.65), lwd=2, ylab="f/g")
    curve(1.3*0.5*exp(-abs(x)), -3, 3,
          add = TRUE, col = "blue", lwd=2)
    x <- rep(0, n)
    M <- sqrt(2*exp(1)/pi)
    g <- function(x) 0.5*exp(-abs(x))
    for (i in 1:n) {
        repeat {
            y <- sample(c(-1, 1), 1)*log(runif(1))
            u <- runif(1, 0, M*g(y))
            if (u < dnorm(y)) {
                x[i] <- y
                break
            }
        }
    }
    hist(x, breaks = 250, freq = FALSE, main = "")
    z <- seq(-3, 3, length = 250)
    lines(z, dnorm(z), lwd = 2)

}
fundex1()
```





3.3 Some Special Methods

3.3.1 Exponential Distribution - General Inverse Method

####**Example** say $U \sim U[0,1]$ and let $\lambda > 0$. Set $Y = -\lambda \log(U)$. Then

$$F_Y(y) = P(Y < y) = P(-\lambda \log U < y) = P(\log U > -y/\lambda) = P(U > \exp\{-y/\lambda\}) = 1 - P(U < \exp\{-y/\lambda$$

so

$Y \sim \text{Exp}(1/\lambda)$

This is actually a special case of a general method:

let X be a continuous r.v. with cdf F. Let F^{-1} be the **generalized** inverse of F, that is

$F^{-1}(y) = \inf\{x : F(x) \geq y\}$

Note that if F is strictly increasing the generalized inverse is just the regular inverse, and that

$F(F^{-1}(x)) = x$

Now say we want to generate a r.v. X with cdf F. Let $U \sim U[0,1]$, then $X = F^{-1}(U) \sim F$ because

$$F_X(x) = P(X \le x) = P(F^{-1}(U) \le x) =$$
$$P(F(F^{-1}(U)) \le F(x)) = P(U \le F(x)) = F(x)$$

Unfortunately the exponential is just about the only application of this method because it is one of the few r.v's with an explicit formula for the cdf.

It is however possible to write a general routine to generate data from a continuous univariate distribution using this method as follows:

say we want to generate data from a density f(x) on a finite interval [A, B]. First we need to find the cdf F, that is

$$F(x) = \int_A^x f(t)dt$$

because this can not (in general) be done analytically we will find F on a fine grid of points numerically. We could use the R function integrate for that:

```
m <- 1000
x <- seq( A, B, length = m)
y <- rep(0, m)
for( i in 1:m)
  y[i] <- integrate(f, A, [i])$value
```

alternatively (and much faster) we can use our own numerical integration routine:

```
y <- f(x)
F <- (x[2]-x[1])/6*cumsum((y[-1]+4*y[-2]+y[-3]))
```

which uses *Simpon's rule.*

if f is not a proper density, that is if $\int_A^B f(t)dt \neq 1$, we can normalize it now very easily :

F = F/F(m)

If we need to evaluate F at an intermediate point we can use the R function approximate:

approx(x, F, xout = ...)$value

but to get the inverse function all we have to do is exchange x and F:

```
approx(F, x, xout = ...)$value
```

and finally the generation of a random variate is done with

```
approx(F, x, xout = runif(1))$value
```

All of this is done in the routine **rPIT**:

```r
rPIT <- function (n, fun, A, B, New = TRUE) {
  if (New) {
      m <- min(2 * n, 1000)
      x <- seq(A, B, length = m)
      y <- fun(x)
      z <- (x[2]-x[1])/6*cumsum((y[-1]+4*y[-2]+y[-3]))
      z <- z/max(z)
      y <- c(0, z)
      xyTmp <- cbind(x, y)
      assign("xyTmp", xyTmp, pos = 1)
  }
  approx(xyTmp[, 2], xyTmp[, 1], runif(n))$y
}
```

This routine has a lot of over-head, to generate just one variate we need to do 1000 function evaluations. On the other hand once we have found the F values we can store them, and from now on we all we need is the last line of the routine, so we get one variate for each call to runif!

The exponential has a relationship with some of the other r.v.s we have discussed and this can be used to generate some of them. For example

$$Y = -2 \cdot \sum_{i=1}^{n} \log(U_i) \sim \chi^2(2n)$$

$$Y = -\beta \cdot \sum_{i=1}^{n} \log(U_i) \sim \Gamma(n, \beta)$$

$$Y = \frac{\sum_{i=1}^{n} \log(U_i)}{\sum_{i=1}^{n+m} \log(U_i)} \sim \text{beta}(n, m)$$

3.3.2 Binomial Distribution

Say we want to generate $X \sim Bin(n, p)$. Now we know that if

$Y_1,..,Y_n$ are iid Ber(p)

then

$Y_1+..+Y_n \sim B(n, p)$

so let

U$_i \sim U[0,1]$, Y$_i$ = I$_{(0,p)}$(U$_i$) and X = Y$_1$ +..+ Y$_n$, then
$X \sim B(n,p)$

3.3.3 Normal Distribution (Box-Muller algorithm)

Say U$_1$ and U$_2$ are iid U[0,1] and set

$$R = \sqrt{-2\log(U_1)} \text{ and } \theta = 2\pi U_2$$
$$X = R\cos\theta \text{ and } Y = R\sin\theta$$

then X and Y are independent standard normal r.v.s

$$\text{let } x = g_1(u,v) = \sqrt{-2\log(u)}\,\cos(2\pi v) \text{ and } y = g_1(u,v) = \sqrt{-2\log(u)}\,\cos(2\pi v)$$
$$\text{now } x^2 + y^2 = -2\log(u)\cdot(\cos^2 2\pi v + \sin^2 2\pi v) = -2\log(u)$$
$$\text{and so } u = h_1(x,y) = \exp\left(-\tfrac{1}{2}(x^2+y^2)\right)$$
$$\text{also } y/x = \frac{\sin(2\pi v)}{\cos(2\pi v)} = \tan(2\pi v)$$
$$\text{and so } v = h_2(x,y) = \frac{1}{2\pi}\arctan\left(\frac{y}{x}\right)$$

the Jacobian of this transform is:

$$J = \begin{vmatrix} e^{-\frac{1}{2}(x^2+y^2)}(-x) & e^{-\frac{1}{2}(x^2+y^2)}(-y) \\ \frac{1}{2\pi}\frac{-y/x^2}{1+(y/x)^2} & \frac{1}{2\pi}\frac{1/x}{1+(y/x)^2} \end{vmatrix} = \begin{vmatrix} e^{-\frac{1}{2}(x^2+y^2)}(-x) & e^{-\frac{1}{2}(x^2+y^2)}(-y) \\ \frac{1}{2\pi}\frac{-y}{x^2+y^2} & \frac{1}{2\pi}\frac{x}{x^2+y^2} \end{vmatrix} =$$
$$e^{-\frac{1}{2}(x^2+y^2)}\frac{1}{2\pi}\frac{1}{1+(y/x)^2}(-x^2-y^2) = -\frac{1}{2\pi}e^{-\frac{1}{2}(x^2+y^2)}$$
$$\text{and so } f_{X,Y}(x,y) = f_{U_1,U_2}(h_1(x,y),h_2(x,y)\cdot|J| = 1\cdot\left|-\frac{1}{2\pi}e^{-\frac{1}{2}(x^2+y^2)}\right| =$$
$$\frac{1}{2\pi}e^{-\frac{1}{2}(x^2+y^2)} = \left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}\right)\cdot\left(\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}y^2}\right)$$

The problem with this algorithm is that it requires the computation of the sin and the cos functions. Here is a similar and much faster algorithm:

1. generate U$_1$ and U$_2$ are iid U[0,1]

2. set $V_1 = 2U_1 - 1$, $V_2 = 2U_2 - 1$ and $S = V_1^2 + V_2^2$

3. If $S>1$, return to step 1
   otherwise set

$$X = \sqrt{\frac{-2\log S}{S}}\, V_1 \text{ and } Y = \sqrt{\frac{-2\log S}{S}}\, V_2$$

then X and Y are iid standard normal. (This is called the polar method)

4 Statistics

4.1 Basic Statistics

A typical situation where we do a simulation is as follows: we have a rv X with probability model

$$P(\cdot|\theta)$$

that is we know the shape of the distribution but not (all of its) parameters. Also weh have $E[X] = \theta$. Now we generate X_1, X_2 .. X_n of these rv's. By the law of large numbers we then have

$$\frac{1}{n}\sum X_i \to \theta$$

What can we say about how good an estimate this is? How large does n have to be to achieve a certain precision?

4.1.1 Sample Mean and Sample Variance

Suppose

$$X_1, X_2, .., X_n \text{ iid } EX_i = \theta \text{ and } VarX_i = \sigma^2$$

let

$$\overline{X} = \frac{1}{n}\sum X_i$$

then

$$E[\overline{X}] = \theta Var[\overline{X}] = \frac{\sigma^2}{n}$$

According to the central limit theorem $\overline{X}$ has an approximately normal distribution, and therefore

$$P\left(|\overline{X} - \theta| > c\frac{\sigma}{\sqrt{n}}\right) \approx 2(1 - \Phi(c))$$

where Φ is the cdf of a standard normal rv.

This tells us how far from the true value our simulation estimate might be. For example: because $\Phi(2) \sim 0.975$, we find that the probability that $\overline{X}$ differs from θ by more than $2\sigma/\sqrt{n}$ is about 0.05.

One problem with using this is that if we don't know θ we almost certainly don't know σ either. So we have to estimate it as well:

$$S^2 = \frac{1}{n-1}\sum\left(X_i - \overline{X}\right)^2$$

Now of course $E[S^2] = \sigma^2$.

Generally this also changes the distribution:

$$\sqrt{n}\left(\overline{X} - \theta\right)/S \sim t_{n-1}$$

but in our situation of simulation n is usually quite large, and then the t distribution is very close to the standard normal.

Often this information is put together in the form of a **confidence interval**. Say our simulation yields X_1,..,X_n, then (if the normal approximation holds)

$$\overline{X} \pm z_{\alpha/2}\frac{S}{\sqrt{n}}$$

is a 100(1-α)% confidence interval for θ. Here z_α is a critical value from the standard normal distribution, that is

$P(Z > z_\alpha) = \alpha$

where $Z \sim N(0,1)$. It is easily found in R with

$z_\alpha = \text{qnorm}(1-\alpha)$

####**Example**

we want to estimate the integral

$$\int_0^1 \exp(-x^2)dx$$

For this generate U_1,..,U_n iid U[0,1]. Then

$$E[\exp(-U^2)] = \int_0^1 \exp(-x^2)dx$$

and so if we set $X_i \ = exp(-U_i^2)$, $\overline{X}$ should converge to the integral.

Now a 90% CI for the integral is given by

$$\overline{X} \pm 1.645\frac{S}{\sqrt{n}}$$

```
f <- function(x) exp(-x^2)
integrate(f, 0, 1)$value
```

```
## [1] 0.7468241
```

```
u <- runif(1e5)
x <- exp(-u^2)
mean(x)
```

```
## [1] 0.7467503
```

```
mean(x) + c(-1, 1)*qnorm(1-0.1)*sd(x)/sqrt(1e5)
```

```
## [1] 0.7459373 0.7475634
```

But what does that mean, "90% CI"? It is as follows: if we did this estimation (or other simulations) over and over again, in the long run 90% of the time the interval would contain the true value, 10% of the time it would not.

In **sb1** we run the simulation 1000 times and check how often the resulting interval contains the true value.

```
## [1] "True Value of I: 0.7468"
## [1] "% of correct CI's for normal based interval: 90.7"
```

One issue is the question whether the central limit theorem actually holds. Again, in a simulation study this is easily verified, just draw a histogram, boxplot or normal probability plot.

4.1.2 Bootstrap Estimate of Standard Error

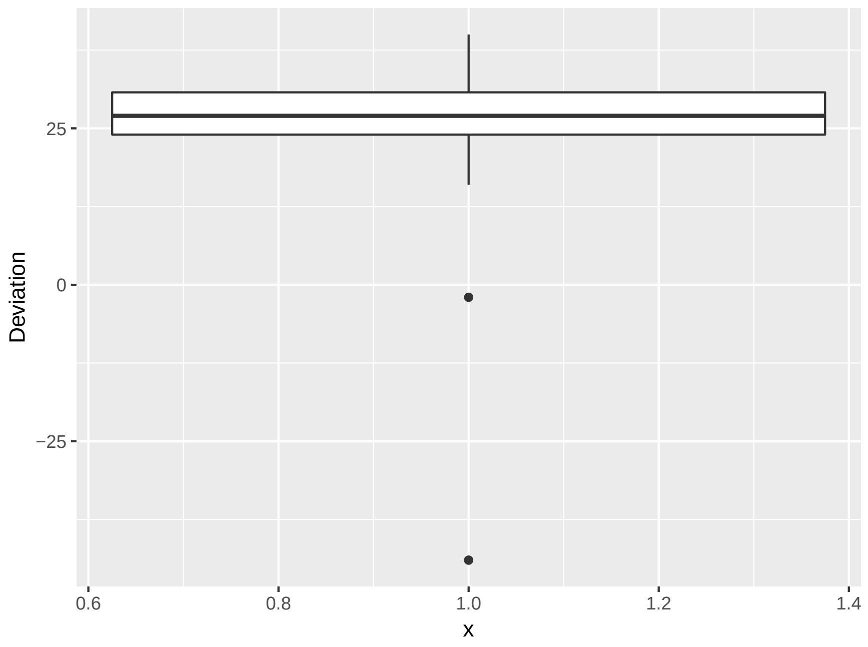
####**Example** Consider Newcomb's measurements of the speed of light. The numbers are the deviations from 24800 nanoseconds:

```
kable.nice(matrix(newcomb$Deviation,
                  ncol=6, byrow = TRUE))
```

28	-44	29	30	24	28
37	32	36	27	26	28
29	26	27	22	23	20
25	25	36	23	31	32
24	27	33	16	24	29
36	21	28	26	27	27
32	25	28	24	40	21
31	32	28	26	30	27
26	24	32	29	34	-2
25	19	36	29	30	22
28	33	39	25	16	23

We want to estimate the mean. However, there are a couple of outliers, and so we use 10% timmed mean which eliminates the lowest and gighest 10% of the data:

```r
ggplot(data=newcomb , aes(1, Deviation)) +
  geom_boxplot()
```



```r
mean(newcomb$Deviation, trim=0.1)
```

```
## [1] 27.42593
```

But how can we get a CI for the median? We don't have a formula for the standard error of the median.

Instead we can use a method called the statistical bootstrap. It starts of very strangely: instead of sampling from a distribution as in a standard MC study, we will now **resample** the data itself, that is if the data is n observations X $_1$, .., X $_n$, then the bootstrap sample is also n numbers **with replacement** from X$_1$, .., X$_n$, that is X^*^$_1$ is any of the original X$_1$, .., X$_n$ with probability 1/n.

In any one **bootstrap sample** an original observation, say X$_1$, may appear once, several times, or not at all.

####**Example** say the data is (5.1, 2.3, 6.4, 7.8, 4.6), then one possible bootstrap sample is (6.4, 4.6, 2.3, 6.4, 5.1)

Say we have a sample X$_1$, .., X$_n$ from some unknown distribution F and we wish to estimate some parameter $\theta = t(F)$. For this we find some estimate $\widehat{\theta}$. How accurate is $\widehat{\theta}$?

####**Example** $X_1, .., X_n \sim F$, $\theta = E(X_1)$ so

$$t(F) = \int xf(x)dx\widehat{\theta} = \overline{X}$$

####**Example** $X_1, .., X_n \sim F$, $\theta = Var(X_1)$ so

$$t(F) = \int (x-\mu)^2 f(x)dx\widehat{\theta} = s^2$$

Here is the algorithm to find the bootstrap estimate of the standard error in $\widehat{\theta}$:

1) Select B independent bootstrap samples $\mathbf{x}^{*1}$, .., $\mathbf{x}^{*B}$, each consisting of n data values drawn with replacement from $\mathbf{x}$. Here B is usually on the order 500.
2) Evaluate the bootstrap replication corresponding to each bootstrap sample, $\widehat{\theta}^*$, b=1,..,B
3) Estimate the standard error $se(\widehat{\theta})$ by the sample standard deviation of the bootstrap replications.

####**Example** say the data is (5.1, 2.3, 6.4, 7.8, 4.6) and we want to estimate the mean μ, then

1) bootstrap sample 1: $\mathbf{x}^{*1} = (6.4, 4.6, 2.3, 6.4, 5.1)$ so $\widehat{\theta}^{*1} = (6.4 + 4.6 + 2.3 + 6.4 + 5.1)/5 = 4.96$
2) bootstrap sample 2: $\mathbf{x}^{*2} = (2.3, 6.4, 4.6, 2.3, 7.8)$ so $\widehat{\theta}^{*2} = (2.3 + 6.4 + 4.6 + 2.3 + 7.8)/5 = 4.68$

and so on.

####**Example** Let's go back to the speed of light. There is a library called *bootstrap* in R to find the bootstrap samples

```r
library(bootstrap)
thetastar <-
  bootstrap(newcomb$Deviation, 2000,
            mean, trim=0.1)$thetastar
```

and a confidence interval can be found with

$$\hat{\theta} \pm z_{\alpha/2} sd(\theta^*)$$

Note there is no $\sqrt{n}$ because sd(thetastar) is already the standard error of the estimate, not the original data.

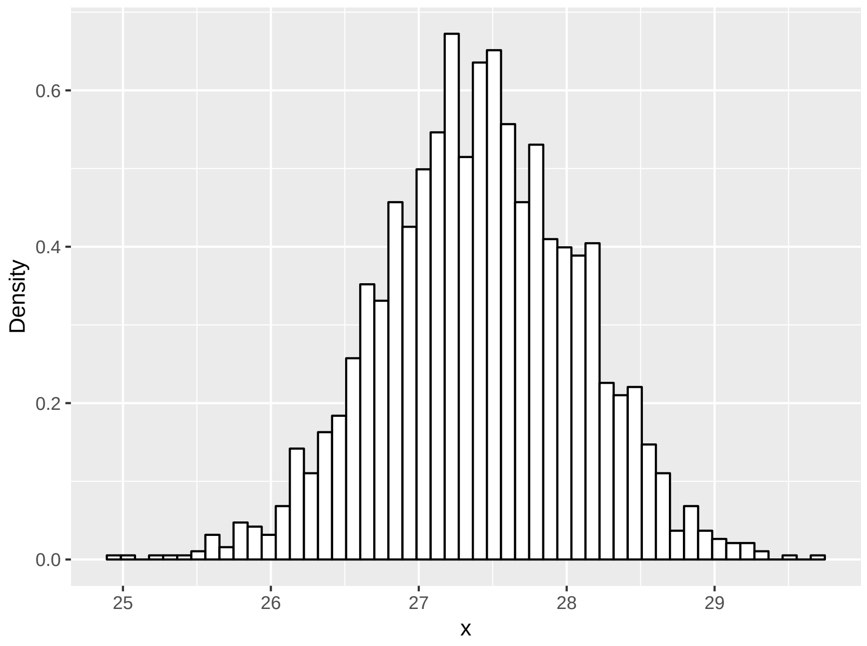
```
cat("Normal Theory Intervals\n")
```

```
## Normal Theory Intervals
```

```
round(mean(newcomb$Deviation, trim=0.1) +
        c(-1, 1)*qnorm(0.975)*sd(thetastar), 2)
```

```
## [1] 26.12 28.73
```

```
bw <- diff(range(thetastar))/50
ggplot(data.frame(x=thetastar), aes(x)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density")
```



The histogram shows that the bootstrap estimates are indeed normal, which is often the case. If they are not we could use a CI based on percentiles as follows:

Using this we find

```
cat("Percentile Bootstrap Intervals\n")
```

```
## Percentile Bootstrap Intervals
```

```
round(quantile(thetastar, c(0.025, 0.975)), 2)
```

```
##  2.5% 97.5%
## 26.11 28.67
```

This idea of the bootstrap is very strange: at first it seems we are getting more out of the data than we should. It is also a fairly new idea, invented by Bradley Efron in the 1980's. Here is some justification why it works:

Let's say we have $X_1, .., X_n$ iid F for some cdf F, and we want to investigate the properties of some parameter θ of F, for example its mean or its median. We have an estimator of θ, say $s(X_1, .., X_n)$, for example

$$s(X_1, .., X_n) = \overline{X}$$

in the case of the mean. What is the error in $s(X_1, .., X_n)$? In the case of the mean this is very easy and we already know that the answer is

$$sd(X)/\sqrt{n}$$

But what if we don't know it and we want to use Monte Carlo simulation to find out. Formally what this means is the following:

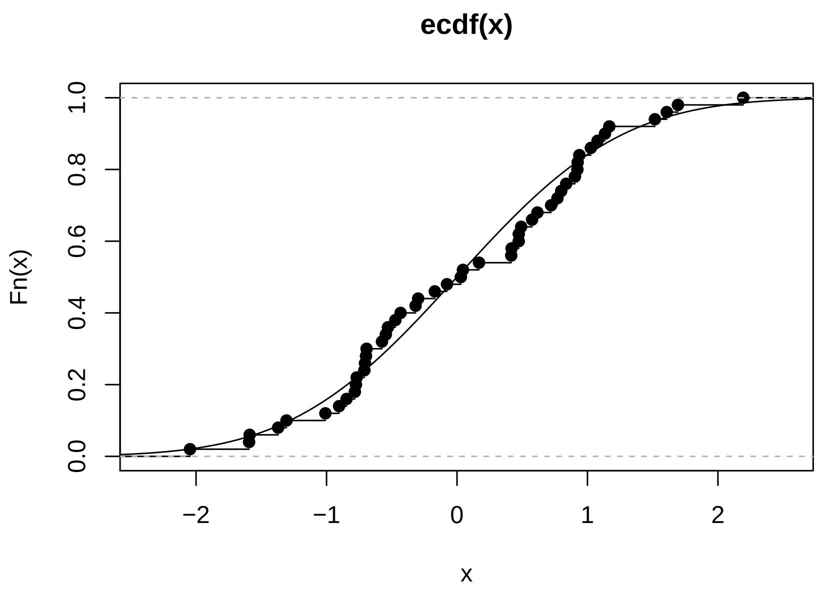
1) generate $X'_1, .., X'_n$ iid F
2) find $\theta_1 = s(X'_1, .., X'_n)$
3) repeat 1 and 2 many times (say 1000 times)
4) Study the MC estimates of θ, for example find their standard deviation.

But what do we do if we don't know that our sample came from F? A simple idea then is to replace sampling from the actual distribution function by sampling from the next best thing, the **empirical distribution function** defined as follows:

$$\hat{F}(x) = \frac{1}{n}\sum I_{[-\infty,x]}(X_i) = \frac{\text{Number of } X_i \le x}{n}$$

Here is an example:

```r
x <- rnorm(50)
plot(ecdf(x))
curve(pnorm, -3, 3, add = TRUE)
```



so the idea of the bootstrap is simple: replace F in the simulation above with Fhat:

1) generate $X_1', .., X_n'$ from $\hat{F}$

2) find $\theta_1 = s(X_1', .., X_n')$

3) repeat 1 and 2 many times (say 1000 times)

4) Study the MC estimates of θ, for example find their standard deviation.

What does it mean, generate data from the empirical distribution function of $\hat{F}$? Actually it means finding a bootstrap sample as described above!

####**Example** Let's illustrate these ideas using an example from a very good book on the bootstrap, "An Introduction to the Bootstrap" by Bradley Efron and Robert Tibshirani. The following table shows the results of a small experiment, in which 7 out of 16 mice were randomly selected to receive a new medical treatment, while the remaining 9 mice were assigned to the control group. The treatment was intended to prolong survival after surgery. The data is the survival times in days:

```
mice
```

```
## $treatment
## [1]  94 197  16  38  99 141  23
##
## $control
## [1]  52 104 146  10  50  31  40  27  46
```

How can we answer the question on whether this new treatment is effective? First of course we can find the within group means and standard deviations:

```r
print(sapply(mice, mean), 4)
```

```
## treatment   control
##     86.86     56.22
```

```r
print(sapply(mice, sd), 4)
```

```
## treatment   control
##     66.77     42.42
```

so we see that the mice who received the treatment lived on average 30.63 days longer. But unfortunately the standard error of the difference is $28.93 = \sqrt{25.24^2 + 14.14^2}$, so we see that the observed difference 30.63 is only $30.63/28.93 = 1.05$ standard deviations above 0.

Let's say next that instead of using the mean we wish to use the median to measure average survival. We find the following:

```r
print(sapply(mice, mean), 4)
```

```
## treatment   control
##     86.86     56.22
```

Now we get a difference in median survival time of 48 days, but what is the standard error of this estimate? Of course there is a formula for the standard error of the median, but it is not simple and just finding it in a textbook would be some work. On the other hand we can use the bootstrap method to find it very easily:

```r
Diff.mean <- matrix(0, 1000, 2)
Diff.median <- matrix(0, 1000, 2)
for (i in 1:1000) {
    x <- sample(mice$treatment, size = 7, replace = T)
    y <- sample(mice$control, size = 9, replace = T)
    Diff.mean[i, ] <- c(mean(x), mean(y))
    Diff.median[i, ] <- c(median(x), median(y))
}
sd.mean <- apply(Diff.mean, 2, sd)
sd.median <- apply(Diff.median, 2, sd)
names(sd.mean) <- c("Treatment", "Control")
names(sd.median) <- c("Treatment", "Control")
print(sd.mean, 4)
```

```
## Treatment   Control
##     22.90     13.25
```

```r
print(sd.median, 4)
```

```
## Treatment   Control
##     37.24     12.49
```

we find that the standard error of the median in the treatment group is about 37, and for the control group it is about 13, so the standard error of the difference is $\sqrt{(37^2 + 13^2)} = 39$, and so 48/39 = 1.2.

This is larger than the one for the mean, but still not statistically significant.

####**Example**
How good is this bootstrap method, that is how well does it calculate the standard error?

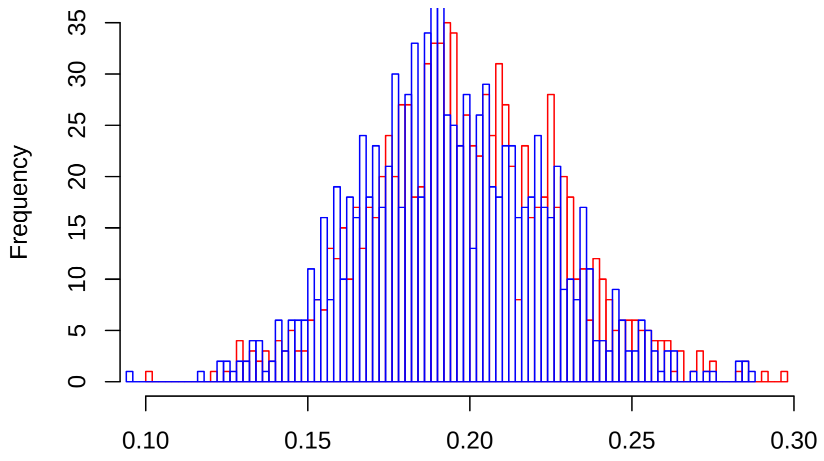
Let's investigate this using a situation where we know the right answer:

Say we have n observations from N(μ,σ), then of course

$$sd(\hat{X}) = \sigma/\sqrt{n}$$

In **bootex** we use both the direct estimate of σ and the bootstrap estimator:

```
bootex(25)
```



```
##                                 Low  High LowBoot HighBoot
## [1,] -0.0519 0.179 0.167 -0.346 0.242  -0.326    0.223
## [2,]  0.1403 0.201 0.194 -0.190 0.470  -0.179    0.459
## [3,]  0.0282 0.203 0.200 -0.305 0.362  -0.300    0.357
## [4,]  0.1052 0.192 0.188 -0.210 0.420  -0.204    0.414
## [5,]  0.1042 0.171 0.171 -0.178 0.386  -0.178    0.386
## [6,]  0.1540 0.197 0.190 -0.169 0.477  -0.159    0.467
```

```
## [1] 89.1 88.1
```

this shows the intervals of first six of 1000 runs and the actual coverage of 90% nominal intervals.

4.2 Verifying the Simulation

Let's say we want to do a simulation study. For this we need to generate $X_1, .., X_n$ from some distribution F. We have written a routine to do that, but how can we be sure that our routine is correct? In this section we will discuss a number methods for verifying that our simulation study is doing the right thing.

4.2.1 Graphical Checks

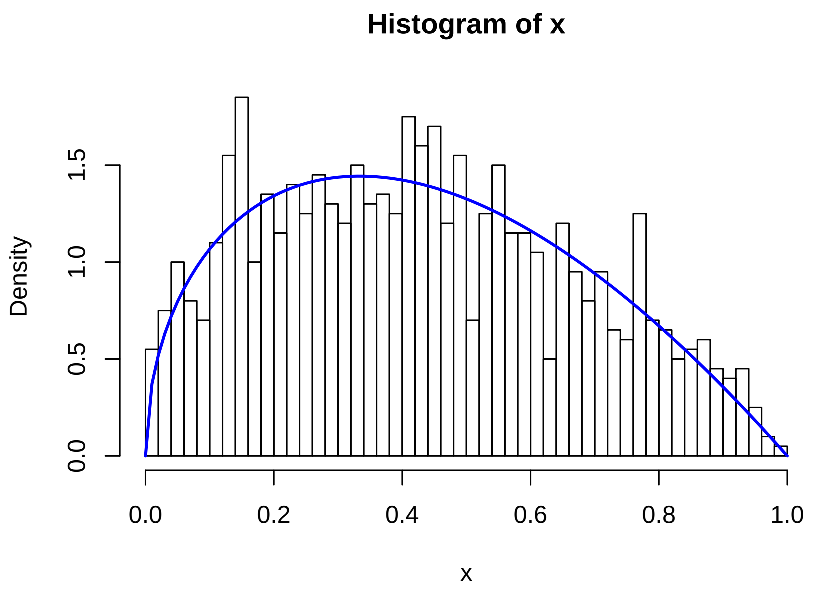
The first idea is to plot a number of graphs, comparing the desired distribution with the simulated data. The first of these is just the histogram:

####**Example** say we want to generate data from the distribution with density

$$f(x) = 3.75(1-x)\sqrt{x},\ 0 < x < 1$$

(That is actually very easy once you realize that F = Beta(3/2,2))

```r
x <- rbeta(1000, 3/2, 2)
hist(x, n = 50, freq = F)
f <- function(x) 3.75*(1-x)*sqrt(x)
curve(f, 0, 1,
      lwd=2, col="blue", add = TRUE)
```



The histogram is a nice graph to check but it is not always clear how good a match we have, especially in the "tails" of the distribution.

Another useful graph is a plot of the empirical cdf vs. the true cdf. Of course for this we need to know the cdf:

```r
sb4 <- function (which = 1, n = 1e4) {
    x <- rbeta(n, 3/2, 2)
    if (which==1) {
        hist(x, n = 100, freq = F)
        z <- seq(0, 1, 0.01)
        lines(z, 3.75*(1-z)*sqrt(z), lwd = 2)
    }
    if (which == 2) {
        x <- sort(x)
        t <- c(0:100)/100
        y <- 3.75 * (2/3 * t^(3/2) - 2/5 * t^(5/2))
        plot(t, y, ylim = c(0, 1), xlab = "x",
             ylab = "", type = "l")
        segments(0, 0, x[1], 0)
        segments(x, c(0:(n - 1))/n, x, c(1:n)/n)
        segments(x[-n], c(1:(n - 1))/n, x[-1],
                 c(1:(n - 1))/n)
        segments(x[n], 1, 1, 1)
    }
    if (which == 3) {
        print("Means")
        print(c(mean(x), 3/7), 3)
        print("Variance")
        print(c(var(x), 3 * 8/49/9), 3)
        print("P(X<0.25)")
        print(c(length(x[x < 0.25])/n,
          3.75*(2/3*0.25^(3/2) - 2/5 * 0.25^(5/2))), 3)
    }

}
```

$$F(t) = \int_{-\infty}^{t} f(x)dx = \int_0^t 3.75(1-x)\sqrt{x}\,dx =$$

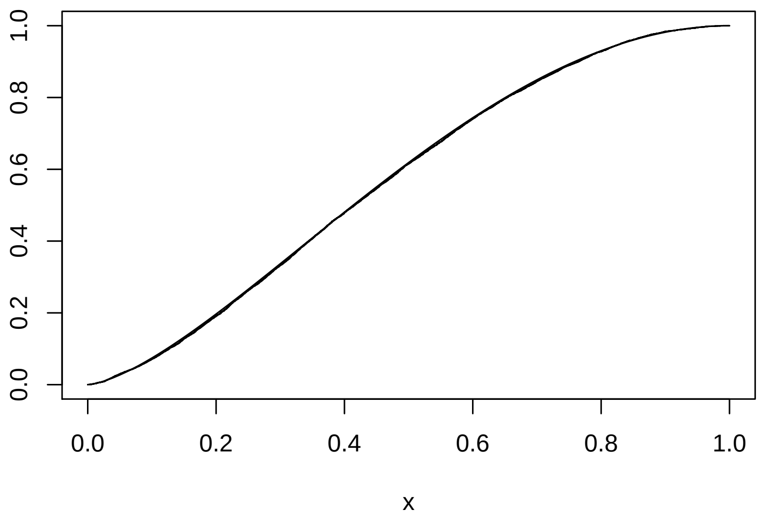
$$3.75\left(\int_0^t x^{1/2}dx - \int_0^t x^{3/2}dx\right) =$$

$$3.75\left(\frac{2}{3}x^{3/2}\Big|_0^t - \frac{2}{5}x^{5/2}\Big|_0^t\right) =$$

$$3.75\left(\frac{2}{3}t^{3/2} - \frac{2}{5}t^{5/2}\right)$$

Now

```r
sb4(2)
```



4.2.2 Numerical Checks

These are the most obvious thing to do for discrete data, just compare the required probabilities with the relative frequencies:

####**Example** Does the R command **sample** actually work? Let's see:

```r
p <- c(1, 2, 5, 1, 2)
x <- sample(1:5, size=1e4, replace=TRUE, prob = p)
print(rbind(table(x)/1e4, p/sum(p)), 3)
```

```
##           1     2     3      4     5
## [1,] 0.0904 0.186 0.456 0.0914 0.176
## [2,] 0.0909 0.182 0.455 0.0909 0.182
```

If the rv takes infinitely many values we might have to combine "low-probability" cases.

If the data is continuous we can still compute some useful numbers:

####**Example** let's again use the example above:

```r
sb4(3)
```

```
## [1] "Means"
## [1] 0.426 0.429
## [1] "Variance"
## [1] 0.0541 0.0544
## [1] "P(X<0.25)"
## [1] 0.269 0.266
```

4.2.3 Formal Tests

Finally we can do actual hypothesis tests, often called *goodness-of-fit tests.* In Statistics we assume that the data was generated by a specific distribution, for example the normal. If we are not sure that such an assumption is justified we would like to test for this.

####**Example** Say we have $X_1, .., X_n$ iid F, and we wish to test $H_0 : F = N(0, 1)$

- **Chisquare Goodness-of-fit Test**

Consider the following result of a simulation study, based on 100 observations:

```
df <- data.frame(x=1:4,
            True=c(0.31, 0.17, 0.05, 0.47),
            Simulation=c(0.3, 0.1, 0.12, 0.48))
kable.nice(df)
```

x	True	Simulation
1	0.31	0.30
2	0.17	0.10
3	0.05	0.12
4	0.47	0.48

Now, is this a good fit? Good enough so we can say our simulation generates the correct data? Obviously if 0.31 is close to 0.3, and so on, we did ok. So what we need is a formula to combine all the info above into one number, a small one if the fit is good and a large one if it is not. There are many ways to do this, the most famous is this one:

$$X^2 = \sum \frac{(O-E)^2}{E}$$

here "O" stands for observed and "E" for expected. The formula uses the actual data, not the frequencies, so O = 31, 17, 5 and 47. The expected are calculated under the null hypothesis, that is assuming the true probabilities hold, so E = np = 30, 10,12, 48. Therefore the test statistic is

$$X^2 = (31-30)^2/30 + (17-10)^2/10 + (5-12)^2/12 + (47-48)^2/48 = 9.04$$

So is 9.04 "large" or "small"? Well, a famous theorem by Wilks says that under some conditions X^2 has a chisquare distribution with k-1 degrees of freedom where k is the number of "categories", here 4. So the p value of the test would be

```
round(1-pchisq(9.04,3), 4)
```

```
## [1] 0.0288
```

and if we use the usual 5% level we would reject the null hypothesis, these simulation does not generate the right data.

####**Example** A famous data set in statistics is the number of deaths from horsekicks in the Prussian army from 1875-1894:

```
kable.nice(horsekicks)
```

Year	Deaths
1875	3
1876	5
1877	7
1878	9
1879	10
1880	18
1881	6
1882	14
1883	11
1884	9
1885	5
1886	11
1887	15
1888	6
1889	11
1890	17
1891	12
1892	15
1893	8
1894	4

It has been hypothesized that this data follows a Poisson distribution. Let's carry out a hypothesis test for this.

First off a Poisson distribution has a parameter, λ. Clearly even if the assumption of a Poisson distribution is correct it will be correct only for some values of λ. What we need to do is to find the the value of λ that minimizes X^2. If we reject the null for that λ, we would also reject it for any other one.

This is called the method of minimum chisquare. Note that it depends on the binning used.

The chisquare goodness-of-fit test is a **large-sample test** , it requires a certain minimal sample size. The usual condition is $E > 5$, although it is known that this is very conservative.

Let's see:

```
table(horsekicks$Deaths)
```

```
##
##  3  4  5  6  7  8  9 10 11 12 14 15 17 18
##  1  1  2  2  1  1  2  1  3  1  1  2  1  1
```

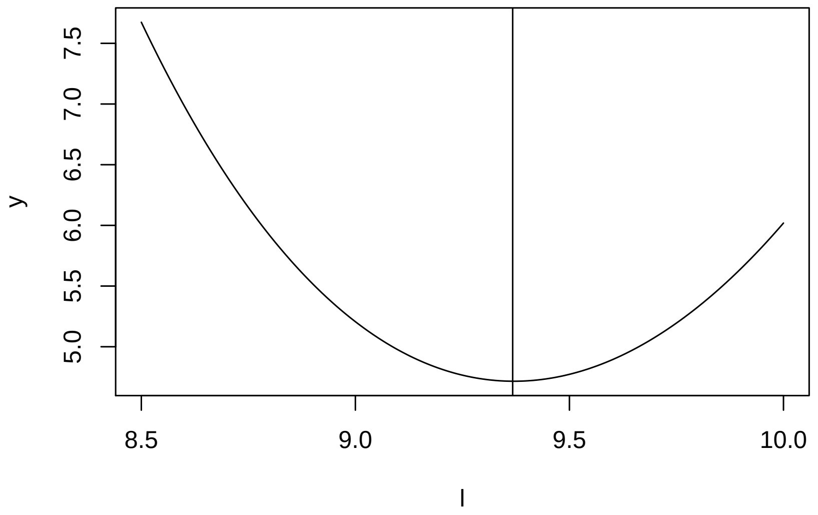
Now these are the observed counts, not the expected, but at least we can get started with this. We will combine cases as follows:

```
df <- data.frame(Bin=c("0-6", "7-9", "10-12", ">12"),
                 Counts=c(6, 4, 5, 5))
kable.nice(df)
```

Bin	Counts
0-6	6
7-9	4
10-12	5
>12	5

Now X^2 as a function of λ is given by

```
x2 <- function(l) {
  bin <- c(0, 6, 9, 12, 50)
  E <- 20*diff(ppois(bin, l))
  O <- c(6, 4, 5, 5)
  sum((O-E)^2/E)
}
l <- seq(8.5, 10, length=250)
y <- 0*l
for(i in seq_along(l))
  y[i] <- x2(l[i])
l0 <- l[which.min(y)]
plot(l, y, type="l")
abline(v=l0)
```



```r
round(l0, 2)
```

```
## [1] 9.37
```

so 9.37 is the value we should use.

A generalization of the theorem above says that under the null hypothesis the X^2 statistic has a χ^2 distribution with k-m-1 degrees of freedom, where k is the number of classes and m is the number of parameters estimated from the data. So here we have k-m-1 = 4-1-1 = 2 d.f, and we find a p-value

```r
round(1-pchisq(x2(9.37), 2), 3)
```

```
## [1] 0.095
```

indicating that the data might well come from a Poisson distribution.

Does it matter what method of estimation is used for the parameter? The answer is yes, and it has to be *minimum chisquare*, that is minimizing the chi square statistic. Note that in general this is NOT the same as maximum likelihood.

In the binning we have used, some E are a bit small. We could of course bin even further, but then we also lose even more information. Instead we can use simulation to find the p value.

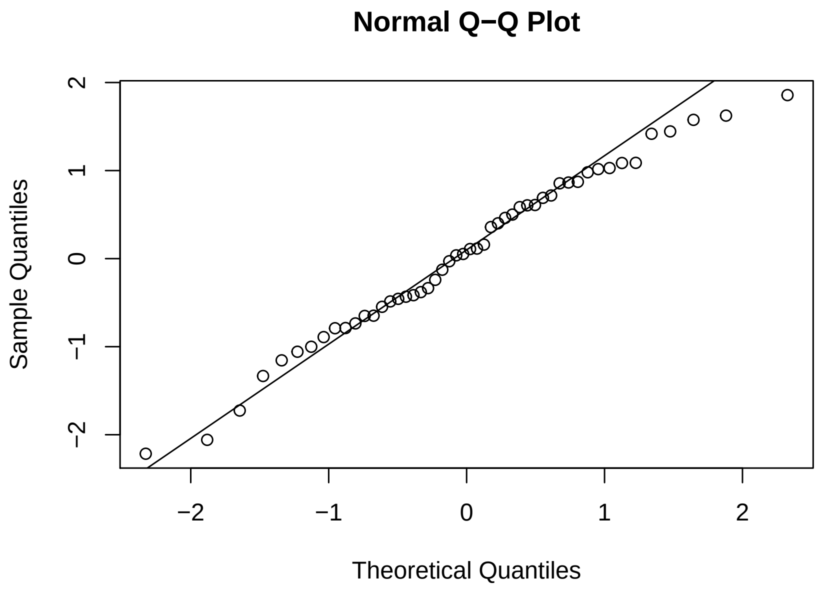
####**Example** Say we have a data set and we want to test whether is comes from a normal distribution. In order to use the chi^2 test we first need to bin the data. There are two basic strategies:

a) Use equal size bins (with the exception of the first and the last)
b) Use adaptive bins chosen so that each bin has roughly the same number of observations.

Testing for normality is a very important problem, although because of simulation not quite as important today as it used to be. There are a number of tests available for this problem, most of them much better (that is with higher power) than the chisquare test. Look for example for the Shapiro-Wilks test and the Anderson-Darling test.

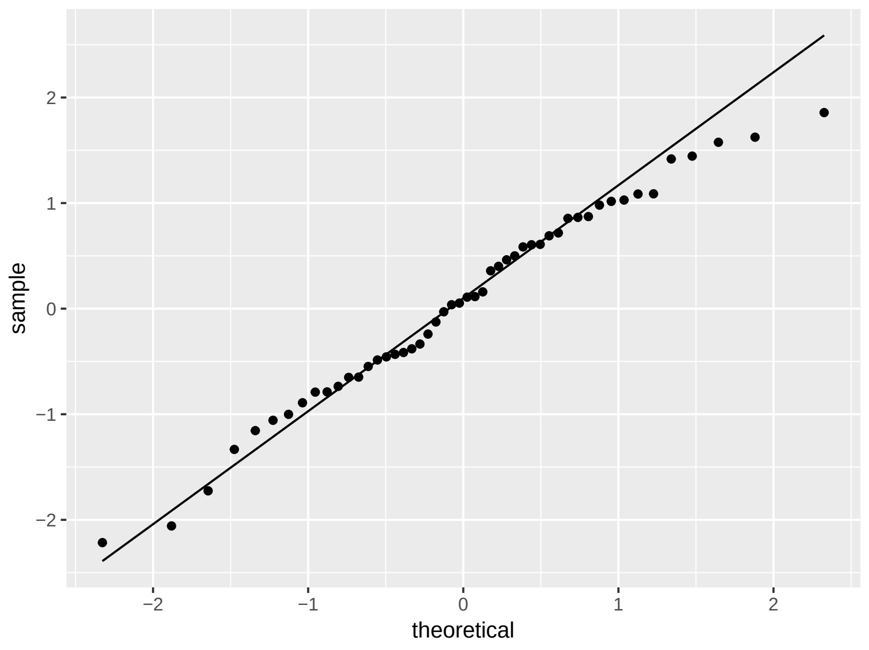
A very good way to assess the distribution of a sample (such as normality) is to draw a graph specifically designed for this purpose, the probability plot. It plots the sample quantiles vs. the quantiles of the hypothesized distribution. If the data follows that distribution the resulting plot should be linear. In R we have the routine qqplot and for the normal distribution especially we have **qqnorm**. **qqline** adds a line that passes the the first and third quartiles to help with reading the graph.

```
x <- rnorm(50)
qqnorm(x)
qqline(x)
```



Or using ggplot2:

```
df <- data.frame(x=x)
ggplot(data=df, aes(sample=x)) +
          geom_qq() + geom_qq_line()
```



4.2.4 Kolmogorov-Smirnov Goodness-of-Fit Test

Say we have $X_1, .., X_n$ which are continuous and independent r.v. and we wish to test $H_0 : X_i \sim F$ for all i. To check this we draw the graph of the empirical vs. the hypothesized cdf. But how close do these have to match each other to decide we have a good fit?

One idea is to look for the largest difference between the two curves. This is exactly what the next, the Kolmogorov-Smirnov test, does. It uses the test statistic

$$D = \max \{|F_n(x) - F(x)|\}$$

At first glance it appears that computing D is hard: it requires finding a maximum of a function which is not differentiable. But inspection of the graphs (and a little calculation) shows that the maximum has to occur at one of the jump points, which in turn happen at the observations. So all we need to do is find $F_n(X_i) - F(X_i)$ for all i.

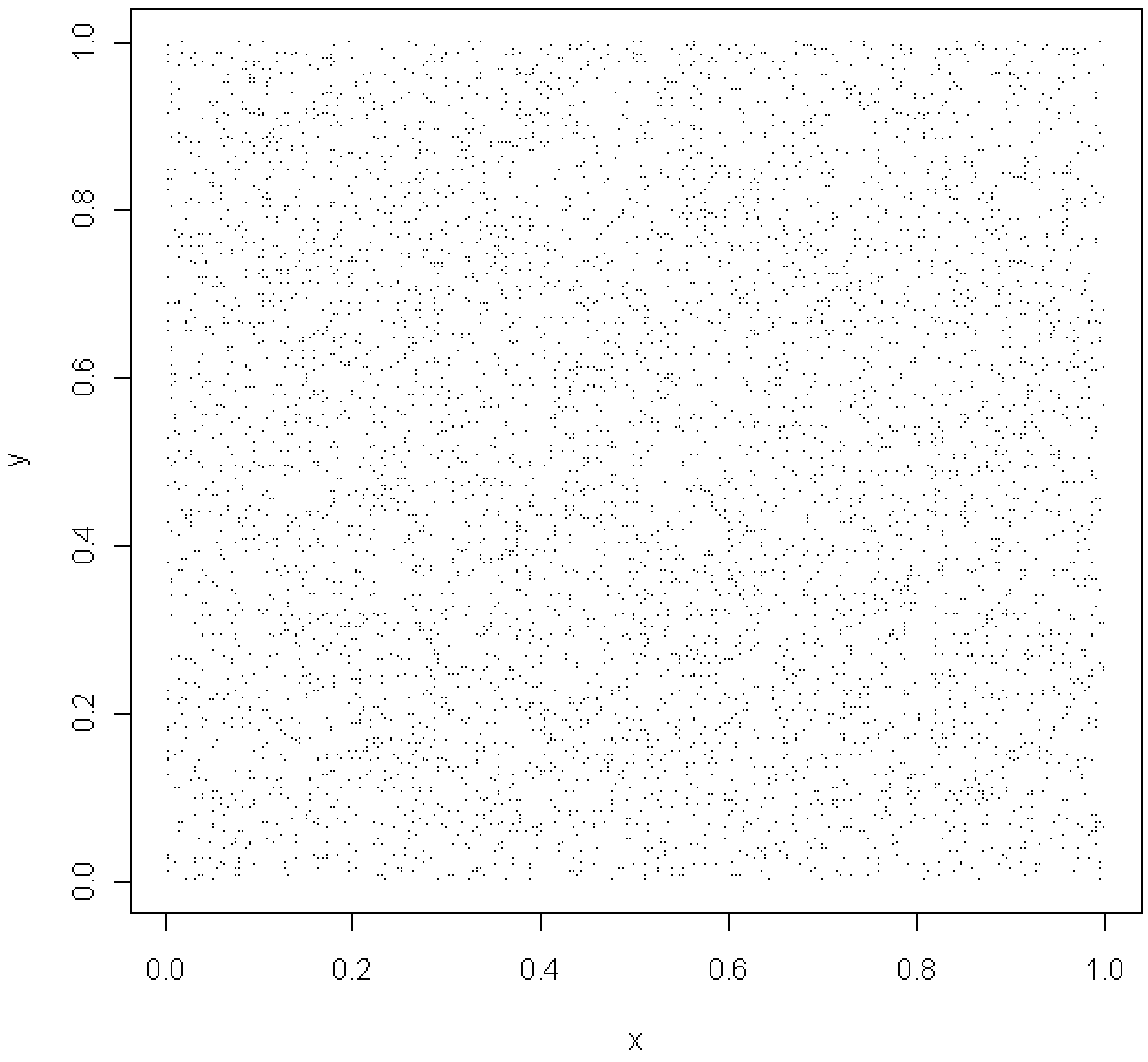
The method is implemented in R in the routine **ks.test** where x is the data set and y specifies the null hypothesis, For example y="pnorm" tests for the normal distribution. Parameters can be given as well. For example ks.test(x,"pnorm",5,2) tests whether X~N(5,2).

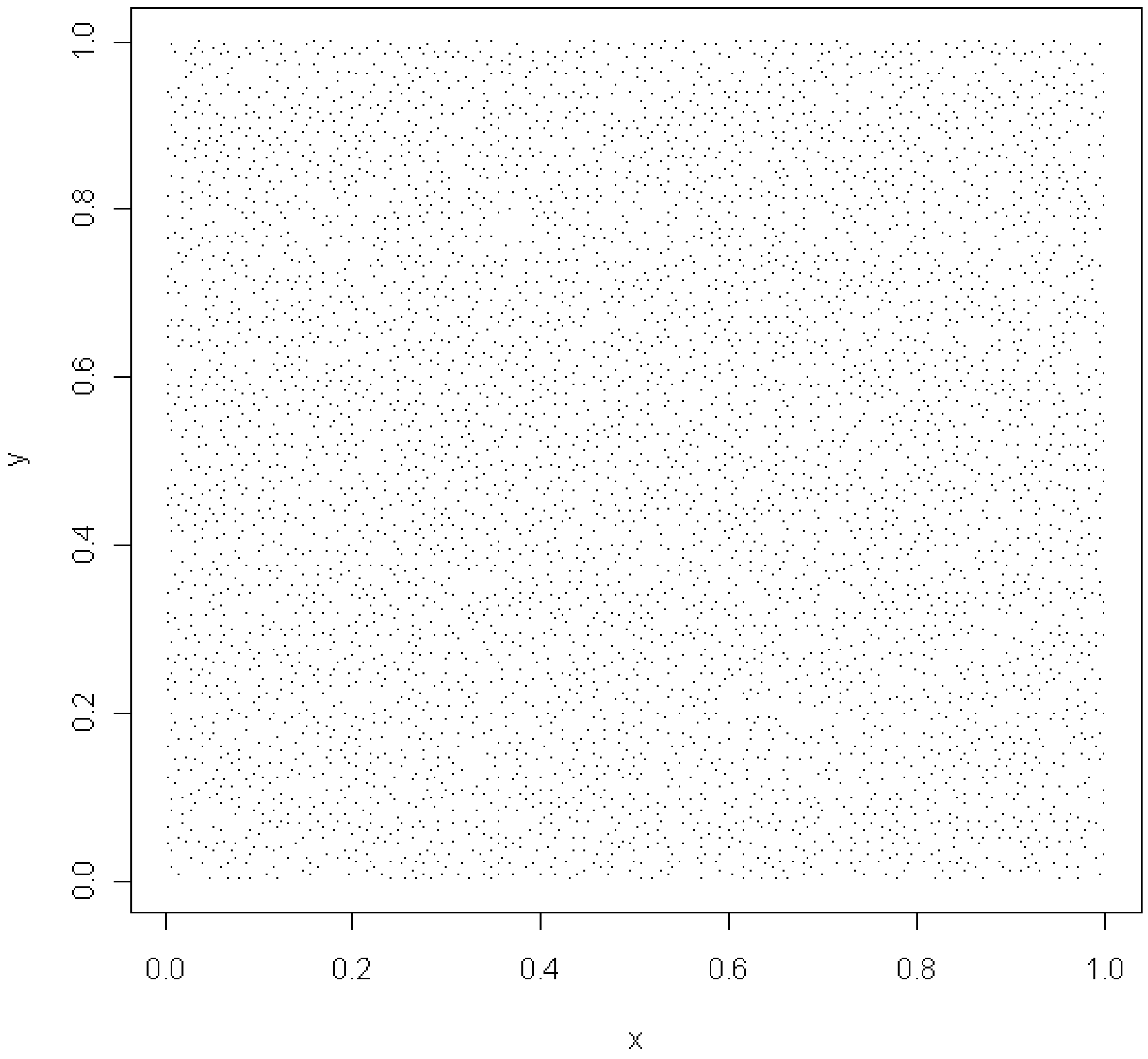
Note that this implementation does **not** allow us to estimate parameters from the data. Versions of this test which allow such estimation are known for some of the standard distributions, but are not part of basic R. We can of course use simulation to implement such tests.

It is generally recognized that the Kolmogorov-Smirnov test is much better than the Chisquare test.

4.2.5 Human Perception of Randomness

####**Example** Consider the following two scatterplots. One of them was done by randomly picking points, the other is not quite so random. Which is which?





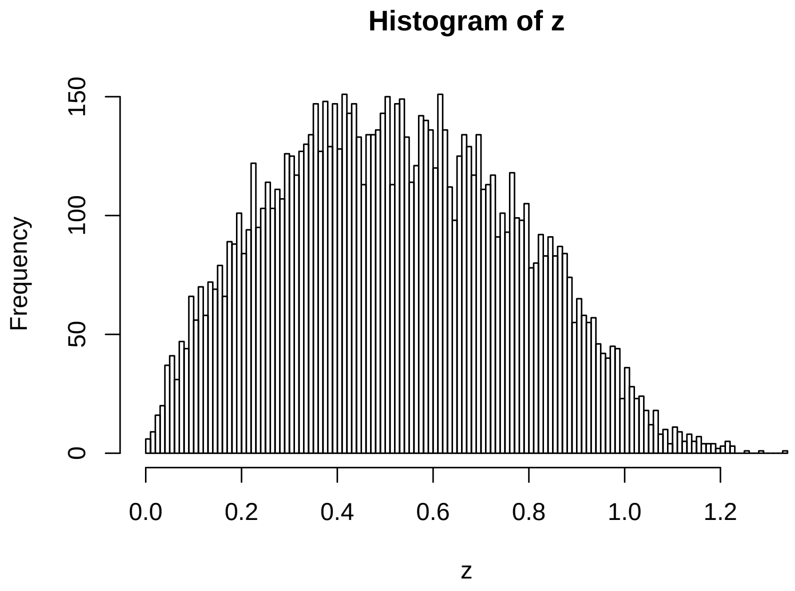
####**Example** A famous statistician used to do the following exercise with his classes: he had a bag with slips of paper. On each paper was either the word "Real" of "Fake". He let each student pick a paper without him seeing it. Then he told them that tonight at home they should read it. If their paper said "Real" they should get a coin, flip it 100 times and write down the sequence of heads and tails. If it said "Fake" they should just make up a sequence but try to make it as real as they could. The next day in class he went around, looked at each students sequence and guessed whether it was "Real" or "Fake". He got it right most of the time. How did he do that?

He looked at the longest "run" that is the longest sequence of either Heads or Tails. There should be at least one run of length 6 or longer, with probability about 80%. Most students who fake it don't put runs anywhere near that long, feeling that this does not look random.

####**Example** Say we randomly select two points in the unit square. Is their distance also uniformly distributed?

The answer is no:

```r
n <- 10000
x1 <- runif(n)
y1 <- runif(n)
x2 <- runif(n)
y2 <- runif(n)
z <- sqrt((x1 - x2)^2 + (y1 - y2)^2)
hist(z, breaks = 100)
```



4.2.6 Higher Dimensional Distributions

As we said earlier, everything gets much more difficult in higher dimensional space. The main problem is that the marginal distributions do not determine the joint distribution, except in the case of independence. As a first test, we can apply the ideas discussed above to the marginals, but if these pass the test we still can't be sure that our simulation works.

As for formal tests, most of them do not generalize to higher dimensions, and those that seem to do run into the **curse of dimensionality**.

First, recall the following:

points in n-dimensional Euclidean space can be described by their coordinates written as n-tuples $(x_1, .., x_n)$.

A point is in the n-dimensional **hypercube** of side length s if $|x_i| < s/2$ for all i. If s=1 it is called the unit hypercube.

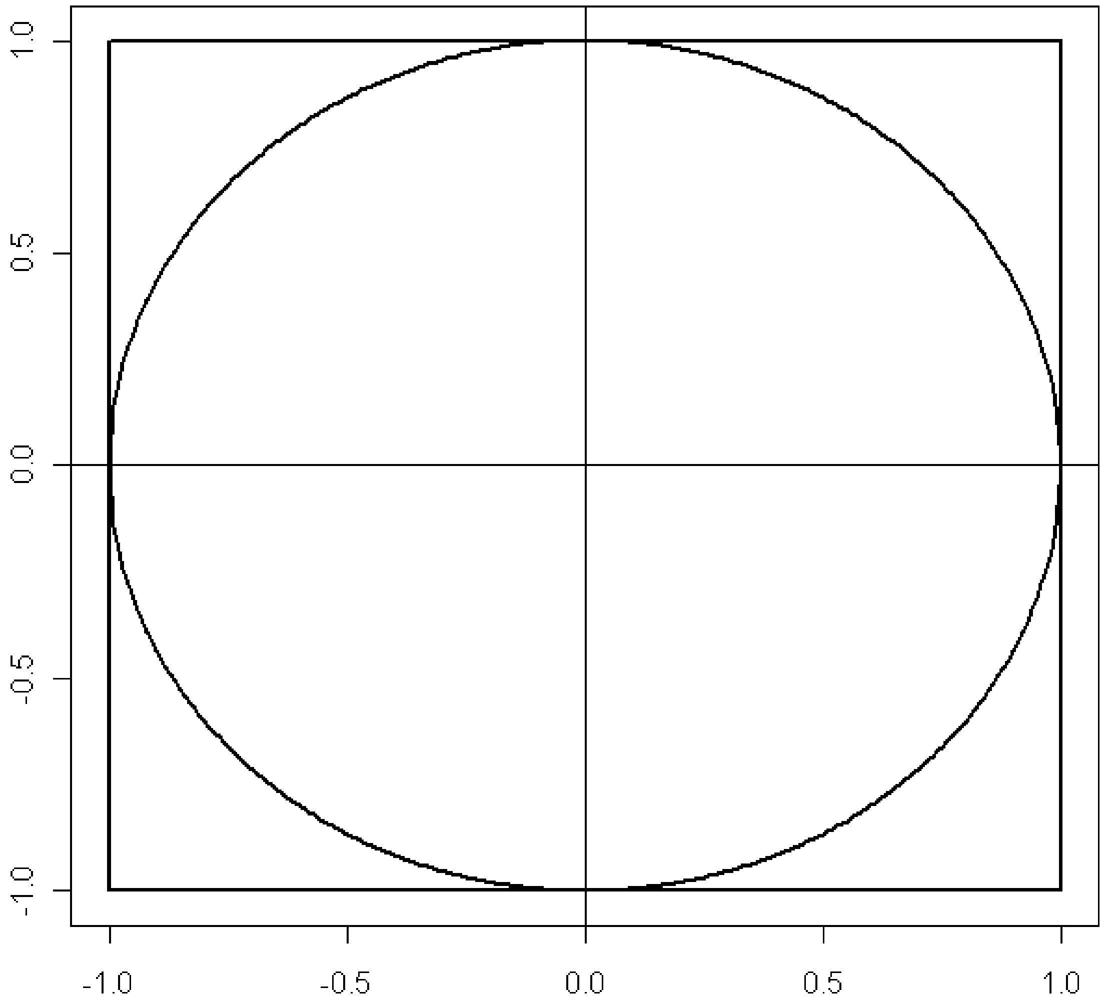
A point is in the n-dimensional **hypersphere** of radius r if

$$\sum x_i^2 < r^2$$

If r=1 it is called the unit hypersphere.

####**Example** What is the volume of the the n-d hypercube? Clearly the answer is s^d, but consider what that means: if $s < 1$ $s^d \to 0$ and if $s > 1$ $s^d \to \infty$ as $d \to \infty$. But the side length is fixed, it is the dimension that goes up!

####**Example** Let's consider the following question: What is the probability that a point generated randomly in the unit square is actually in the unit circle?



In two dimensions we can do this analytically:

$X, Y \sim U[-1,1]$, independent

$$P(\sqrt{X^2+Y^2} \le 1) = P(X^2+Y^2 \le 1) =$$

$$\int_{-1}^{1} P(X^2+Y^2 \le 1 | Y=y) f_Y(y)dy =$$

$$\int_{-1}^{1} P(X^2+y^2 \le 1)\tfrac{1}{2}dy =$$

$$\int_{-1}^{1} P(|X| \le \sqrt{1-y^2})\tfrac{1}{2}dy =$$

$$\int_{-1}^{1} P(-\sqrt{1-y^2} \le X \le \sqrt{1-y^2})\tfrac{1}{2}dy =$$

$$\int_{-1}^{1} \frac{2\sqrt{1-y^2}}{2}\tfrac{1}{2}dy = \int_{0}^{1} \sqrt{1-y^2}\,dy =$$

$$\tfrac{1}{2}(y\sqrt{1-y^2} + \arcsin(y))|_0^1 = \tfrac{1}{2}\arcsin(1) = \tfrac{\pi}{2} \approx 0.7854$$

but in higher dimensions we need simulation:

```
n <- 10000
Probability <- rep(0, 10)
names(Probability) <- 1:10
for (k in 1:10) {
  x <- matrix(runif(n * k, -1, 1), n, k)
  d <- apply(x^2, 1, sum)
  Probability[k] <- round(length(d[d < 1])/n, 3)
}
Probability
```

```
##     1     2     3     4     5     6     7     8     9    10
## 1.000 0.781 0.528 0.305 0.166 0.078 0.039 0.017 0.006 0.003
```

What we find is very strange: if d=10 p=0.003, which seems to say that in 10-dimensional space all randomly chosen points are in the edges!

We can also calculate the ratio of the volumes:

Volume of Hypercube: s^d

Volume of Hypersphere: $\frac{r^d \pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$

$$\frac{\text{Volume of unit Hypersphere}}{\text{Volume of unit Hypercube}} =$$

$$\frac{\frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}}{2^d} = \frac{\pi^{d/2}}{2^d \Gamma(\frac{d}{2}+1)} \to 0 \text{ as } d \to \infty$$

so the the ratio goes to 0 although the volume of the hypercube goes to infinity and the hypersphere touches the hypercube in 2d places!

####**Example** Multivariate normal distribution.

Let's investigate the probability that a randomly chosen observation from a multivariate normal distribution is "out in the tails", that is some given distance from the mean. Let $X^d \sim N(0, I)$ be the "standard" normal in R^d, then the density of X^d is given by

$$f(x) = (2\pi)^{-d/2} exp(-x'x/2)$$

Let $S_0.01$ (x) be the set of all points where the value of the density is 1% of it's highest value, which is of course obtained at the mean. Now

$$\frac{f(x)}{f(0)} = \exp(-x'x/2)$$

so

$$-2\log\frac{f(x)}{f(0)} = x'x = \sum_{i=1}^{d} X_i^2 \sim \chi_d^2$$

so

$$P\left(\frac{f(x)}{f(0)} \le \frac{1}{100}\right) = P\left(\chi_d^2 \ge -2\log\frac{1}{100}\right)$$

and we find

```
d <- c(1, 5*1:5)
p <- round(1 - pchisq(2*log(100), d), 4)
names(p) <- d
p
```

```
##      1      5     10     15     20     25
## 0.0024 0.1010 0.5123 0.8663 0.9803 0.9983
```

so in higher dimensions almost all the observations are out in the tails, almost none are close to the mean!

One consequence of the curse of dimensionality is that the chisquare goodness-of-fit test does not extend past 2 or 3 dimensions:

####**Example** say we have observations X_1 ,..,X_n in R^d and we want to test whether they come from a uniform distribution, that is

$$f(x) = 1 \text{ if } 0 < x_i < 1$$

for i = 1,..,d. First we need to bin the data. Let's say we use the following binning: in each dimension we divide the interval into 10 equal sized bins, 0-1/10, 1/10-2/10,..,9/10-1. What is the probability that a randomly chosen observation from a uniform falls into one of these bins? Because of the uniform the probability is the same for all bins, and we have

$$P(0 < X_i < 1/10, i = 1,..,d) = \prod P(0 < X_i < 1/10) = 1/10^d$$

Recall the requirement for the chisquare is that $E > 5$, so we need at least $n = 5 * 10^d$ observations. For d=3 that means n=5000, for d=5 it means n=500,000!

The required sample size grows exponentially with the dimension.

4.3 A Realistic Example

Note that in the following we use the definition

$$f(x) = 1/\lambda \exp(-x/\lambda)$$

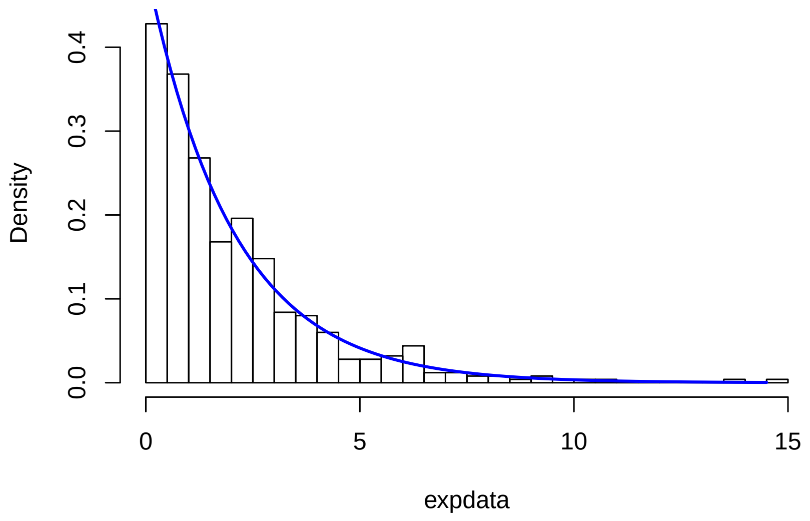
for the exponential. So the best estimator for the rate (the maximum likelihood estimator) is $\overline{X}$.

Say we have the following problem: we have a data set, in **expdata**, which we suspect is from an exponential distribution. How can we make sure of that? First off we need to do a few checks:

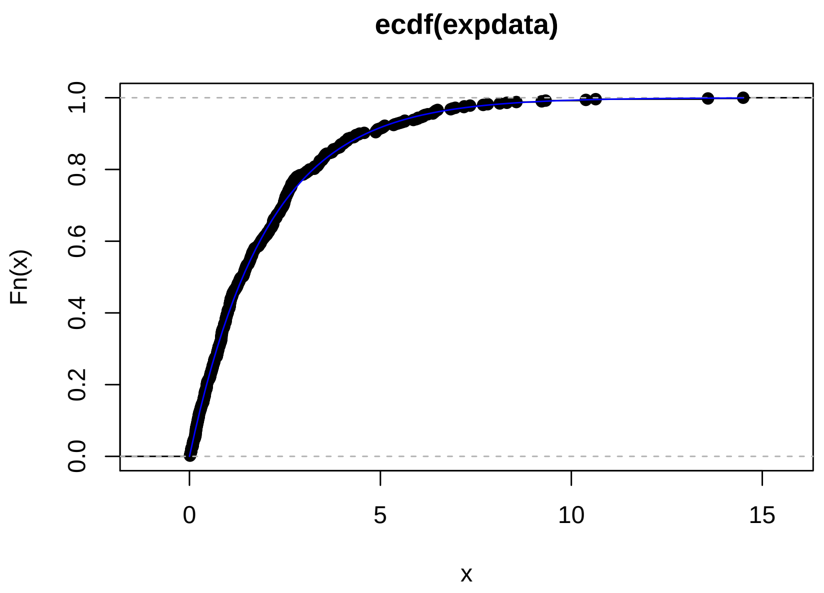
4.3.1 Graphical

First we draw the histogram and the empirical cdf, both with the respective exponential curves, using the sample mean as the rate.

```
n <- length(expdata)
lambdahat <- mean(expdata)
hist(expdata, 25, freq = FALSE, main = "")
f <- function(x) 1/lambdahat*exp(-x/lambdahat)
curve(f, 0, max(expdata),
      lwd=2, col="blue", add = TRUE)
```



```
plot(ecdf(expdata))
F <- function(x) 1-exp(-x/lambdahat)
curve(F, 0, max(expdata),
      lwd=1, col="blue", add = TRUE)
```



The curves appear to be close, but it is difficult to tell just how close.

4.3.2 Testing

Next we carry out some hypothesis tests, with $H_0 : X_i \sim$ Exponential

- Chisquare

The data is continuous, so we need to bin it. There are a number of decisions we need to make, all of them might influence the outcome of the test:

- what type of bins? Let's use adaptive binning (about equally many observations in each bin)
- how many bins? Let's pick 10, not for any good reason at all!

```
nbins <- 10
bins <- c(0, quantile(expdata, prob = (1:(nbins - 1))/nbins), max(expdata) + 1)
O <- hist(expdata, breaks = bins, plot = FALSE)$counts
E <- length(expdata)*diff(F(bins))
chisq <- sum((O - E)^2/E)
out <- round(c(chisq, 1 - pchisq(chisq, nbins - 2)), 4)
names(out) <- c("Chisquare", "p-value")
out
```

```
## Chisquare   p-value
##    5.5197    0.7009
```

- KS test

Here the main problem is that we have to estimate the rate λ, so the calculation of the p-value done by **ks.test** is no good.

Instead we will use simulation as follows:

- Generate x' <- rexp(500, $\overline{X}$)
- find KS statistic for x', using 1/mean(x') as rate
- repeat 1000 times
- p-value is percentage of simulated KS statistics greater than that of data.

```
KS.data <- ks.test(expdata, "pexp", rate = lambdahat)$statistic
KS.sim <- rep(0, 1000)
for(i in 1:1000) {
    x.prime <- rexp(n, lambdahat)
    KS.sim[i] <- ks.test(x.prime, "pexp", rate = 1/mean(x.prime))$statistic
}
out <- round(c(KS.data, length(KS.sim[KS.sim > KS.data])/1000), 4)
names(out) <- c("KS.test", "p-value")
out
```

```
## KS.test p-value
##  0.4905  0.0000
```

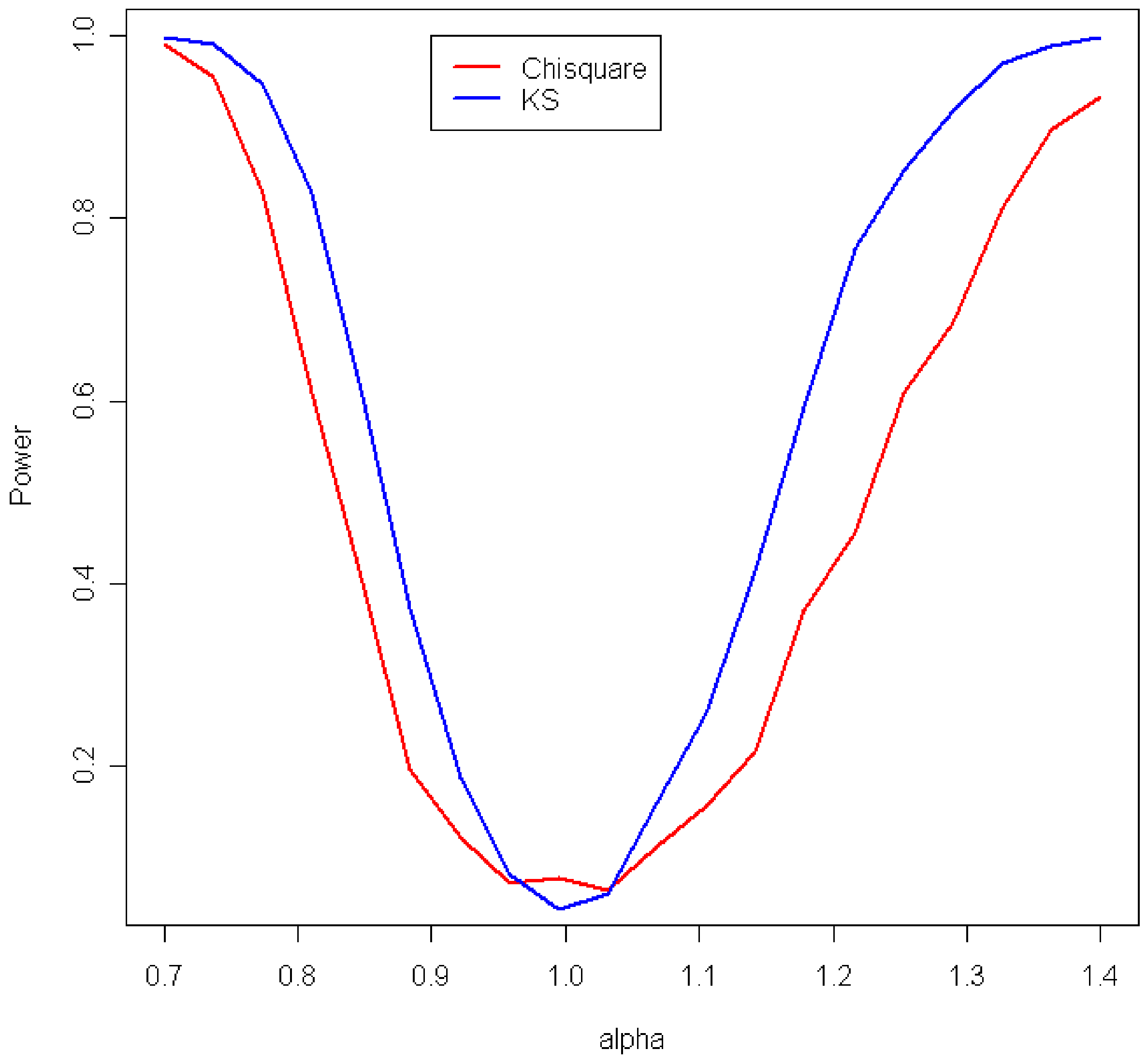
Now we have a problem: the chisquare test says the data may well be from an exponential rv (p-value=0.701) whereas the KS test says no (p-value=0.000). Who do we believe?

4.3.3 Power Study

What we need to do is a study of the power of these tests. That is we need to answer the question: if the data is not from an exponential distribution, how likely are these tests to tell us? The idea is simple: generate data from a rv not exponential, and see what the tests say. In practice, though this is very difficult, there are unaccountably many distributions to pick from. In real live we need to decide which distribution(s) we are most worried about, and test against those. Let's say we worry about the data really being from a gamma distribution, after all, the exponential is a special case of the gamma. So we will do the following:

- generate $X \sim \text{Gamma}(\alpha, \beta)$
- find the p-values of the chisquare and the KS-test for X, just as above
- repeat many times and check the percentage the tests reject the null hypothesis.

Now if $\alpha = 1$, H_0 is true and the powers should be around 0.05, the farther away from 1, the closer to 1 the powers should be. In the next graph we have the power curves:



So we see that the KS test has greater power against a Gamma alternative, so if that is what we are worried about we should indeed reject the null hypothesis.

5 Markov Chain Monte Carlo - MCMC

5.1 Discrete - Time Markov Chains

Often in this course we started an example with "Let $X_1, .., X_n$ be an iid sequence ...". This independence assumption makes a lot of sense in many statistics problems, mainly when the data comes from a random sample. But what happens if the data is not independent?
In that case the most important thing is to describe the dependence structure of that data,

that is to say what the relationship of X_i and X_k is. There many such structures. In this section we will study one of the most common, called a Markov chain.

The sequence of r.v. X_1, X_2, .. is said to be a Markov chain if for any event A we have

$$P(X_n \in A | X_1 = x_1, .., X_{n-1} = x_{n-1}) = P(X_n \in A | X_{n-1} = x_{n-1})$$

that is X_n depends only on X_{n-1} but not on any of the r.v. before it.

####**Example** (Random Walk)

Say we flip a coin repeatedly. Let the random variable Y_i be 1 if the i^th flip is heads, -1 otherwise.

Now let

$$X_n = \sum_{i=1}^{n} Y_i$$

Clearly we have

$$P(X_n = x_n | X_1 = x_1, .., X_{n-1} = x_{n-1}) = \begin{cases} 0.5 \text{ if } x_n = x_{n-1} - 1 \\ 0.5 \text{ if } x_n = x_{n-1} + 1 \\ 0 \text{ otherwise} \end{cases}$$

If we think of the index n as a time variable, then all that matters for the state of the system at time n is where it was at time n-1, but not on how it got to that state.

The random walk above is a discrete-time stochastic process because the time variable n takes discrete values (1,2, etc.) There are also continuous time stochastic processes $\{X(t), t \geq 0\}$.

Our random walk example is also a discrete state space stochastic process because the r.v. X_n can only take countably many values (0, $\pm$1,$\pm$2 etc.). Other stochastic processes might have a continuous state space.

For a Markov chain (that is a stochastic process with a discrete state space) all the relevant (probability) information is contained in the probability to get from state i to state j in k steps. For k=1 this is contained in the transition matrix $P = (p_{ij})$, and in fact as we shall see that P is all we need.

####**Example** (Random Walk, cont)

Here we have $p_{ij} = 1/2$ if $|i - j| = 1$, 0 otherwise

####**Example** (Asymmetric Random Walk)

As above the state space are the integers but now we go from i to i+1 with probability p, to i-1 with probability q and stay at i with probability 1-p-q.

####**Example** (Ehrenfest chain)

Say we have two boxes, box 1 with k balls and box 2 with r-k balls. We pick one of the balls at random and move it to the other box. Let X_n be the number of balls in box 1 at time

n. We have

$$\begin{aligned}
&X_n \in 0, 1, .., r\\
&p_{k,k+1} = (r-k)/r\\
&p_{k,k-1} = k/r\\
&p_{i,j} = 0 \text{ otherwise}
\end{aligned}$$

Ehrenfest used this model to study the exchange of air molecules in two chambers connected by a small hole, so small that only one molecule at a time can pass through.

Say we have a Markov chain $\{X_n\}$, $n = 1, 2, ..$ with transition matrix P. Define the n-step transition matrix $P^{(n)} = (p_{ij}^{(n)})$ by

$$p_{ij}^{(n)} = P(X_n = j|X_1 = i)$$

Of course $P^{(1)} = P$. Now

$$p_{ij}^{(2)} = P(X_{n+2} = j|X_n = i) =$$

$$\sum\nolimits_{k\in S} P(X_{n+2} = j|X_n = i, X_{n+1} = k)P(X_{n+1} = k|X_n = i) =$$

$$\sum\nolimits_{k\in S} P(X_{n+2} = j|X_{n+1} = k)P(X_{n+1} = k|X_n = i) =$$

$$\sum\nolimits_{k\in S} p_{ik}p_{kj} = (PP)_{ij}$$

$$\text{In general } p_{ij}^{(n)} = \left(\prod_{i=1}^{n} P\right)_{ij}$$

####**Example** (Ehrenfest chain)

Let's find the 2-step transition matrix for the Ehrenfest chain with $r = 3$. The transition matrix is given by

$$P = \left(\begin{array}{cc|cccc} & & & j & & \\ & & 0 & 1 & 2 & 3 \\ \hline & 0 & 0 & 1 & 0 & 0 \\ i & 1 & 1/3 & 0 & 2/3 & 0 \\ & 2 & 0 & 2/3 & 0 & 1/3 \\ & 3 & 0 & 0 & 1 & 0 \end{array}\right)$$

and so the 2-step transition matrix is

$$P^{(2)} = P \cdot P = \begin{pmatrix} 1/3 & 0 & 2/3 & 0 \\ 0 & 7/9 & 0 & 2/9 \\ 2/9 & 0 & 7/9 & 0 \\ 0 & 2/3 & 0 & 1/3 \end{pmatrix}$$

In order to find $P^{(n)}$ we could just find PPP..P n-times. With a little linear algebra this becomes easier: For many matrices P there exists a matrix U and a diagonal matrix D such that

$$P = UDU^{-1}$$

Here is how to find U and D:

First we need to find the eigenvalues of the matrix P, that is we need to find the solutions of the equations

$$Px = \lambda x$$

This is equivalent to

$$(P - \lambda I)x = 0$$

or

$$\det(P - \lambda I) = 0$$

So we have:

$$\det(P - \lambda I) = \begin{vmatrix} -\lambda & 1 & 0 & 0 \\ 1/3 & -\lambda & 2/3 & 0 \\ 0 & 2/3 & -\lambda & 1/3 \\ 0 & 0 & 1 & -\lambda \end{vmatrix} =$$

$$-\lambda \cdot \begin{vmatrix} -\lambda & 2/3 & 0 \\ 2/3 & -\lambda & 1/3 \\ 0 & 1 & -\lambda \end{vmatrix} - 1 \cdot \begin{vmatrix} 1/3 & 2/3 & 0 \\ 0 & -\lambda & 1/3 \\ 0 & 1 & -\lambda \end{vmatrix} =$$

$$-\lambda \cdot [-\lambda(\lambda^2 - 1/3) - 2/3(-2/3\lambda) + 0] - [1/3(\lambda^2 - 1/3)] =$$

$$-\lambda[-\lambda^3 + 1/3\lambda + 4/9\lambda] - 1/3\lambda^2 + 1/9 = \lambda^4 - 10/9\lambda^2 + 1/9 = 0$$

$$\lambda^2 = \frac{10/9 \pm \sqrt{(10/9)^2 - 4/9}}{2} = \frac{1}{2}\left(\frac{10}{9} \pm \sqrt{\frac{100 - 4 \cdot 9}{9^2}}\right) = \frac{1}{2}\left(\frac{10 \pm 8}{9}\right) = 1 \text{ or } 1/9$$

so the eigenvalues are

$\lambda_1 = -1$, $\lambda_2 = -1/3$, $\lambda_3 = 1/3$ and $\lambda_4 = 1$

The D above now is the matrix with the eigenvalues on the diagonal.

The columns of the matrix U are the corresponding eigenvectors (with Euclidean length 1), so for example

$$(P-\lambda_1 I)x = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1/3 & 1 & 2/3 & 0 \\ 0 & 2/3 & 1 & 1/3 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = 0$$

$$\begin{aligned} x_1 + x_2 &= 0 \\ 1/3x_1 + x_2 + 2/3x_3 &= 0 \\ 2/3x_2 + x_3 + 1/3x_4 &= 0 \\ x_3 + x_4 &= 0 \end{aligned}$$

Of course we have $\det(P-\lambda I)=0$, so this system is does not have a unique solution. Setting $x_1 = 1$ we can then easily find a solution $x = (1, -1, 1, -1)$.

This vector has Euclidean length $\sqrt{(}1^2+(-1)^2+1^2+(-1)^2) = 2$, so the normalized eigenvector is $x = (1/2, -1/2, 1/2, -1/2)$

Similarly we can find eigenvectors for the other eigenvalues.

Alternatively (and a lot easier!) we can use the R function **eigen** to do the calculations for us!

Why does this help in computing $P^{(n)}$? It turns out that we have

$$P^{(2)} = PP = UDU^{-1}UDU = UDDU^{-1} = UD^2U^{-1}$$

and

$$D^2 = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{pmatrix}^2 = \begin{pmatrix} \lambda_1^2 & 0 & 0 & 0 \\ 0 & \lambda_2^2 & 0 & 0 \\ 0 & & \lambda_3^2 & 0 \\ 0 & 0 & 0 & \lambda_4^2 \end{pmatrix}$$

and in general we have $P^{(n)} = UD^nU^{-1}$.

Let's find the 2-step and 3-step transition matrix of the Ehrenfest chain with $r = 3$:

```r
r <- 3
statespace <- 0:r
pi <- choose(rep(r, r + 1), 0:r)/2^r
P <- matrix(0, r + 1, r + 1)
dimnames(P) <- list(statespace, statespace)
P[1, 2] <- 1
P[r + 1, r] <- 1
for (k in 2:r) {
    P[k, k - 1] <- (k - 1)/r
    P[k, k + 1] <- (r - k + 1)/r
}
print(P, 3)
```

```
##       0     1     2     3
## 0 0.000 1.000 0.000 0.000
## 1 0.333 0.000 0.667 0.000
## 2 0.000 0.667 0.000 0.333
## 3 0.000 0.000 1.000 0.000
```

```r
D <- diag(eigen(P)$values)
U <- eigen(P)$vectors
print(round(U %*% D^2 %*% solve(U), 3))
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] 0.333 0.000 0.667 0.000
## [2,] 0.000 0.778 0.000 0.222
## [3,] 0.222 0.000 0.778 0.000
## [4,] 0.000 0.667 0.000 0.333
```

```r
print(round(U %*% D^3 %*% solve(U), 3))
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] 0.000 0.778 0.000 0.222
## [2,] 0.259 0.000 0.741 0.000
## [3,] 0.000 0.741 0.000 0.259
## [4,] 0.222 0.000 0.778 0.000
```

The routine **ehrenfest** with which=1 computes $P^{(n)}$ for the Ehrenfest chain.

```r
ehrenfest <- function (which = 1, n = 10000, r = 3) {
    statespace <- 0:r
    pi <- choose(rep(r, r + 1), 0:r)/2^r
    P <- matrix(0, r + 1, r + 1)
    dimnames(P) <- list(statespace, statespace)
    P[1, 2] <- 1
    P[r + 1, r] <- 1
    for (k in 2:r) {
        P[k, k - 1] <- (k - 1)/r
        P[k, k + 1] <- (r - k + 1)/r
    }
    D <- diag(eigen(P)$values)
    U <- eigen(P)$vectors
    if (which == 1) {
        return(round(U %*% D^n %*% solve(U), 4))
    }
    X <- rep(-1, n)
    X[1] <- sample(statespace, size = 1, prob = pi)
    for (i in 2:n) {
        if (X[i - 1] == 0) {
            X[i] <- 1
            next
        }
        if (X[i - 1] == r) {
            X[i] <- r - 1
            next
        }
        X[i] <- sample(c(X[i-1]+1, X[i-1]-1), size=1,
            prob = c((r - X[i-1])/r, X[i-1]/r))
    }
    if (which == 4) {
        return(X)
    }
    if (which == 2) {
        return(rbind(pi, table(X)/n))
    }
    if (which == 3) {
        print(c(mean(X), sum(0:r * pi)))
        print(c(mean(X^2), sum((0:r)^2 * pi)))
        print(c(mean(log(X + 1)), sum(log(0:r+1)*pi)))
    }

}
ehrenfest(1, 1)
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,] 0.0000 1.0000 0.0000 0.0000
## [2,] 0.3333 0.0000 0.6667 0.0000
## [3,] 0.0000 0.6667 0.0000 0.3333
## [4,] 0.0000 0.0000 1.0000 0.0000
```

```
ehrenfest(1, 2)
```

```
##        [,1]   [,2]   [,3]   [,4]
## [1,] 0.3333 0.0000 0.6667 0.0000
## [2,] 0.0000 0.7778 0.0000 0.2222
## [3,] 0.2222 0.0000 0.7778 0.0000
## [4,] 0.0000 0.6667 0.0000 0.3333
```

```
ehrenfest(1, 3)
```

```
##          [,1]   [,2]   [,3]   [,4]
## [1,] 0.0000 0.7778 0.0000 0.2222
## [2,] 0.2593 0.0000 0.7407 0.0000
## [3,] 0.0000 0.7407 0.0000 0.2593
## [4,] 0.2222 0.0000 0.7778 0.0000
```

5.1.1 Stationary Distribution

Consider again the Ehrenfest chain, and compute $P^{(n)}$ for $n \to \infty$:

```
print(round(U %*% D^10 %*% solve(U), 3))
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.25 0.00 0.75 0.00
## [2,] 0.00 0.75 0.00 0.25
## [3,] 0.25 0.00 0.75 0.00
## [4,] 0.00 0.75 0.00 0.25
```

```
print(round(U %*% D^100 %*% solve(U), 3))
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.25 0.00 0.75 0.00
## [2,] 0.00 0.75 0.00 0.25
## [3,] 0.25 0.00 0.75 0.00
## [4,] 0.00 0.75 0.00 0.25
```

You notice that $P^{(n)}$ seems to converge to a limit. We will now study this limit.

Let S be the state space of a Markov Chain X with transition matrix P.

Let π be a "measure" on S. Then π is called a stationary measure of X if $\pi^T P = \pi^T$.

We won't discuss exactly what it means for π to be a "measure". You can think of it in the same way as a probability distribution, only that we don't have $\sum \pi_i = 1$.

Note:

$$\pi^T P = \pi^T (P^T \pi)^T = \pi^T P^T \pi = \pi(P^T - I)\pi = 0$$

so again the system of equations is singular.

####**Example** (Ehrenfest chain) To find a (?) stationary measure we have to solve the system of equations

$$p_{ij} = \sum_i \pi_i P_{ij} \; i = 0, 1.., r$$

often we can get unique solution by requiring that π be a proper probability distribution, that is that $\sum \pi_i = 1$.

Here this means the system

$$\begin{aligned}\pi_0 &= 1/3\pi_1 \\ \pi_1 &= \pi_0 + 2/3\pi_2 \\ \pi_2 &= 2/3\pi_1 + \pi_3 \\ \pi_3 &= 1/3\pi_2 \\ \pi_1 + \pi_2 + \pi_3 &= 1\end{aligned}$$

which has the solution

$$\pi = (1/8, 3/8, 3/8, 1/8)$$

In the general case we find the stationary measure

$$\pi_i = \binom{r}{j}/2^j$$

$i = 0, .., r.$

The interpretation is the following: Say we choose the initial state X_0 according to π, that is $P(X_0 = i) = \pi_i$. Then π_i is the long-run proportion of time the chain spends at state i, that is

$$\pi_i = \lim_{N\to\infty} \sum_{k=1}^{N} I[X_n = i]/N$$

####**Example** Let's illustrate this for the Ehrenfest chain with r=3:

```
r <- 3
statespace <- 0:r
pi <- choose(rep(r, r + 1), 0:r)/2^r
n <- 1000
X <- rep(-1, n)
X[1] <- sample(statespace, size = 1, prob = pi)
for (i in 2:n) {
    if (X[i - 1] == 0) {
        X[i] <- 1
        next
    }
    if (X[i - 1] == r) {
        X[i] <- r - 1
        next
    }
    X[i] <- sample(c(X[i - 1] + 1, X[i - 1] - 1), size = 1, prob = c((r - X[i - 1])/r,
}
print(rbind(pi, sim=table(X)/n), 3)
```

```
##         0     1     2     3
## pi  0.125 0.375 0.375 0.125
## sim 0.124 0.362 0.376 0.138
```

####**Example** (Random Walk) Let S be the integers and define a Markov chain by $p_{i,i+1} = p$ and $p_{i,i-1} = q = 1 - p$. A stationary measure is given by $\pi_i = 1$ for all i because $(\pi P)_i = 1p + 1q = 1$.

Now assume $p \neq q$ and define $pi_i = (p/q)^i$. Then

$$\sum_i \pi_i P_{ij} = \pi_{j+1} P_{j+1,j} + \pi_{j-1} P_{j-1,j} =$$
$$\left(\frac{p}{q}\right)^{j+1} q + \left(\frac{p}{q}\right)^{j-1} p = \left(\frac{p}{q}\right)^{j} (p+q) = \left(\frac{p}{q}\right)^{j} = \pi_j$$

Note that this shows that stationary measure are not unique.

One use of the stationary distribution is an extension of the WLLN to Markov chains. That is, say h is a function on the state space, then

$$\lim_{n\rightarrow\infty} \frac{1}{n} \sum_{i=1}^{n} h(X_i) = Eh(Z)$$

where Z is a r.v. with density π.

This is illustrated in **ehrenfest** with which=3 for $h(x) = x$, $h(x) = x^2$ and $h(x) = \log(x + 1)$

```
ehrenfest(3)
```

```
## [1] 1.4988 1.5000
## [1] 3.0016 3.0000
## [1] 0.8441320 0.8451966
```

One of the main results for Markov chains is the following:

If the Markov chain $\{X_n\}$ is irreducible and ergodic, then

$$\lim_{n\rightarrow\infty} P(X_n = j) = \pi_j$$

####**Example** Of course this result does not apply to the Ehrenfest chain, which is not aperiodic, but the result holds anyway as we have seen.

####**Example** consider the following Markov chain: if at i it next moves to i+1 with probability p or to 1 with probability 1-p. Let's find its stationary distribution:

$$p_{i,i+1} = p = 1 - p_{i,1}$$

$$P = \begin{pmatrix} q & p & 0 & 0 & .. \\ q & 0 & p & 0 & .. \\ q & 0 & 0 & p & .. \\ q & 0 & 0 & 0 & p \\ q & 0 & 0 & 0 & .. \end{pmatrix}$$

$$\pi^T P_1 = \sum \pi_i q = q = \pi_1$$

$$\pi^T P_k = \pi_{k-1} p = \pi_k$$

$$\pi_k = \pi_{k-1} p = \pi_{k-2} p^2 = .. =$$

$$\pi_0 p^{k-1} = q p^{k-1}$$

so the stationary distribution is a geometric!

5.2 MCMC - Markov Chain Monte Carlo

The starting point of this section is the result that if the Markov chain $\{X_n\}$ is irreducible and ergodic, then

$$\lim_{n\to\infty} P(X_n = j) = \pi_j$$

The idea is to use this as follows: say I want to generate data from a distribution π. Now if I can find an irreducible and ergodic Markov chain $\{X_n\}$ which has π as its stationary measure, we can generate observations from $\{X_n\}$, wait a while until its limiting distribution is reached (?) and then take the $\{X_n\}$ as if they came from π.

5.2.1 The Metropolis-Hastings Algorithm

Let's say we want to generate observations from a distribution π on {1,..,m}. Let Q be the transition probability matrix of an irreducible Markov chain on {1,..,m}. Define the Markov chain $\{X_n\}$ as follows:

When $X_n = i$ a r.v. X with $P(X = j) = q_{ij}$ is generated. (This of course means we need to know how to generate observations from Q). If $X = j$, then set $X_{n+1} = j$ with probability α_{ij} and equal to i with probability $1 - \alpha_{ij}$. Now $\{X_n\}$ is a Markov chain with transition probabilities given by:

$$\begin{aligned} p_{ij} &= q_{ij}\alpha_{ij} \text{ if } i \neq j \\ p_{ii} &= q_{ii} + \sum_{k\neq i} q_{ik}(1-\alpha_{ik}) \end{aligned}$$

This Markov chain will be time-reversible and have stationary measure π if

$$\pi_i P_{ij} = \pi_j P_{ji} \text{ for all } j \neq i$$

which is equivalent to

$$\pi_i q_{ij}\alpha_{ij} = \pi_j q_{ji}\alpha_{ji}$$

and is easy to check that this will be satisfied if we set

$$\alpha_{ij} = \min\left(\frac{\pi_j q_{ji}}{\pi_i q_{ij}}, 1\right)$$

One of the reasons this algorithm is so useful is the following: say we only know the values in π up to a constant, that is we have a sequence $b_j, j = 1, ..m$, $b_j \geq 0$ and $\sum b_j = B$. We want to generate observations from π with $\pi_j = b_j/B$. Then the above algorithm works without the need to find B because

$$\alpha_{ij} = \min\left(\frac{\pi_j q_{ji}}{\pi_i q_{ij}}, 1\right) = \min\left(\frac{b_j q_{ji}}{b_i q_{ij}}, 1\right)$$

With this we get the Metropolis-Hastings Algorithm:

1) Choose an irreducible Markov chain with transition probabilities Q and choose some integer k between 1 and m
2) Let n=0 and $X_0 = k$
3) generate a r.v. X such that $P(X = j) = q_{x_0 j}$ and generate $U \sim U[0,1]$
4) If $U < b_X q_{X,X_n} / b_{X_n} q_{X_n,X}$ then $NS = X$, else $NS = X_n$
5) $n = n+1$, $X_n = NS$
6) Go to 3

Notice the similarities between this algorithm and the accept-reject method. The main difference, and the reason this algorithm is so useful, are that here we don't need to find c, which usually requires a maximization and we don't need B, as discussed above.

####**Example** Let's begin with a very simple example, $X \sim Bin(N,p)$. First we need the "proposal" distribution Q. We are actually quite free to make almost any choice here. Let's try the following: if $X[k] = x$, we next randomly choose either $x-1$, x, or $x+1$. If $x = 0$ we choose either 0, 1 or 2 and if $x = N$ we randomly choose $x = N-2, N-1$ or N. Therefore in either case we have $q_{ij} = 1/3$ and so

$$b_i q_{i,j} / b_j q_{j,i} = \text{dbinom}(i,N,p)/\text{dbinom}(j,N,p)$$

Let's see what the R program looks like:

```
N <- 5
p <- 0.5
B <- 1e4
X <- rep(0, B)
for(i in 2:B){
  if(X[i-1]==0) NS <- sample(0:2, 1)
  if(X[i-1]>0 & X[i-1]<N) NS <- sample(X[i-1]+c(-1:1), 1)
  if(X[i-1]==N) NS <- sample((N-2):N, 1)
  if(runif(1) < dbinom(NS,N,p)/dbinom(X[i-1],N,p)) X[i] <- NS
  else X[i] <- X[i-1]
}
out <- matrix(0, 2, N+1)
colnames(out) <- 0:N
out[1, ] <- round(table(X)/B, 3)
out[2, ] <- round(dbinom(0:N, N, p), 3)
out
```

```
##          0     1     2     3     4     5
## [1,] 0.014 0.151 0.334 0.324 0.160 0.016
## [2,] 0.031 0.156 0.312 0.312 0.156 0.031
```

and this is works very well.

Notice another big difference between the accept-reject algorithm and Metrolopis-Hastings: there we need a distribution on the whole support of X that we can generate. Here we only need one that let's us go from one observation to another.

####**Example**

Let's generate $X \sim G(p)$ so $b_j = P(X=j) = pq^{j-1}$. Therefore we have

$$b_i/b_j = \left(pq^{i-1}\right)/\left(pq^{j-1}\right) = q^{i-j}$$

As a proposal distribution we will use

$$\begin{aligned} q_{11} &= \frac{1}{2} \\ q_{x,x+1} &= \frac{1}{2} \\ q_{x,x-1} &= \frac{1}{2} \text{ if } x > 1 \end{aligned}$$

so always $q_{i,j}/q_{j,i} = 1$.

So now

$$\begin{aligned} &b_1 q_{1,1}/b_1 q_{1,1} = 1 \\ &b_2 q_{2,1}/b_1 q_{1,2} = q \\ &\text{Let } x > 1 \\ &b_x q_{x,x+1}/b_{x+1} q_{x+1,x} = 1/q \\ &b_{x+1} q_{x+1,x}/b_x q_{x,x+1} = q \end{aligned}$$

```r
rgeomMCMC <- function(p) {
  B <- 1e4
  X <- rep(1, B)
  for(i in 2:B){
    which <- TRUE
    if(X[i-1]==1) {
      NS <- sample(1:2, 1)
      if(NS==2)  which <- (runif(1) < 1-p)
    }
    else {
      NS <- X[i-1]+sample(c(-1, 1), 1)
      if(NS==X[i-1]+1) which <- (runif(1) < 1-p)
      else which <- (runif(1) < 1/(1-p))
    }
    if(which) X[i] <- NS
    else X[i] <- X[i-1]
  }
  tmp <- table(X)/B
  x <- as.numeric(names(tmp))
  out <- matrix(0, 2, length(tmp))
  colnames(out) <- x
  out[1, ] <- round(tmp, 3)
  out[2, ] <- round(p*(1-p)^(x-1), 3)
  head(out, 10)
}
rgeomMCMC(0.25)
```

```
##          1     2     3     4     5     6     7     8     9    10    11
## [1,] 0.259 0.203 0.152 0.102 0.076 0.054 0.040 0.031 0.024 0.016 0.013
## [2,] 0.250 0.188 0.141 0.105 0.079 0.059 0.044 0.033 0.025 0.019 0.014
##         12    13    14    15    16    17    18    19    20
## [1,] 0.011 0.009 0.005 0.002 0.002 0.001 0.000 0.000 0.000
## [2,] 0.011 0.008 0.006 0.004 0.003 0.003 0.002 0.001 0.001
```

```
rgeomMCMC(0.5)
```

```
##          1     2     3     4     5     6     7     8     9    10    11
## [1,] 0.506 0.251 0.123 0.059 0.027 0.014 0.008 0.005 0.003 0.001 0.001
## [2,] 0.500 0.250 0.125 0.062 0.031 0.016 0.008 0.004 0.002 0.001 0.000
##         12 13 14 15 16
## [1,] 0.001  0  0  0  0
## [2,] 0.000  0  0  0  0
```

```
rgeomMCMC(0.75)
```

```
##          1     2     3     4     5     6
## [1,] 0.749 0.189 0.046 0.012 0.002 0.001
## [2,] 0.750 0.188 0.047 0.012 0.003 0.001
```

####**Example** say we want to generate $X \sim N(\mu, \sigma)$.

Notice that this is a continuous random variable, but as we will see, that makes no real difference!

Again we need a proposal distribution. Let's consider two: if we are at the point x we choose the next point from

a) $U[x-\epsilon, x+\epsilon]$ for some $\epsilon > 0$.

b) $N(x, \epsilon)$ for some $\epsilon > 0$.

Now

a) $q_{xy} = 1/(2\epsilon)$ if $x-\epsilon < y < x+\epsilon$, 0 otherwise

b) $q_{xy} = dnorm(y, x, \epsilon)$

So the algorithm uses:

a) $X = \text{runif}(1, X_n - \epsilon, X_n + \epsilon])$

$$b_X q_{X,X_n} / b_{X_n} q_{X_n,X} = \frac{\text{dnorm}(X, \mu, \sigma)}{\text{dnorm}(X_n, \mu, \sigma)}$$

b) $X = \text{rnorm}(1, X_n, \epsilon)$

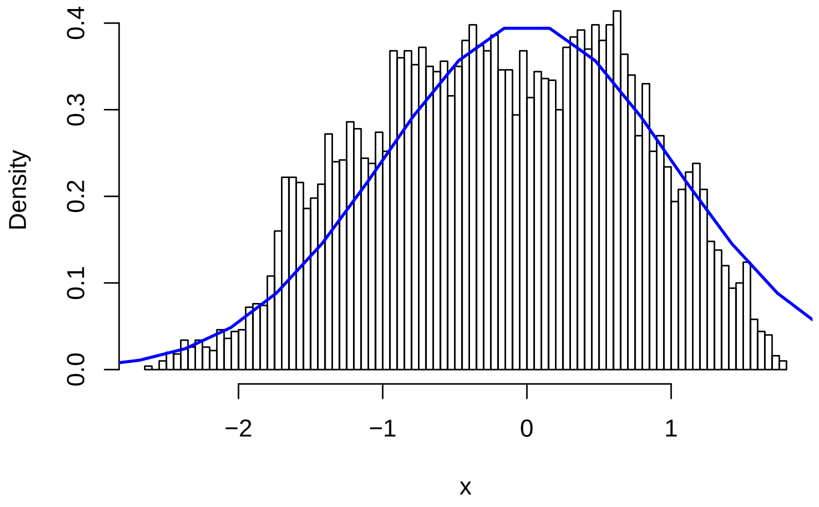
$$b_X q_{X,X_n} / b_{X_n} q_{X_n,X} = \frac{\text{dnorm}(X, \mu, \sigma)\text{dnorm}(X_n, X, \epsilon)}{\text{dnorm}(X_n, \mu, \sigma)\text{dnorm}(X, X_n, \epsilon)}$$

This is implemented in

```r
normMCMC <- function(method=1, n=10000,
                     eps=1, mu=0, sig = 1,
                     start=1, Graph="Both") {
    Xn <- rep(0, n)
    for (i in 2:n) {
        U <- runif(1)
        Accept <- FALSE
        if(method == 1) {
          X <- runif(1, Xn[i-1]-eps, Xn[i-1]+eps)
          if(U<dnorm(X, mu, sig)/dnorm(Xn[i-1], mu, sig))
            Accept <- TRUE
        }
        if(method==2) {
            X <- rnorm(1, Xn[i-1], eps)
            if(U<dnorm(X, mu, sig)*
               dnorm(Xn[i-1], X, eps)/
               dnorm(Xn[i-1], mu, sig)/
               dnorm(X, Xn[i-1], eps))
              Accept <- TRUE
        }
        if(Accept) {
            NS <- X
        }
        else {
            NS <- Xn[i-1]
        }
        Xn[i] <- NS
    }
    if(Graph=="Both") {
       par(mfrow = c(1, 2))

    }
    if(Graph=="Burn")
       plot(1:n, Xn, type = "l")
    if(Graph=="Hist") {
      hist(Xn[start:10000], breaks=100, freq=FALSE,
         xlab="x", main = "")
      x <- seq(mu-3*sig, mu+3*sig, length = 20)
```

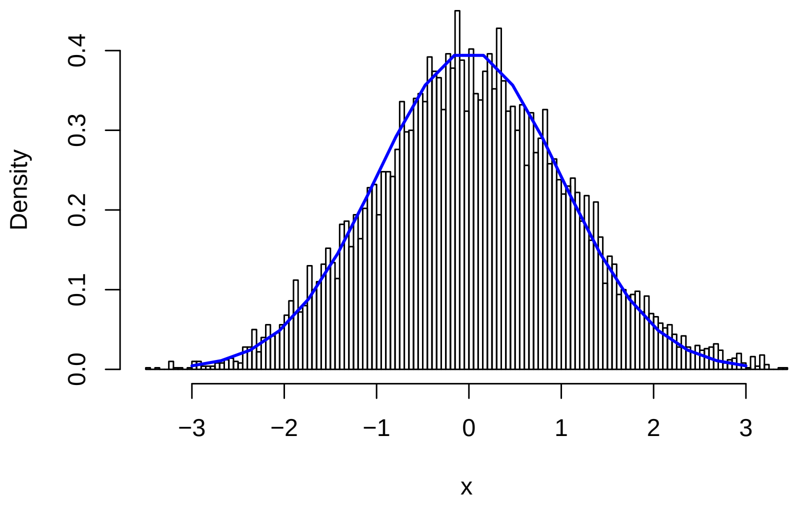
```
        lines(x, dnorm(x, mu, sig),
            lwd=2, col="blue")
    }

}
```

```
normMCMC(method=1, mu=0, sig=1, eps=0.1, Graph="Hist")
```



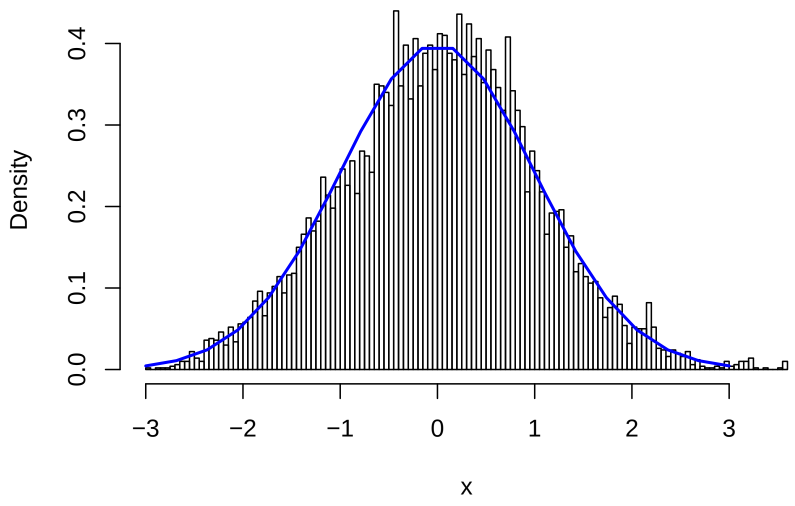
That's not very good. Let's try a different ϵ:

```
normMCMC(method=1, mu=0, sig=1, eps=0.5, Graph="Hist")
```



And method 2:

```r
normMCMC(method=2, mu=0, sig=1, eps=0.5, Graph="Hist")
```



As we can see it takes a little bit of trial and error to get the right proposal distribution (here the ϵ).

Compare this algorithm, and its implementation, with the accept-reject algorithm. Here we needed practically no calculations at all.

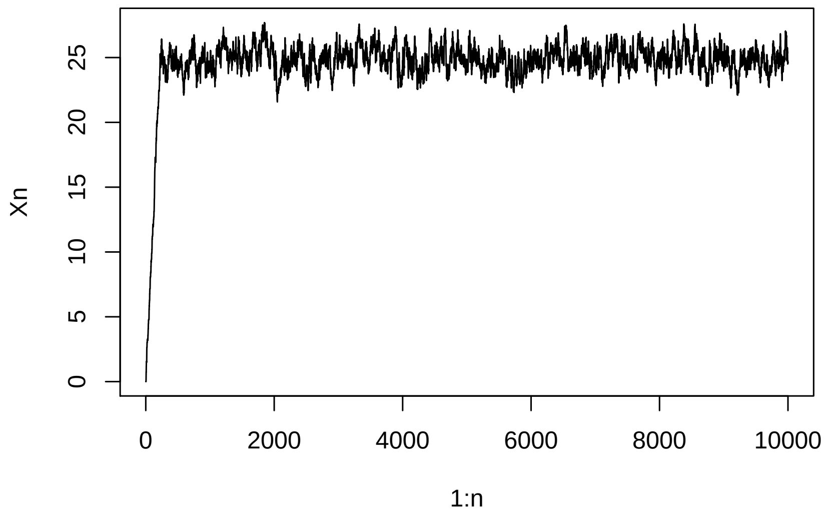
There are two main difficulties with the MCMC method in practice:

1) It can take a lot of computational effort, for example if we want to generate just 1 variate at a time we still might have to generate 10000 others before the stationary distribution is reached.
2) It can be very difficult in practice to know when the stationary distribution is reached, that is when the "burn-in" period is over.

####**Example**

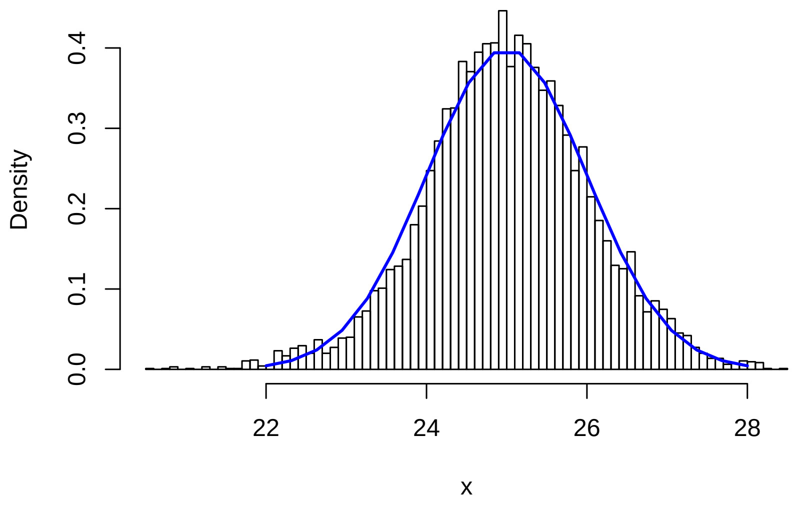
Let's consider the normal again, but this time with $\mu = 25$. Our routine always starts as 0, so it will take a while until it gets to likely values of X. One can look at this by considering the sequence of generated values:

```
normMCMC(method=1, mu=25, sig=1, eps=0.5, Graph="Burn")
```



so we should disregard the first 500(?) or so variates:

```
normMCMC(method=1, mu=25, sig=1, eps=0.5,
         start=501, Graph="Hist")
```



There are examples where the chain seems to have settled down for very long periods but is not actually at the stationary distribution yet.

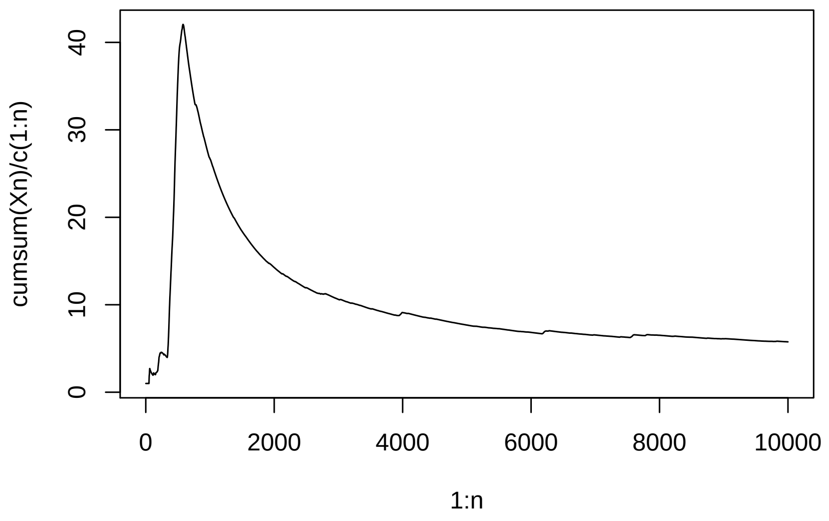
####**Example** say we want to generate r.v.'s X such that $P(X = k) = c/k^r$, $r > 1$ and $k = 1, 2, ...$ We did this already for $k = 2$ using the accept-reject algorithm, with the Cauchy as the Y distribution. Now we would need to know c for any value of r, which is not possible. Let's instead use MH. First we use the following proposal distribution:

if $X_n \le m$, $X \sim U\{1..(2m+1)\}$ for some m (here m=10) otherwise $X \sim U\{-m, m\}$

so $q_{X,X_n} = 1/(2m+1)$ and

$$b_X q_{X,X_n}/b_{X_n} q_{X_n,X} = \frac{\mathrm{c}/X^r}{\mathrm{c}/X_n^r} = \left(\frac{X_n}{X}\right)^r$$

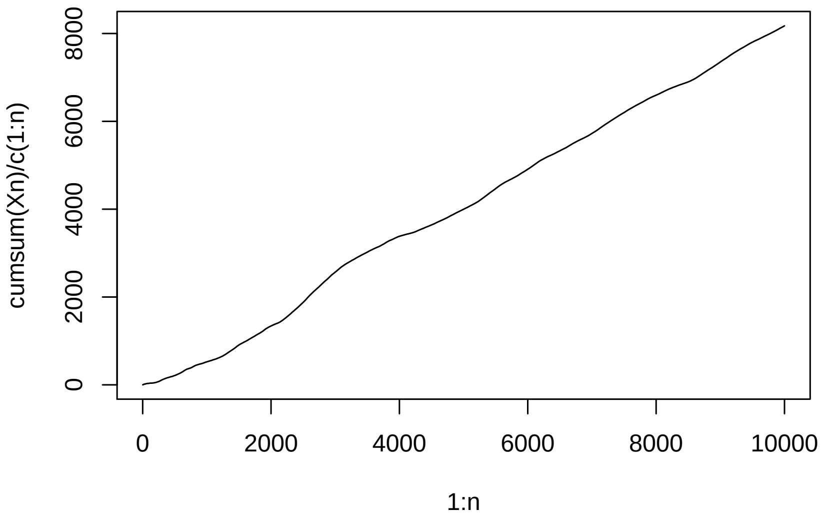
```
mcmcInvr <- function(which=1, n=10000, r=2, m=10) {
    Xn = rep(1, n)
    if (which == 1) {
        for (i in 2:n) {
            if (Xn[i - 1] <= m)
                X <- sample(1:(2 * m + 1), size = 1)
            else X <- sample(Xn[i - 1] + c(-m:m), size = 1)
            if (runif(1) < (Xn[i - 1]/X)^r)
                Xn[i] <- X
            else Xn[i] <- Xn[i - 1]
        }
    }
    if (which == 2) {
        q <- function(x, y) {
            dbinom(x - 1, 2 * y, 0.5)
        }
        for (i in 2:n) {
            X <- rbinom(1, 2 * Xn[i - 1], 0.5) + 1
            if (runif(1) < (Xn[i - 1]/X)^r * q(X, Xn[i - 1])/q(Xn[i -
                1], X))
                Xn[i] <- X
            else Xn[i] <- Xn[i - 1]
        }
    }
    plot(1:n, cumsum(Xn)/c(1:n), type="l")
    x <- Xn[1001:n]
    z <- table(x)/length(x)
    k <- as.numeric(names(z))
    const <- 1/sum(c(1:max(x))^(-r))
    truep <- const/k^r
    z <- rbind(z, truep)
    round(z[,truep>1/1000],3)
}
mcmcInvr()
```



```
##           1     2     3     4     5     6     7    8     9    10    11
## z     0.624 0.127 0.060 0.039 0.025 0.017 0.011 0.01 0.010 0.007 0.006
## truep 0.613 0.153 0.068 0.038 0.025 0.017 0.013 0.01 0.008 0.006 0.005
##          12    13    14    15    16    17    18    19    20    21    22
## z     0.004 0.006 0.004 0.004 0.004 0.004 0.003 0.002 0.003 0.002 0.001
## truep 0.004 0.004 0.003 0.003 0.002 0.002 0.002 0.002 0.002 0.001 0.001
##          23    24
## z     0.002 0.001
## truep 0.001 0.001
```

next let's try $X \sim Bin(2Xn[i-1], 0.5) + 1$. This shows that not all choice of Q work:

```
mcmcInvr(2)
```



```
##
## z
## truep
```

####**Example** Let's consider a normal mixture model, that is we have $N_1 \sim N(\mu_1, \sigma_1)$, $N_2 \sim N(\mu_2, \sigma_2)$ and $Z \sim Ber(\alpha)$ and we observe

$$X = ZN_1 + (1-Z)N_2$$

let's say we want to carry out a **Bayesian analysis**. This means we will treat the parameters as random variables. To keep things simple we assume that μ_1, σ_1, μ_2, σ_2 are known and the only parameter is α. As a rv α has a distribution, called the **prior**. An obvious choice is a beta distribution (because $0 \le \alpha \le 1$), and again to keep things simple we will use $\alpha \sim \text{Beta}(\tau, \tau)$, that is we use a prior centered at 0.5. In a standard Bayesian analysis we will have to calculate the **posterior** distribution, that conditional density of

$$\alpha | X_1 = x_1, X_2 = x_2, .., X_n = x_n$$

$$f(\mathbf{x},\alpha) = \prod_{i=1}^{n}(\alpha\varphi(X_i|\mu_1,\sigma_1) + (1-\alpha)\varphi(X_i|\mu_2,\sigma_2))b(\alpha|\tau,\tau)$$

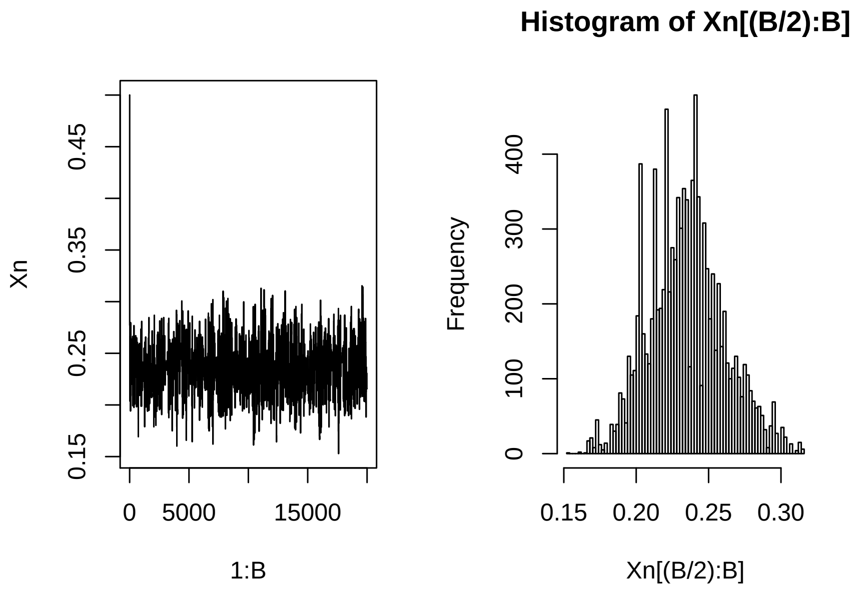
$$f(\mathbf{x}) = \int_0^1 f(\mathbf{x},\alpha)d\alpha = 1/C$$

$$f(\alpha|\mathbf{x}) = \frac{f(\mathbf{x},\alpha)}{f(\mathbf{x})} = Cf(\mathbf{x},\alpha)$$

Finding the exact posterior distribution means finding the marginal distribution which in this case is hopeless analytically. Fortunately we don't need it for the Metropolis-Hastings algorithm.

Here is the routine:

```
mcmcMix <- function(truealpha, N, mu1=0, sig1=1,
                    mu2=2, sig2=0.5, B=20000, eps=1.0) {
    set.seed(1111)
    Z <- sample(c(0, 1), size=N, replace=TRUE,
                prob = c(1-truealpha, truealpha))
    data <- (1-Z)*rnorm(N, mu1, sig1)+
                Z*rnorm(N, mu2, sig2)
    phi1 <- dnorm(data, mu1, sig1)
    phi2 <- dnorm(data, mu2, sig2)
    f <- function(x, y) {
        exp(sum(log((1-x)*phi1+x*phi2)-
                log((1-y)*phi1+y*phi2) ))
    }
    Xn <- rep(0.5, B)
    for (i in 2:B) {
        X <- runif(1, max(0, Xn[i-1]-eps),
                   min(1, Xn[i-1]+eps))
        X <- runif(1)
        if (runif(1) < f(X, Xn[i-1])) {
            Xn[i] <- X
        }
        else {
            Xn[i] <- Xn[i-1]
        }
    }
    par(mfrow = c(1, 2))
    plot(1:B, Xn, type = "l")
    hist(Xn[(B/2):B], breaks = 100)
    round(c(truealpha, quantile(Xn[(B/2):B],
                           c(0.05, 0.5, 0.95))), 3)
}
mcmcMix(truealpha=0.25, N=500, eps=0.15)
```



```
##           5%   50%   95%
## 0.250 0.195 0.233 0.280
```

####**Example**
Let's generate data from the rv (X, Y) with $f(x, y) = c/(x + y)$ $1 < x < 2$, $1 < y < 2$ using the Metropolis-Hastings algorithm.

Here we will use the following Markov process: if $X_n[i-1, 1] = x$, choose

$X \sim U[1, 2\epsilon]$ if $x < 1 + 2\epsilon$

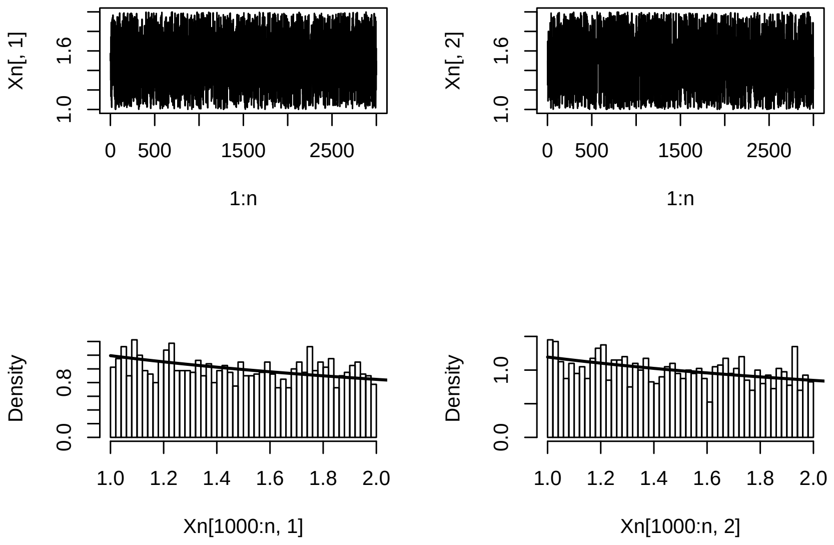
$X \sim U[x - \epsilon, x + \epsilon]$ if $1 + \epsilon < x < 2 - \epsilon$

$X \sim U[2 - 2\epsilon, 2]$ if $x > 2 - 2\epsilon$

for some $\epsilon > 0$, and the same for Y.

```
 mcmcXY <- function (eps, n = 3000)
{
    f <- function(x) 1/sum(x)
    Q <- function(x) {
        if(x<1+2*eps)
            return(runif(1, 1, 1+2*eps))
        if (x>2-2*eps)
            return(runif(1, 2-2*eps, 2))
        return(runif(1, x-eps, x+eps))
    }
    Xn <- matrix(1.5, n, 2)
    X <- c(0, 0)
    for (i in 2:n) {
        X[1] <- Q(Xn[i-1, 1])
        X[2] <- Q(Xn[i-1, 2])
        if(runif(1) < f(X)/f(Xn[i-1, ])) {
            Xn[i, ] <- X
        }
        else {
            Xn[i, ] <- Xn[i-1, ]
        }
    }
    par(mfrow = c(2, 2))
    plot(1:n, Xn[, 1], type = "l")
    plot(1:n, Xn[, 2], type = "l")
    x = seq(1, 2.2, 0.01)
    hist(Xn[1000:n, 1], breaks=50, freq=FALSE, main="",
        density=-1)
    lines(x, 2.943*(log(x+2) - log(x+1)), lwd = 2)
    hist(Xn[1000:n, 2], breaks=50, freq=FALSE, main="",
        density=-1)
    lines(x, 2.943*(log(x+2) - log(x+1)), lwd = 2)

}
```

```
mcmcXY(eps=0.5)
```



Note that for $\epsilon < 0.4$ or so this does not work, the chain gets stuck close to the corners.

Of course for $\epsilon = 0.5$ we have $X[i] \sim U[1,2]$, so the chain is an independent sequence, and the method still works!

####**Example** Recall the example at the end of the section on general methods for generating data: we have a random vector $(X_1, .., X_d)$ with density

$$f(x_1, .., x_d) = c \prod x_i, \; 0 \le x_1 \le .. \le x_d \le 1$$

To generate data from this density use the following Markov process Q:

if Y is the last point, randomly choose a coordinate j from 1:d and choose X=Y except that

$$X_j \sim U[Y_{j-1}, Y_{j+1}]$$

(with $Y_0 = 0$ and $Y_{d+1} = 1$). Notice that X again is a possible point from this rv and that

$$b_X q_{X,X_n} / b_{X_n} q_{X_n,X} = X_j / Y_j$$

```
mcmcd <- function (n = 1e4, d = 10)
{
    Xn <- matrix(0, n, d)
    Xn[1, ] <- c(1:d)/(d+1)
    for(i in 2:n) {
        X <- Xn[i-1, ]
        j <- sample(1:d, 1)
        if (j==1)
            X[1] <- runif(1, 0, X[2])
        if (j==d)
            X[d] <- runif(1, X[d-1], 1)
        if (j>1 & j<d)
            X[j] <- runif(1, X[j-1], X[j+1])
        Xn[i, ] <- Xn[i-1, ]
        if (runif(1) < X[j]/Xn[i-1, j])
            Xn[i, j] <- X[j]
    }
    Xn <- Xn[1000:n, ]
    if(d==2) {
        par(mfrow = c(2, 2))
        hist(Xn[, 1], breaks = 50, freq = FALSE)
        x <- seq(0, 1, 0.01)
        lines(x, 4*x*(1-x^2))
        hist(Xn[, 2], breaks = 50, freq = FALSE)
        lines(x, 4*x^3)
    }
    if(d==3) {
        par(mfrow = c(2, 2))
        hist(Xn[, 1], breaks=50, freq=FALSE, main="")
        x <- seq(0, 1, 0.01)
        lines(x, 6*x*(1-x^2)^2)
        hist(Xn[, 2], breaks=50, freq=FALSE, main="")
        lines(x, 12*x^3*(1-x^2))
        hist(Xn[, 3], breaks=50, freq=FALSE, main="")
        lines(x, 6*x^5)
    }
    if(d>3) {
        par(mfrow = c(1, 1))
        hist(Xn[, d], breaks=50, freq=FALSE, main="")
        x <- seq(0, 1, 0.01)
        lines(x, 2*d*x^(2*d-1))
    }

}
```

This seems almost to easy! How can we verify that this indeed generates the right data? In general this is really impossible, but let's at least do it for some special cases:

d=2:

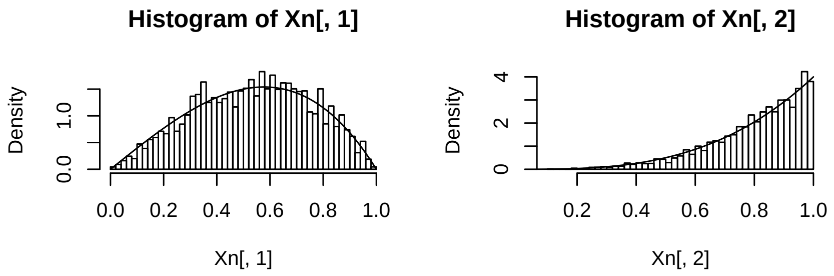
$$f_x(x) = \int_x^1 cxydy = cx\frac{y^2}{2}|_x^1 = \frac{c}{2}x(1-x^2),\ 0 < x < 1$$

$$\int_0^1 f_x(x)dx = \int_0^1 \frac{c}{2}x(1-x^2)dx = \frac{c}{2}(\frac{x^2}{2} - \frac{x^4}{4}|_0^1 =$$

$$\frac{c}{2}(\frac{1}{2} - \frac{1}{4}) = \frac{c}{8} \text{ so } c = 8$$

$$f_y(y) = \int_0^y 8xydx = 4xy^2|_0^y = 4y^3,\ 0 < y < 1)$$

```
mcmcd(d=2)
```



d=3:

$$f_{x,y}(x,y) = c\int_y^1 xyzdz = cxy\frac{z^2}{2}\Big|_y^1 = \frac{c}{2}xy(1-y^2),\ 0<x<y<1$$

$$f_x(x) = \int_x^1 \frac{c}{2}xy(1-y^2)dy = \frac{c}{2}x(\frac{y^2}{2} - \frac{y^4}{4}\Big|_x^1 =$$

$$\frac{c}{2}x(\frac{1}{2} - \frac{1}{4} - (\frac{x^2}{2} - \frac{x^4}{4})) = \frac{c}{8}x(1-2x^2+x^4) = \frac{c}{8}x(1-x^2)^2,..0<x<1$$

$$\int_0^1 f_x(x)dx = \frac{c}{8}(\frac{x^2}{2} - 2\frac{x^4}{4} + \frac{x^6}{6}\Big|_0^1 = \frac{c}{8}(\frac{1}{2} - \frac{1}{2} + \frac{1}{6}) = \frac{c}{48} \text{ so } c = 48$$

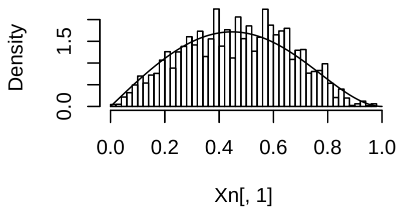
$$\text{so } f_x(x) = 6x(1-x^2)^2,..0<x<1$$

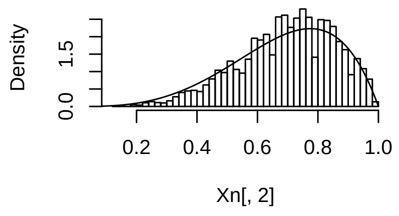
$$fy(y) = \int_0^y 48xy(1-y^2)dx = 24x^2y(1-y^2)\Big|_0^y = 24y^3(1-y^2),..0<y<1$$

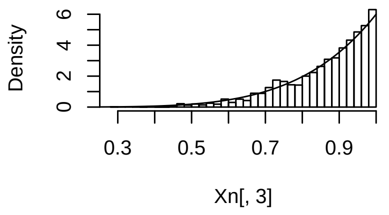
$$f_{y,z}(y,z) = \int_0^y 48xyzdx = 24x^2yz\Big|_0^y = 24y^3z,\ 0<y<z<1$$

$$f_z(z) = \int_0^z 24y^3zdy = 6y^4z\Big|_0^z = 6z^5,\ 0<z<1$$

```
mcmcd(d=3)
```







Notice that

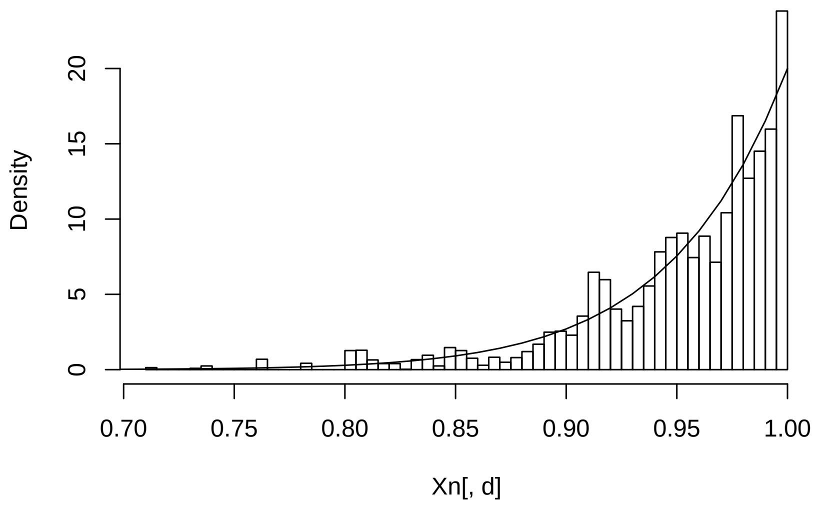
if d=2 $f_{x_2}(x) = 4x^3$

if d=3 $f_{x_3}(x) = 6x^5$

so maybe

$f_{x_d}(x) = 2dx^{2d-1}$?

```
mcmcd(d=10)
```

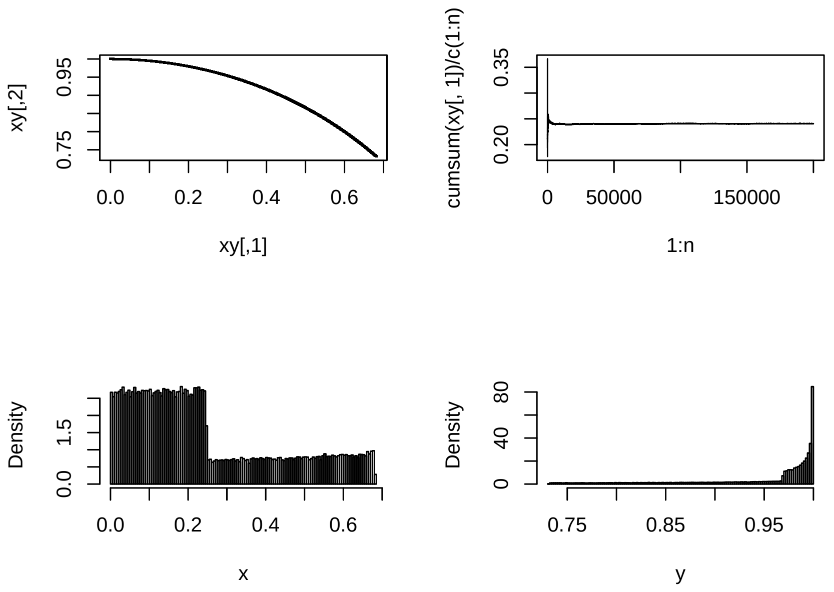


this appears to be true

####**Example** let (X, Y) be a random vector which takes values uniformly on the unit circle, that $f(x, y) = 1/(2\pi)$ for $\{(x, y) : x^2 + y^2 = 1\}$. We want to generate data from (X, Y).

this is quite easy to do with polar coordinates: let $Z \sim U[0, 2\pi]$ and set $X = \sin(Z)$, Y $\cos(Z)$, done in **mcmcCircle(1)**

```
mcmcCircle <- function (which = 1, n = 2 * 1e+05, eps = 0.2)
{
    xy <- matrix(0, n, 2)
    if (which == 1) {
        phi <- runif(n, 0, 2 * pi)
        xy[, 1] <- sin(phi)
        xy[, 2] <- cos(phi)
    }
    if (which == 2) {
        xy[1, ] <- c(1, 0)
        for (i in 2:n) {
            u <- runif(1, xy[i - 1, 1] - eps, xy[i - 1, 1] +
                eps)
            v <- runif(1, xy[i - 1, 2] - eps, xy[i - 1, 2] +
                eps)
            xy[i, ] <- c(u, v)/sqrt(u^2 + v^2)
        }
    }
    par(mfrow = c(2, 2))
    plot(xy, pch = ".")
    plot(1:n, cumsum(xy[, 1])/c(1:n), type = "l")
    lines(1:n, cumsum(xy[, 2])/c(1:n), col = "blue")
    hist(xy[c((n/2):n), 1], 100, freq=FALSE, main = "", xlab = "x")
    hist(xy[c((n/2):n), 2], 100, freq=FALSE, main = "", xlab = "y")

}
mcmcCircle(1)
```



how about doing it with MCMC? The problem here is that we need to choose another point, again on the circle. Let's do this: we pick a point uniformly in

$$[x-\epsilon, x+\epsilon] \text{ x } [y-\epsilon, y+\epsilon]$$

and then find the point on the circle closest to it.

How can we find this point? we need to

$$\min\{(u-x)^2 + (v-y)^2 \ : x^2 + y^2 \ = \ 1\}$$

$$\frac{d}{dx} \ = 2(u-x)(-1) + \lambda 2x = 0$$

$$(1+\lambda)x = u, \ (1+\lambda)y = v$$

$$(1+\lambda)^2 x^2 + (1+\lambda)^2 y^2 \ = u^2 + v^2$$

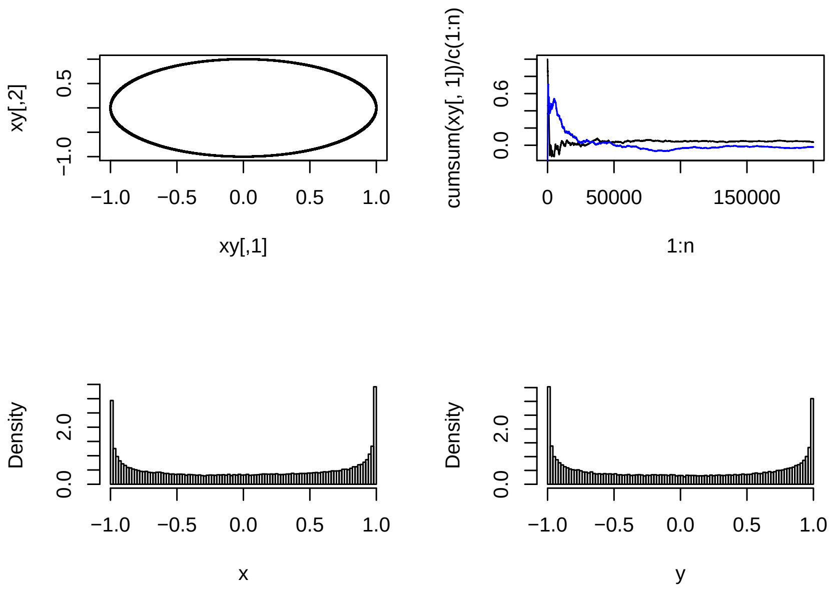
$$(1+\lambda)^2 \ = u^2 + v^2$$

$$1 + \lambda \ = \ \sqrt{u^2 + v^2}$$

$$x \ = \ \frac{u}{\sqrt{u^2+v^2}}, \ y \ = \ \frac{v}{\sqrt{u^2+v^2}}$$

and this is done in

```
mcmcCircle(2)
```



The advantage of this solution is that it can be generalized: say we want to pick (X,Y) uniformly from the points on the curve with $g(x,y)=0$. Now if this is not a circle the polar coordinates don't help but the MCMC solution still works (although finding the point on the curve closest to (u,v) probably now means solving a nonlinear system!)

####**Example** A standard exercise in probability is to show that if X,Y are iid Pois(λ)

$$X|X+Y=n \sim \text{Bin}(n,1/2)$$

and so

$$E[X|X+Y=n]=n/2$$

Let's say we want to generalize this and find

$$E[X_1|X_1+..+X_d=n]$$

In a direct simulation approach we would do the following:

1) generate $X_1,..,X_d$ iid P(λ)
2) if $X_1+..+X_d=n$ set $Z=X_1$, otherwise go to 1)
3) repeat 1 and 2 say 1000 times and the find the mean of the Z

The problem is that if d is not small we will rarely find $X_1+..+X_d=n$ and so we will need to run through 1 and 2 many times to find an acceptable X_1. Of course we have $X_1+..+X_d \sim P(d\lambda)$, so

$$P(\sum_{i=1}^{d} X_i = n) = \frac{(d\lambda)^n}{n!}e^{-d\lambda}$$

for example if d=5, $\lambda=1$ and n=10 we have p=0.018, so we would find a good candidate only every 1/0.018=55 tries.

mcmcPois(1) does it anyway.

```
mcmcPois <- function (which = 1, d = 2, n = 2 * d, lambda = 1)
{
    if (which == 1) {
        m = 1000
        x1 = rep(0, m)
        counter = 0
        for (i in 1:m) {
            repeat {
                counter = counter + 1
                x = rpois(d, lambda)
                if (sum(x) == n)
                  break
            }
            x1[i] = x[1]
        }
        return(round(mean(x1), 2))
    }
    if (which == 2) {
        m = 11000
        x1 = rep(0, m)
        x = c(n, rep(0, d - 1))
        for (i in 1:m) {
            y = x
            I = sample(1:d, size = 2)
            x[I[1]] = rpois(1, lambda)
            x[I[2]] = n - sum(x[-I[2]])
            if (runif(1) > dpois(x[I[1]], lambda)/dpois(y[I[1]],
                lambda))
                x = y
            x1[i] = x[1]
        }
        return(round(mean(x1[1001:m]), 2))
    }
}
```

Now let's use the Metropolis-Hastings algorithm. We begin with the point $(n, 0, .., 0)$. Then in each step we choose two coordinates with

$i \sim U[1, .., d]$, $j \sim U[1, .., d]$, $i \neq j$ and $z \sim rpois(1, \lambda)$

Now we set $x^{(k+1)} = x^{(k)}$

Finally if

$U[0, 1] < \text{dpois}(z, \lambda)/\text{dpois}(x[i], \lambda)$

we set

$x^{(k+1)}[i] = z$, $x^{(k+1)}[j] = n - \sum x^{(k+1)}[-i]$

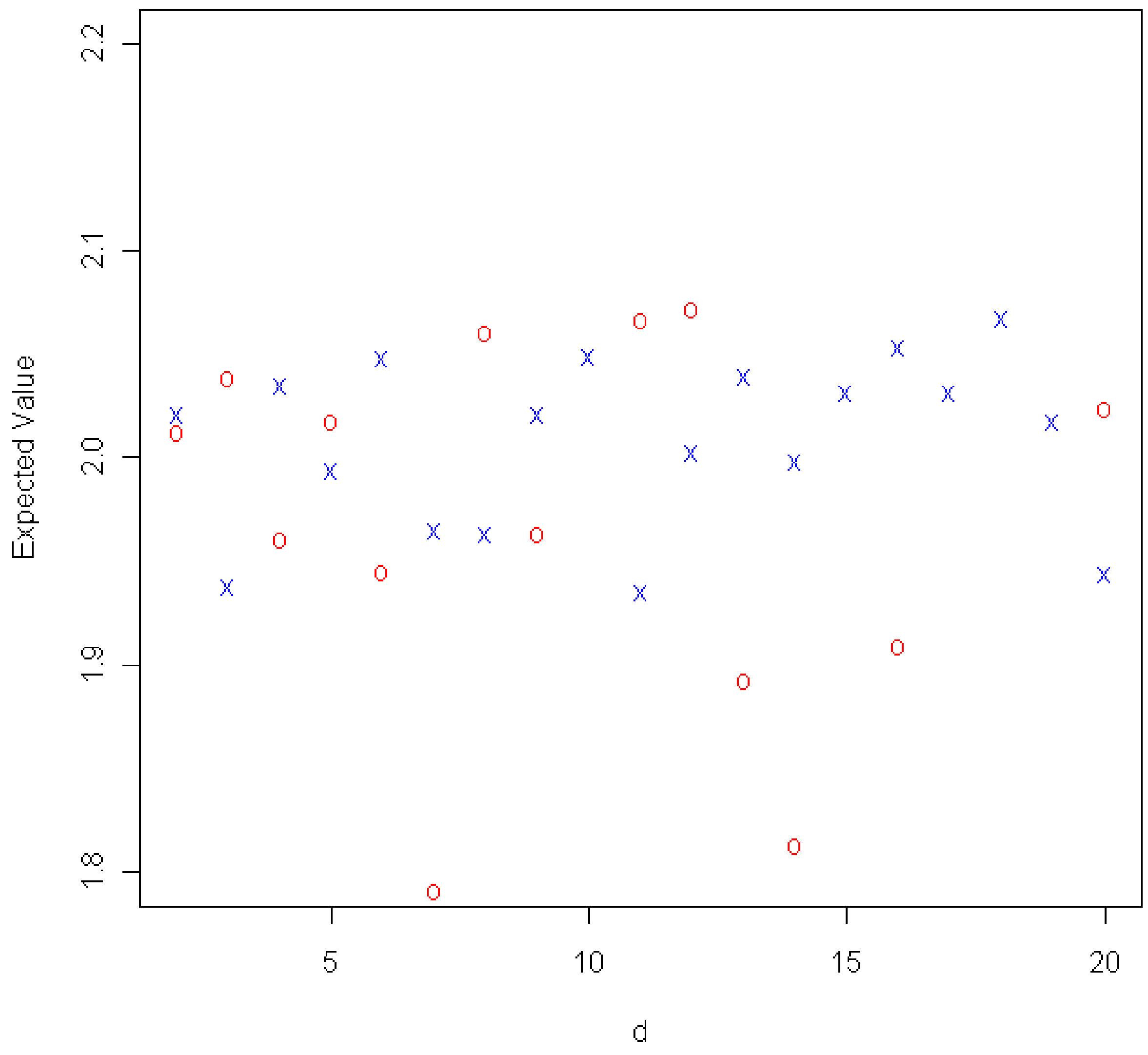
so that again we have $x_1 + .. + x_d = n$

In this case we can use the direct simulation as a check on the MCMC simulation, at least where the direct simulation is not to slow. First let's check the case d=2, $\lambda = 1.0$, where we know the correct answer: n/2. The dots are the estimated values using the MCMC algorithm

```
for(i in 2:5) {
  print(c(i, mcmcPois(1, d=i), mcmcPois(2, d=i)))
}
```

```
## [1] 2.00 2.02 2.02
## [1] 3.00 2.02 1.94
## [1] 4.00 1.94 2.05
## [1] 5.00 2.09 2.15
```

Next consider the case where n is the right size so that $X_1 + .. + X_d = n$ happens reasonably often, namely $n = d\lambda$. Using $\lambda = 2$ we find

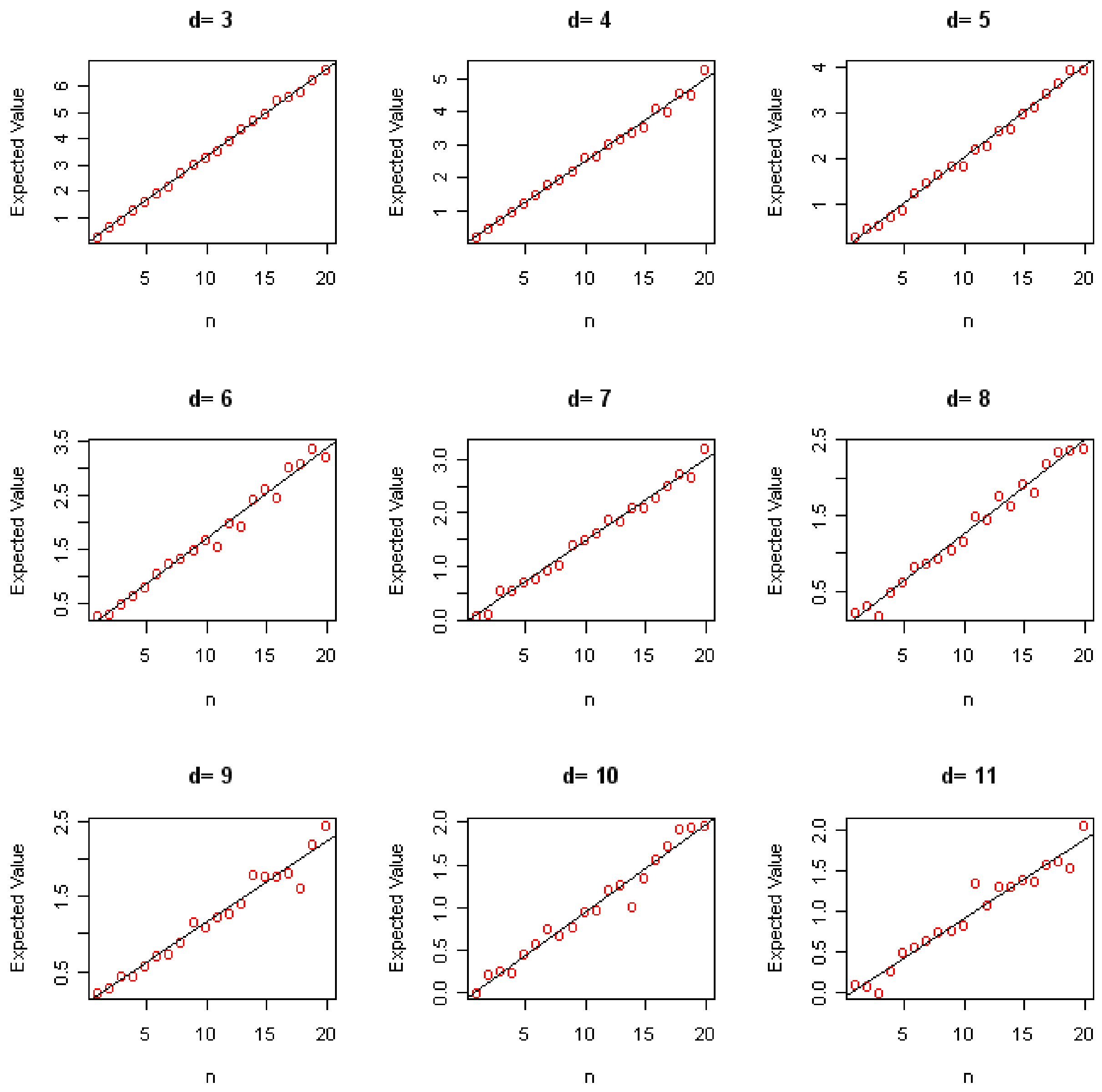


so it appears our MCMC simulation works.

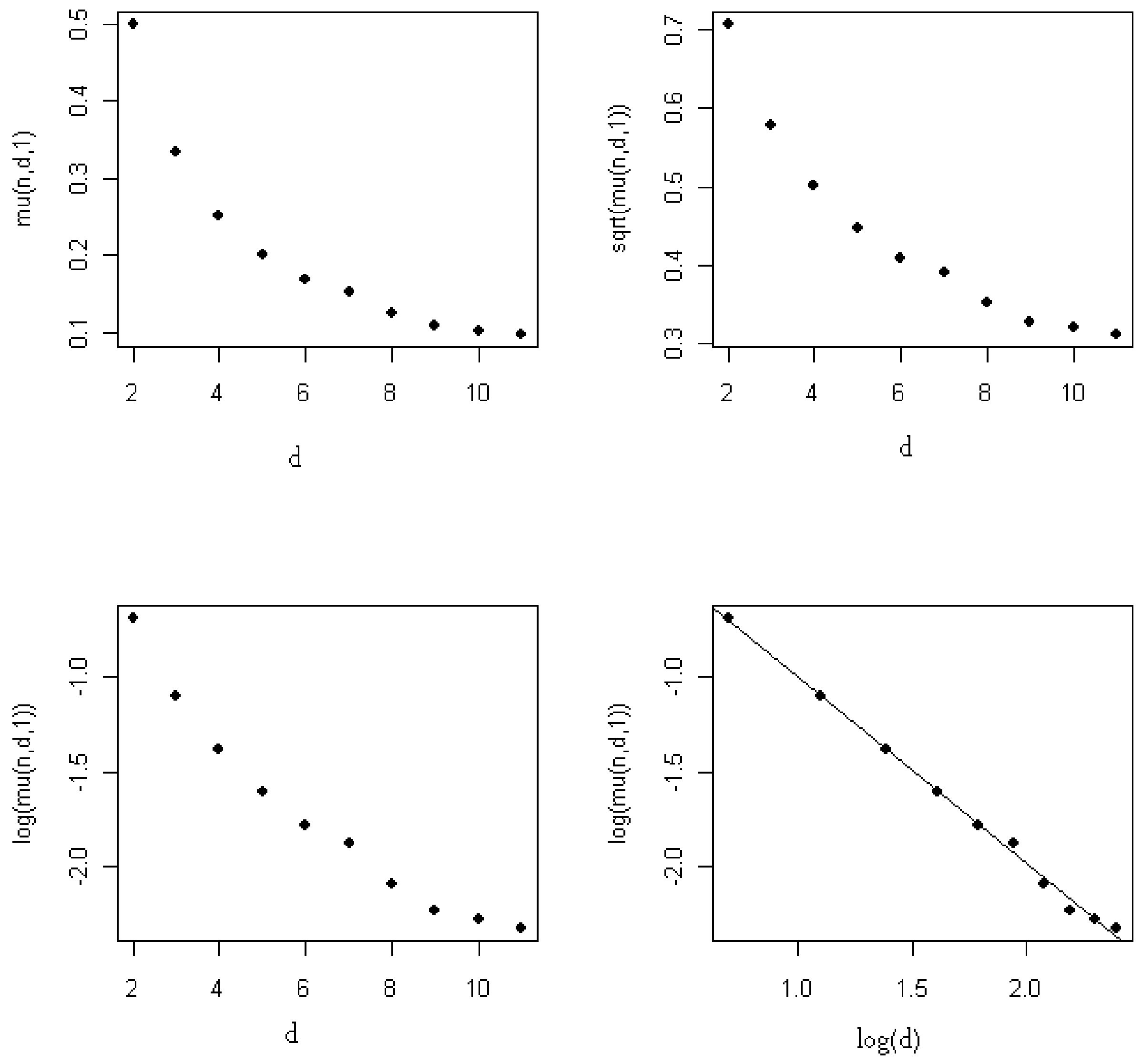
Now let's see whether we can use our simulation to derive a formula for

$\mu; (d, n, \lambda) = E[X_1 | X_1 + .. + X_d = n]$

In the next graph we have the plots for $\lambda = 1$, $n = 1 : 20$ and $d = 3 : 11$ together with the least squares regression lines:



It appears that as a function of n $\mu(n, d, 1)$ is linear with an intercept of 0. How about its dependence on d? In the next graph we have the plot of n vs. the slope of the least squares regression lines, together with several transformations:



The log-log transform yields a straight line relationship! Its equation is given by

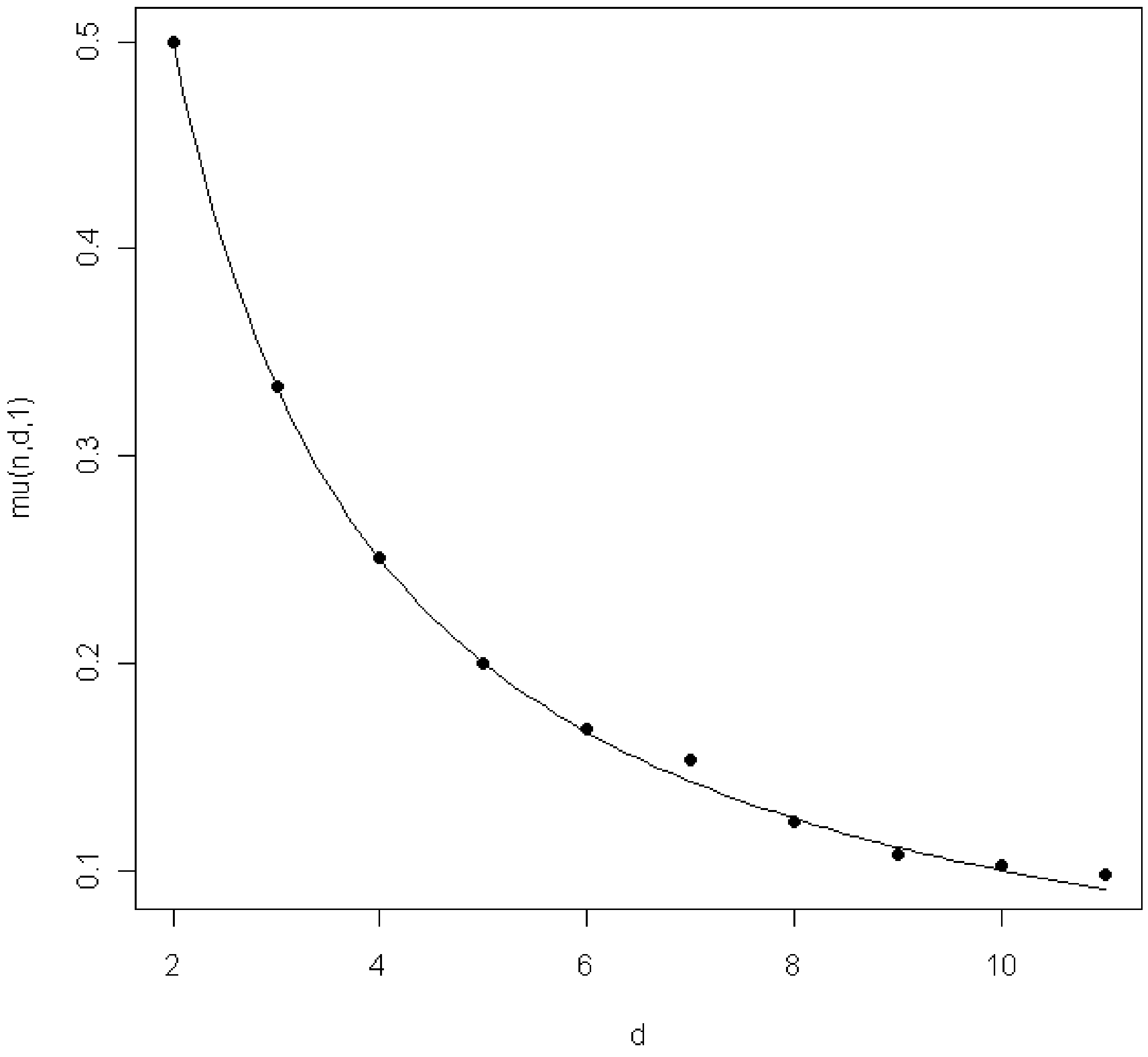
$$y = -0.02040 - 0.98070x$$

so it seems the intercept might be 0 and the slope -1, which means $\log(\mu(d, n, 1))$ proportional to $-\log(d)$

`or`

$\mu(d, n, 1)$ proportional to 1/d

The next graph has the slopes in the original scale, with the 1/d line:

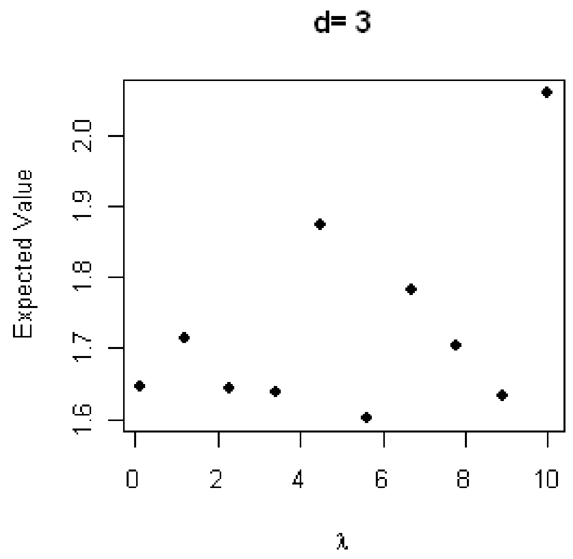


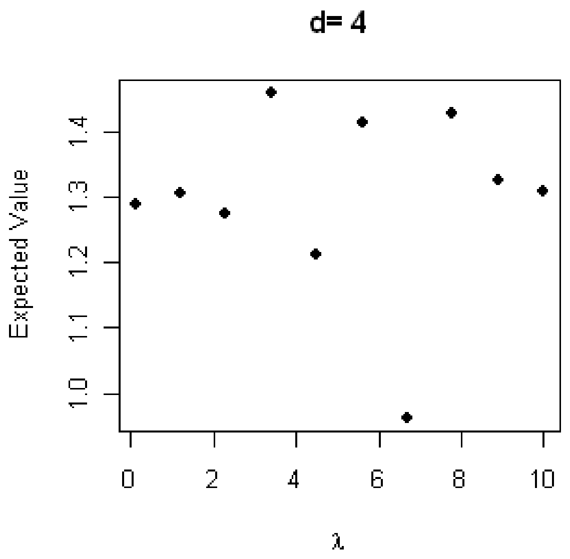
and that seems to fit really well! So now we know (or at least suspect)

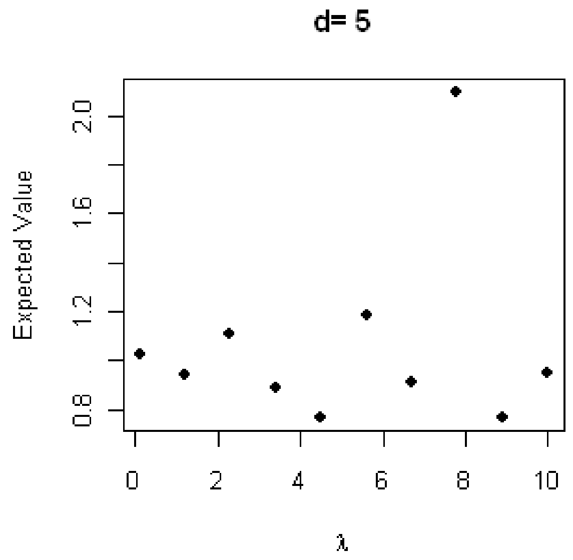
]$\mu(d, n, 1) = n/d$

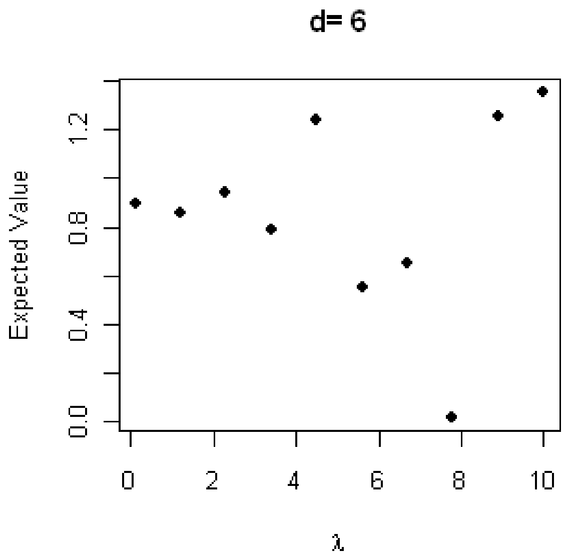
which fits the known result ($\mu(2, n, 1) = n/2$) perfectly!

Last, the dependence on λ. In the next graph we do the simulation for n=5, d=3, 4, 5 and 6 and λ from 0.1 to 10:









It seems there is no dependence on λ, and so we find our function:

$$\mu(d, n, \lambda) = n/d$$

Actually, we can also just do the math:

$$\mu(n,d,\lambda) = E[X_1|\textstyle\sum_{i=1}^d X_i = n] =$$
$$\textstyle\sum_{x=0}^n xP(X_1 = x|\sum_{i=1}^d X_i = n)$$
$$P(X_1 = x|\textstyle\sum_{i=1}^d X_i = n) = \frac{P(X_1=x,\sum_{i=1}^d X_i=n)}{P(\sum_{i=1}^d X_i=n)} = \frac{P(X_1=x)P(\sum_{i=2}^d X_i=n-x)}{P(\sum_{i=1}^d X_i=n)} =$$
$$\frac{\frac{\lambda^x}{x!}e^{-\lambda}\frac{[(d-1)\lambda]^{n-x}}{(n-x)!}e^{-(d-1)\lambda}}{\frac{[d\lambda]^n}{n!}e^{-d\lambda}} = \binom{n}{x}\frac{(d-1)^{n-x}}{d^n} = \binom{n}{x}\left(\frac{1}{d}\right)^x\left(1-\frac{1}{d}\right)^{n-x}$$
so $X_1|\sum_{i=1}^d X_i = n \sim Bin(n,\frac{1}{d})$ and
$$\mu(n,d,\lambda) = E[X_1|\textstyle\sum_{i=1}^d X_i = n] = \frac{n}{d}$$

And finally, a really easy proof:

$$t = E[\sum X|\sum X = t] = \sum E[X_i|\sum X = t] = nE[X_1|\sum X = t]$$

####**Example** let's write a "general one-dim data generator" routine. That is, for any function f with

$$f(x) \geq 0 \text{ on } [A,B]\, c = \int_A^B f(x)dx < \infty$$

we want our routine to generate data from the corresponding density g(x)=f(x)/c.

One way to do this would be to find c via numerical integration and use accept-reject. Instead we will use the Metropolis-Hastings algorithm.

Because f is defined on a finite interval we can use

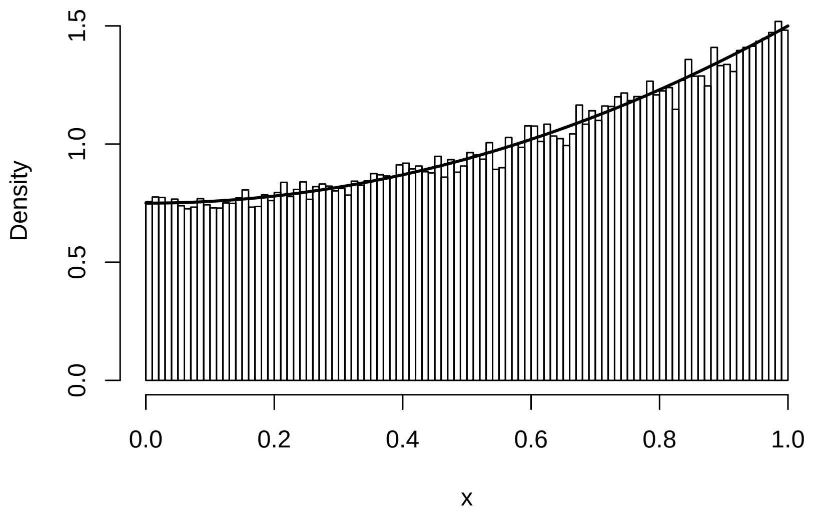
$$q_{x,y} = \text{runif}(1, A, B)$$

Then

$$\begin{aligned}&b_x q_{x,y}/(b_y q_{y,x}) =\\ &f(x)(1/(B-A)/[f(y)(1/(B-A))] =\\ &f(x)/f(y)\end{aligned}$$

this is done in **mcmcf**, which also uses the generated data to find c, draws the histogram and adds the true density.

```
 mcmcf <- function (fun, A=0, B=1, n=1e5, m=1e4)
{
    f <- function(x) {
        eval(parse(text=fun),envir=list(x))
    }
    x<-rep(0, n)
    x[1] <- (A+B)/2
    for(i in 2:n) {
        y <- runif(1,A,B)
        if(runif(1)<f(y)/f(x[i-1])) x[i]<-y
        else x[i]<-x[i-1]
    }
    hist(x,n=100, freq=FALSE, main="")
    z<-seq(A,B,length=250)
    fz <- f(z)
    I <- sum( (fz[-1]+fz[-250]))/2*(z[2]-z[1])
    lines(z,fz/I,lwd=2)

 }
mcmcf("1+x^2")
```



####**Example** say the random vector (X_1, X_2, X_3) has density

$$f(i,j,k) = \frac{c}{i^2+j^2+k^2} \text{ if } i+j+k=M$$

for some known M, i,j and k any integer (except if M=0 i=j=k=0 is not allowed.)

Now for this rv none of the methods we discussed before is going to work. Let's use Metropolis-Hastings as follows:

1) select two coordinates at random:

```r
d <- sample(1:3, size=2)
```

2) select a new value for x[i,d[1]] with

```r
y <- x[i-1, d[1] + sample(-l:l, size=1)]
```

(and we can play around with different values of l)

3) set x[i,d[2]] so that sum(x[i,])=M

$q_{x,y} = 1/(2 * l + 1)$ for all x and y with $|x - y| \le l$

$b_x = c/(i^2 + j^2 + k^2)$

and so

$$\begin{aligned}
&b_x q_{x,y}/(b_y q_{y,x}) = \\
&\left[c/(i^2 + j^2 + k^2)\right] / \left[c/(u^2 + v^2 + z^2)\right] = \\
&(u^2 + v^2 + z^2)/(i^2 + j^2 + k^2)
\end{aligned}$$

One problem here is that this rv is so complicated, it is not even clear what we could do to check that our routine works.

5.3 The Gibbs Sampler

Suppose we want to generate data from a random vector $X_1, .., X_n$ with joint density $f(x_1, .., x_n)$. Unfortunately we know f only up to a constant, that is

$$\begin{aligned}
&f(x_1, .., x_n) = c \cdot u(x_1, .., x_n) \\
&\int_{-\infty}^{\infty} u(x_1, .., x_n) = 1
\end{aligned}$$

Now the conditional distribution of

$$X_k | X_1 = x_1, ., X_{k-1} = x_{k-1}, , X_{k+1} = x_{k+1}, .., X_n = x_n$$

is given by

$$f_k(x_k|x_1,\dots,x_{k-1},x_{k+1},\dots,x_n) =$$

$$\frac{f(x_1,..,x_n)}{\int..\int f(x_1,..,x_n)dx_1..dx_{k-1}dx_{k+1}..dx_n} =$$

$$\frac{c\cdot u(x_1,..,x_n)}{\int..\int c\cdot u(x_1,..,x_n)dx_1..dx_{k-1}dx_{k+1}..dx_n} =$$

$$\frac{u(x_1,..,x_n)}{\int..\int u(x_1,..,x_n)dx_1..dx_{k-1}dx_{k+1}..dx_n}$$

so the density of the conditional distribution function does not depend on c.
The idea of the Gibbs sampler is to generate a sequence of simulated values of

$$f_k(x_k|x_1,..,x_{k-1},x_{k+1},..,x_n)$$

with k going from 1 to n and then starting all over again.
Example Say we want to generate from

$$(X,Y) \sim N(\boldsymbol{\mu},\boldsymbol{\Sigma})$$

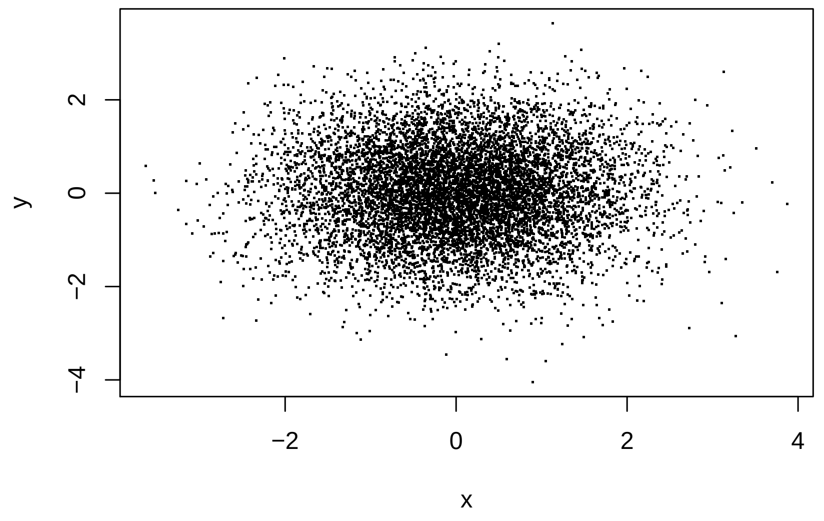
where $\boldsymbol{\mu} = (\mu_x,\mu_y)$ and

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \sigma_x\sigma_y\rho \\ \sigma_x\sigma_y\rho & \sigma_y^2 \end{bmatrix}$$

Recall

$$X|Y = y \sim N(\mu_x + \rho\frac{\sigma_x}{\sigma_y}(y-\mu_y),\sigma_x\sqrt{(1-\rho^2)})$$

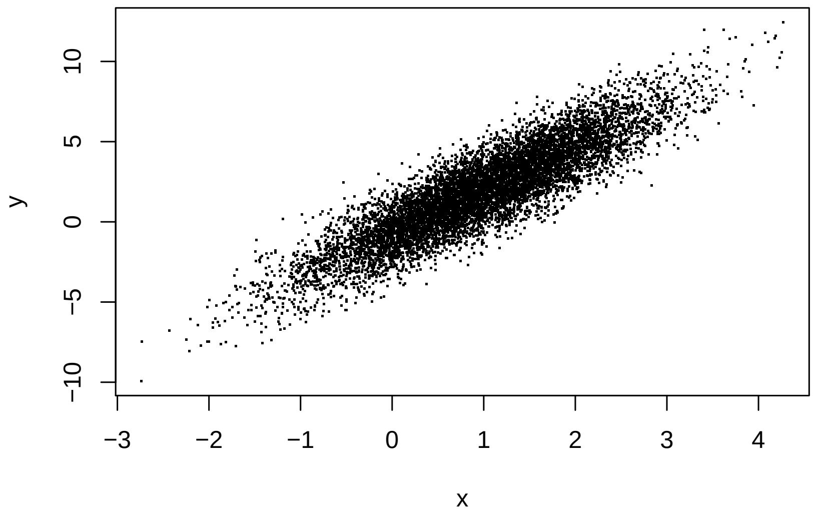
```r
gibbsMN <- function(n=1e4, mu=c(0, 0), sigma=c(1, 1), rho=0){
  x <- rep(0, n)
  y <- rep(0, n)
  for(i in 2:n){
    x[i] <- rnorm(1, mu[1]+rho*sigma[1]/sigma[2]*(y[i-1]-mu[2]),
                  sigma[1]*sqrt(1-rho^2))
    y[i] <- rnorm(1, mu[2]+rho*sigma[2]/sigma[1]*(x[i]-mu[1]),
                  sigma[2]*sqrt(1-rho^2))
      }
  cbind(x, y)[-c(1:1000), ]
}
xy <- gibbsMN()
plot(xy, pch=".")
```



```r
round(c(apply(xy,2,mean), apply(xy,2,sd), cor(xy)[1, 2]), 3)
```

```
##     x     y     x     y
## 0.010 0.021 1.005 0.989 0.008
```

```r
xy <- gibbsMN(mu=c(1,2), sigma = c(1,3), rho = 0.9)
plot(xy, pch=".")
```



```r
round(c(apply(xy,2,mean), apply(xy,2,sd), cor(xy)[1, 2]), 3)
```

```
##     x     y     x     y
## 0.981 1.938 0.964 2.879 0.891
```

It can be shown that the Gibbs sampler is actually a special case of the Metropolis-Hastings algorithm:

Let $\boldsymbol{X} = (X_1, .., X_n)$ be a r.v. with probability mass function $f(\boldsymbol{x})$ that need only be specified up to a multiplicative constant, and suppose that we want to generate a r.v. whose distribution is that of $\boldsymbol{X}$. That is we want to generate a r.v. with density $f(\boldsymbol{x}) = c \cdot u(\boldsymbol{x})$ where u is known but c is not. Using the Gibbs sampler assumes that for any i and values x_j, $j \neq i$, we can generate a r.v. X with density

$$P(X = x) = P(X_i = x | X_j = x_j, j \neq i)$$

It operates the Metropolis-Hastings algorithm on a Markov chain with states $\boldsymbol{x} = (x_1, .., x_n)$ and with transition probabilities defined as follows:

Whenever the present state is $\boldsymbol{x}$, a coordinate that is equally likely to be any of the $1, .., n$ is chosen. If coordinate i is chosen, then a r.v. X whose probability mass function is as above is generated and if X=x the state

$$\boldsymbol{y} = (x_1, .., x_{i-1}, x, x\ i + 1, .., x_n)$$

is considered as the next candidate state. In other words, with $\boldsymbol{x}$ and $\boldsymbol{y}$ given the Gibbs sampler uses the Metropolis-Hastings algorithm with

$$q_{\mathbf{xy}} = \frac{1}{n} P(X_i = x | X_j = x_j, j \neq i) = \frac{p(\mathbf{y})}{nP(X_j = x_j, j \neq i)}$$

Because we want the limiting distribution to be p the vector $\boldsymbol{y}$ is then accepted as the new state with probability

$$\alpha_{\mathbf{xy}} = \min\left(\frac{p(\mathbf{y}) q_{\mathbf{yx}}}{p(\mathbf{x}) q_{\mathbf{xy}}}, 1 \right) = \min\left(\frac{p(\mathbf{y}) p(\mathbf{x})}{p(\mathbf{x}) p(\mathbf{y})}, 1 \right) = 1$$

So in the Gibbs sampler the candidate state is **always** accepted as the next state!

Example Here is one of the standard models used in the actuarial sciences (Insurance) to model the number of claims that might have to be paid on a certain type of policy:

$$Y \sim Beta(\alpha, \beta)$$

$$M \sim Pois(\lambda)$$

$$N = M | M > 0$$

$$X | Y = y, N = n \sim Bin(n, y)$$

The idea is this: there is a random number N of policies of the same type (car insurance, health ins, etc.) Obviously $N > 0$ otherwise it's to boring. Each insurance has a probability Y to be claimed, so X is the number of policies that get claimed.

We want to generate data for X. In order to use the Gibbs sampler we need all the conditional distributions. We already have

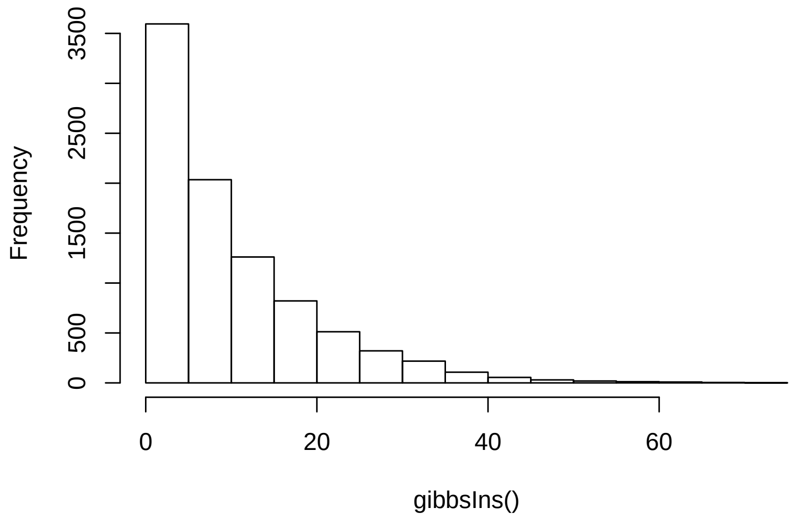
$$X|Y = y, N = n \sim \text{Bin}(n, y)$$

It can be shown that

$$
\begin{aligned}
&Y|X = x, N = n \sim \text{Beta}(x+\alpha, n-x+\beta)\\
&M|X = x, Y = y \sim \text{Pois}(\lambda(1-y))\\
&N = M + x
\end{aligned}
$$

So the Gibbs sampler is as follows:

```
gibbsIns <- function(n=1e4, alpha=1, beta=100, lambda=1000){
  X <- rep(0, n)
  Y <- rep(0, n)
  M <- rep(0, n)
  N <- rep(0, n)
  Y[1] <- alpha/(alpha+beta)
  X[1]<-Y[1]*lambda
  M[1] <- lambda
  N[1] <- ifelse(M[1]>0,M[1],1)
  for(i in 2:n){
    X[i] <- rbinom(1, N[i-1], Y[i-1])
    Y[i] <- rbeta(1, X[i]+alpha, N[i-1]-X[i]+beta)
    M[i] <- rpois(1, lambda*(1-Y[i]))
    N[i] <- M[i]+X[i]
  }
  X[-c(1:1000)]
}
hist(gibbsIns(), main="")
```



Example : One of the main uses of the Gibbs sampler is in Bayesian analysis. Say we have $X \sim \text{Bin}(n, p)$ and $p \sim \text{Beta}(\alpha, \beta)$ and we want a sample from the posterior distribution $p|X$. Then the joint distribution of X and p is the beta-binomial distribution given by

$$f(x,p) = C \cdot \binom{n}{x} p^{x+\alpha-1}(1-p)^{n-x+\beta-1}, x = 0,..,n, 0 \le p \le 1$$

To use the Gibbs sampler we need the conditional distributions of $X|p$ and $p|X$:

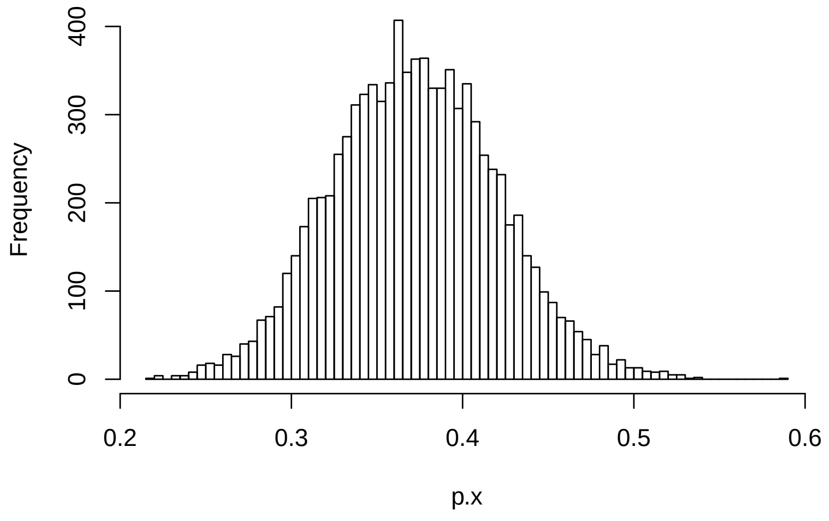
$$X|p \sim \text{Bin}(n,p)$$
$$p|x \sim \text{Beta}(x+\alpha, n-x+\beta)$$

so the Gibbs sampler is as follows:

```
gibbsBin <- function(x, n, B=1e4, alpha=1, beta=1, lambda=1){
  p <- rep(0.5, B)
  X <- rep(0, B)
  for(i in 2:B) {
    X[i] <- rbinom(1, n, p[i-1])
    p[i] <- rbeta(1, x+alpha, n-x+beta)
  }
  p[-c(1:1000)]
}
```

As a specific example, say in a sample of 100 employees of a company we have 37 women and 63 men, and we want to find a 90% interval estimate for the percentage of female employees. We have no prior knowledge of this company, so we will use U[0,1] (=beta(1,1)) as our prior.

```
p.x <- gibbsBin(x=37, n=100)
hist(p.x, 100, main="")
```



```r
round(quantile(p.x, c(0.025, 0.975)), 3)
```

```
##  2.5% 97.5%
## 0.282 0.469
```

Notice that the interval is very similar to the standard frequentist solution (Clopper-Pearson 1934 intervals)

```r
round(binom.test(x=37, n=100)$conf.int, 3)
```

```
## [1] 0.276 0.472
## attr(,"conf.level")
## [1] 0.95
```

Historically Bayesian analysis suffered from the problem that prior distributions had to be chosen so it was possible to calculate the posterior distribution (one popular choice are so-called conjugate priors, where the posterior distribution is the same as the prior one, except for the parameters) even though those priors were not a good description of our "prior belief". The Gibbs sampler allows us to be much more free in our choice of prior.

Example

say we want to generate data from the random vector (X, Y, Z) with density $f(x, y, z) = K(x + y + z)$, $0 < x < y < z < 1$.

To use the Gibbs Sampler we need all the conditional distributions:

$$
\begin{aligned}
&f_{Y,Z}(y,z) = \int_0^y K(x+y+z)dx = \\
&K(\frac{1}{2}x^2 + xy + xz|_0^y = \\
&K(\frac{1}{2}y^2 + y^2 + yz) = \\
&K(\frac{3}{2}y^2 + yz) \\
&f_{X|Y=y,Z=z}(x|y,z) = \frac{K(x+y+z)}{K(\frac{3}{2}y^2+yz)} = \\
&\frac{x+y+z}{\frac{3}{2}y^2+yz} \\
&0 < x < y
\end{aligned}
$$

Notice that the constant K vanishes in the conditional distribution. This will always happen, so we will ignore it from now on.

Next we find

$$
\begin{aligned}
&f_{X,Z}(x,z) = \int_x^z x+y+zdy = \\
&xy + \frac{1}{2}y^2 + yz|_x^z = \\
&xz + \frac{1}{2}z^2 + z^2 - x^2 - \frac{1}{2}x^2 - xz = \\
&\frac{3}{2}z^2 - \frac{3}{2}x^2 \\
&f_{Y|X=x,Z=z}(y|x,z) = \frac{x+y+z}{\frac{3}{2}z^2 - \frac{3}{2}x^2} \\
&x < y < z
\end{aligned}
$$

and finally

$$
\begin{aligned}
&f_{X,Y}(x,y) = \int_y^1 x+y+zdz = \\
&xz + yz + \frac{1}{2}z^2|_y^1 = \\
&x + y + \frac{1}{2} - xy - y^2 - \frac{1}{2}y^2 = \\
&x + y + \frac{1}{2} - xy - \frac{3}{2}y^2 \\
&f_{Z|X=x,Y=y}(z|x,y) = \frac{x+y+z}{x+y+\frac{1}{2} - xy - \frac{3}{2}y^2}
\end{aligned}
$$

Now that we have the marginals we need to be able to generate data from them. To do this notice that all three are linear functions of the form

$$g(x) = b(a+x) \text{ for } u < x < v$$

Now

$$
\begin{aligned}
&G(x) = \int_u^x b(a+t)dt = \\
&b(at + \frac{1}{2}t^2|_u^x = \\
&b(ax + \frac{1}{2}x^2 - au - \frac{1}{2}u^2)
\end{aligned}
$$

now we have $G(v) = 1$ and so

$$b = \frac{1}{a(v-u) + \frac{1}{2}(v^2 - u^2)}$$

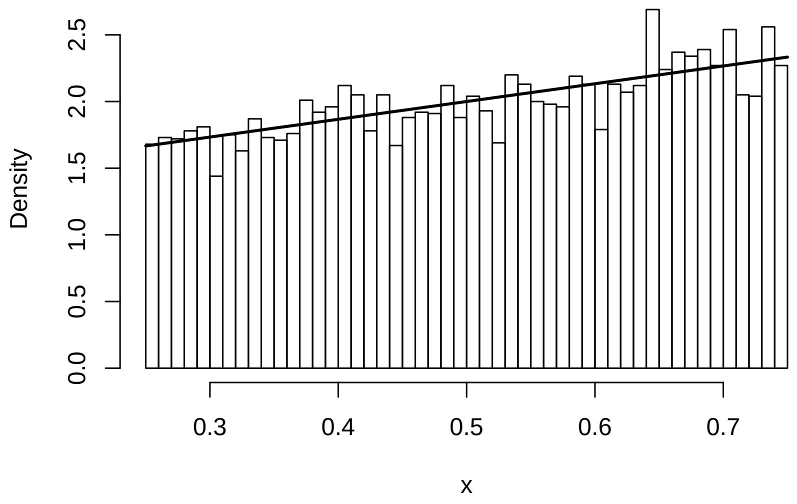
and we can find the inverse of G with

$$
\begin{aligned}
&G(x) = y \\
&b(ax + \frac{1}{2}x^2 - au - \frac{1}{2}u^2) = y \\
&\frac{1}{2}x^2 + ax - au - \frac{1}{2}u^2 - \frac{y}{b} = 0 \\
&x_{1,2} = -a \pm \sqrt{a^2 + 2(au + \frac{1}{2}u^2 + \frac{y}{b})}
\end{aligned}
$$

na now if $U \sim U[0,1]$ we have $G^{-1}(U)$ has this linear distribution.

Let's do a quick check to see whether this works (and that it is the $+$ in the quadratic formula!)

```
u <- 0.25
v <- 0.75
a <- 1
b <- 1/(a*(v-u)+(v^2-u^2)/2)
x <- (-a)+sqrt(a^2+2*(a*u+u^2/2+runif(1e4)/b))
hist(x, 50, freq=FALSE, main="")
curve(b*(a+x), u,v, add=TRUE, lwd=2)
```



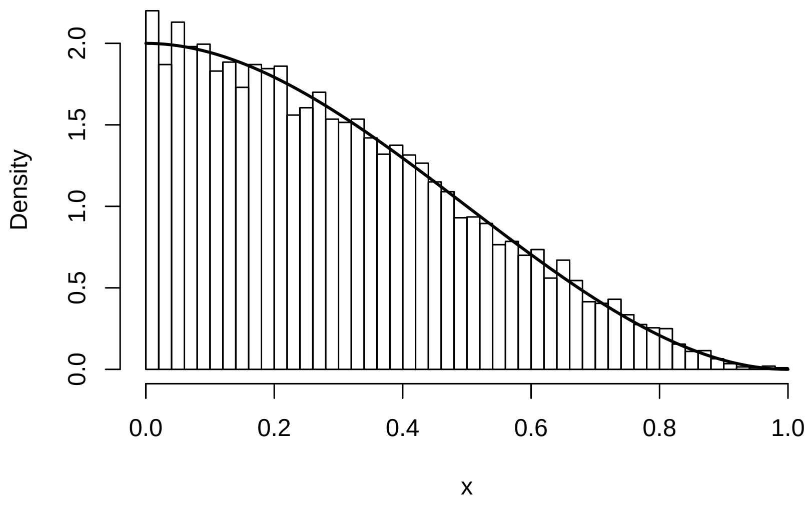
and now we can implement the Gibbs sampler:

```
n <- 1e4
x <- rep(0, n)
y <- rep(1/3, n)
z <- rep(2/3, n)
for(i in 2:n) {
  u <- 0
  v <- y[i-1]
  a <- y[i-1]+z[i-1]
  b <- 1/(a*(v-u)+(v^2-u^2)/2)
  x[i] <- (-a)+sqrt(a^2+2*(a*u+u^2/2+runif(1)/b))
  u <- x[i]
  v <- z[i-1]
  a <- x[i]+z[i-1]
  b <- 1/(a*(v-u)+(v^2-u^2)/2)
  y[i] <- (-a)+sqrt(a^2+2*(a*u+u^2/2+runif(1)/b))
  u <- y[i]
  v <- 1
  a <- x[i]+y[i]
  b <- 1/(a*(v-u)+(v^2-u^2)/2)
  z[i] <- (-a)+sqrt(a^2+2*(a*u+u^2/2+runif(1)/b))
}
```

Does this do the job? Let's check the marginal of X:

$$
\begin{aligned}
&f(x,z) = \frac{3}{2}(z^2 - x^2)\\
&0 < x < z < 1\\
&f(x) = \int_x^1 \frac{3}{2}(z^2 - x^2)dz =\\
&\frac{1}{2}z^3 - \frac{3}{2}x^2 z|_x^1 =\\
&\frac{1}{2} - \frac{3}{2}x^2 + x^3\\
&\int_0^1 \frac{1}{2} - \frac{3}{2}x^2 + x^3 dx =\\
&\frac{1}{2}x - \frac{1}{2}x^3 + \frac{1}{4}x^4|_0^1 =\\
&\frac{1}{2} - \frac{1}{2} + \frac{1}{4} = \frac{1}{4}
\end{aligned}
$$

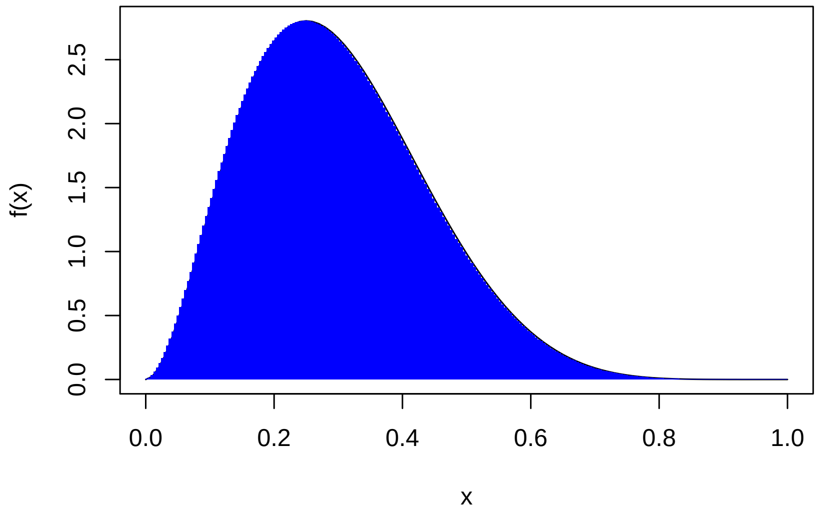
```
hist(x, 50, freq=FALSE, main="")
curve(4*(x^3-3/2*x^2+1/2), 0, 1, lwd=2, add=TRUE)
```



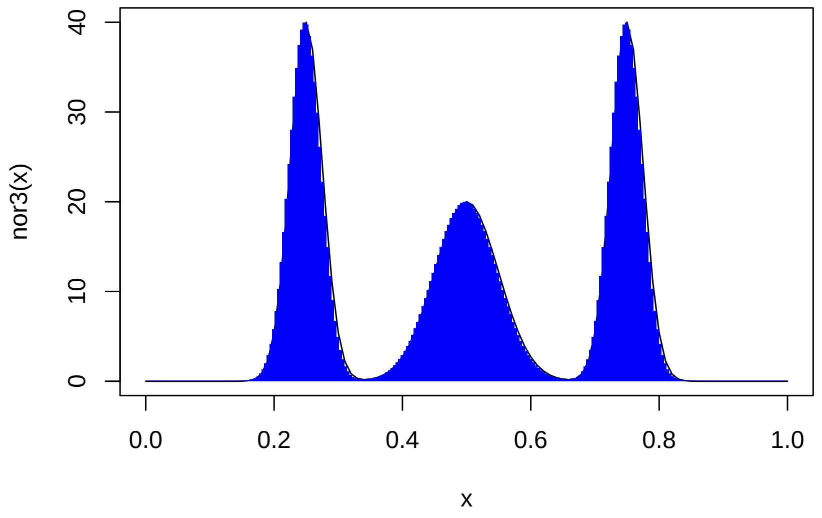
Looks good!

5.4 The Slice Sampler

In the discussion on the fundamental theorem of Simulation we had the following picture



and the idea was to generate a pair of uniforms in the rectangle and accept the y if $u < f(y)$. Now let's say that instead the density we want to simulate looks like this:



Clearly if we simply simulate pairs of uniforms most of the time the pair will be rejected, and the algorithm will be very inefficient. In light of the discussion on the Metropolis-Hastings algorithm is seems it would be a good idea to instead generate a Markov chain, and it is clear that it has to be a chain with a stationary distribution that is uniform on the rectangle.

A natural solution to this is to use a random walk process since they (almost always) result in uniform stationary distributions. Also an obvious way to construct the random walk is to alternate steps in the two directions, say first along the x axis, then the y axis, then the x axis and so on. Finally it turns out going in one direction we can always take steps of the same size. With this we have the following

Algorithm (2D slice sampler)
At iteration i, simulate

1) $u^{(i+1)} \sim U[0, f(x^{(i)})]$
2) $x^{(i+1)} \sim U[A^{(i+1)}]$ where $A^{(i+1)} = \left\{x : f(x) \geq u^{(i+1)}\right\}$

As before with the Metropolis-Hastings algorithm, f need not be normalize.

The hard part of this algorithm is the solution of the inequality $u \leq f(x)$. Of course if f has in inverse this is simple:

####**Example** we want to simulate data from

$$f(x) = \frac{1}{2}\exp(-\sqrt{x}) \ , \ x > 0$$

Now

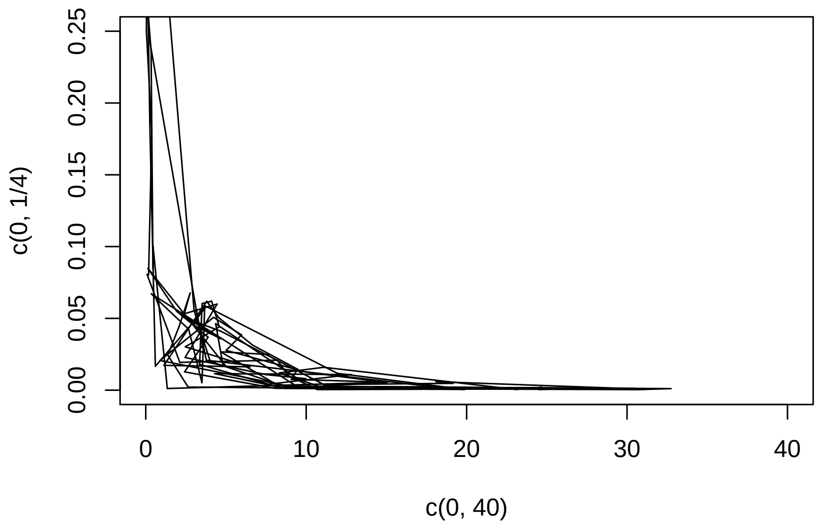
$$
\begin{aligned}
y &= \frac{1}{2}\exp(-\sqrt{x}) \\
2y &= \exp(-\sqrt{x}) \\
\log(2y) &= -\sqrt{x} \\
x &= \log(2y)^2
\end{aligned}
$$

so we have

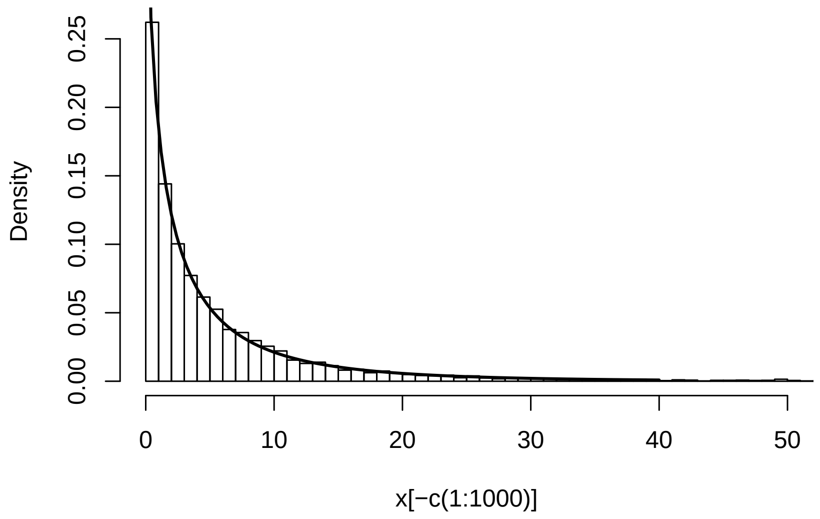
1) $u^{(i+1)} \sim U[0, \frac{1}{2}\, exp(-\sqrt{x}^{(i)}))]$

2) $x^{(i+1)} \sim U[0, log(2u^{(i+1)})^2]$

```r
B <- 1e4
x <- runif(B, 0, 1)
u <- runif(B, 0, 1)
plot(c(0, 40), c(0, 1/4), type="n")

for(i in 2:B) {
  u[i] <- runif(1, 0, 1/2*exp(-sqrt(x[i-1])))
  x[i] <- runif(1, 0, log(2*u[i])^2)
  if(i<100) {
    segments(x[i-1], u[i-1], x[i], u[i])
  }
}
```



```r
hist(x[-c(1:1000)], 100, freq=FALSE, main="", xlim = c(0,50))
curve(0.5*exp(-sqrt(x)), 0, 40, add = TRUE, lwd=2)
```



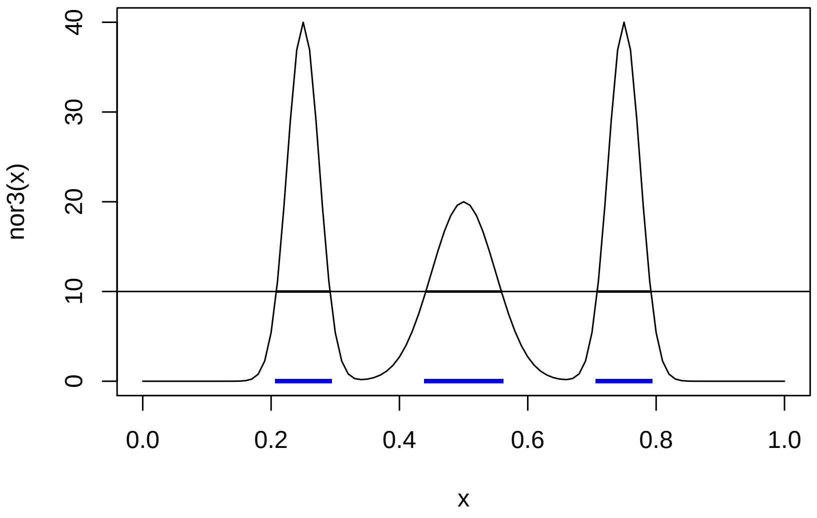
If it is not possible to calculate f^{-1}, maybe because f is multi-modal, we can try a kind of numerical inversion :

####**Example** the function shown at the beginning of this chapter is actually the following:

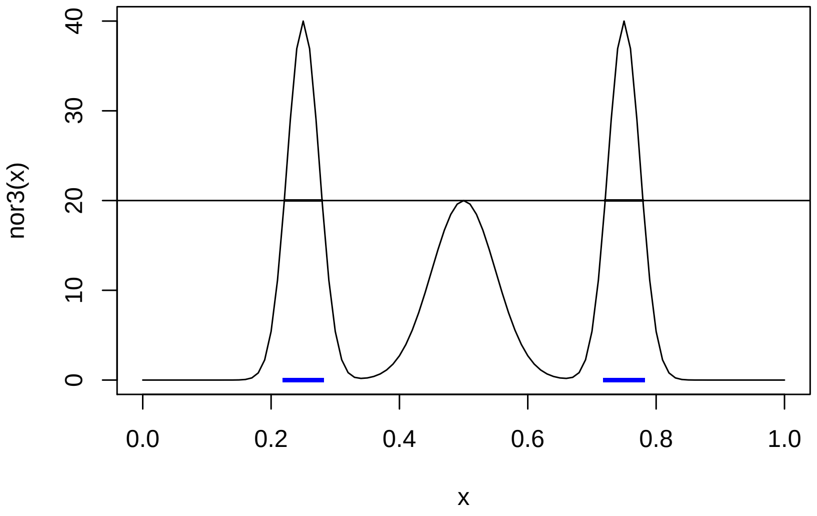
$$\begin{aligned}\text{nor3}(x) = &\exp(-(x-0.25)^2/2/0.025^2)/0.025+\\ &\exp(-(x-0.5)^2/2/0.05^2)/0.05+\\ &\exp(-(x-0.75)^2/2/0.025^2)/0.025\end{aligned}$$

Let's use the 2D slice sampler to simulate from this curve. Say in step 1) we picked $u^{(i+1)} = 10$, then in step 2) we are supposed to pick a point uniformly from the blue set:

```r
y <- 10
x <- seq(0, 1, length=1000)
fx <- nor3(x)
xinf <- x[fx>y]
curve(nor3, 0, 1)
abline(h=y)
points(xinf, rep(y, length(xinf)), pch=".")
points(xinf, rep(0, length(xinf)), pch=".",
       cex=3, col="blue")
```

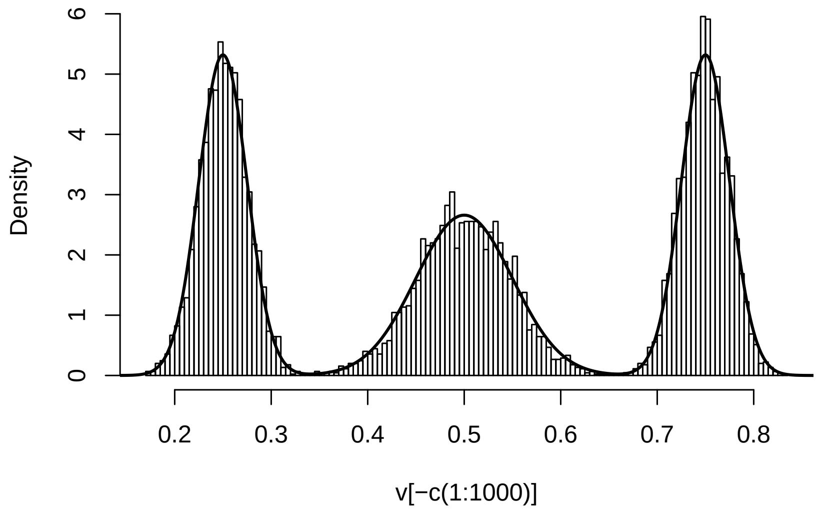


```
y <- 20
xinf <- x[fx>y]
curve(nor3, 0, 1)
abline(h=y)
points(xinf, rep(y, length(xinf)), pch=".")
points(xinf, rep(0, length(xinf)), pch=".",
       cex=3, col="blue")
```



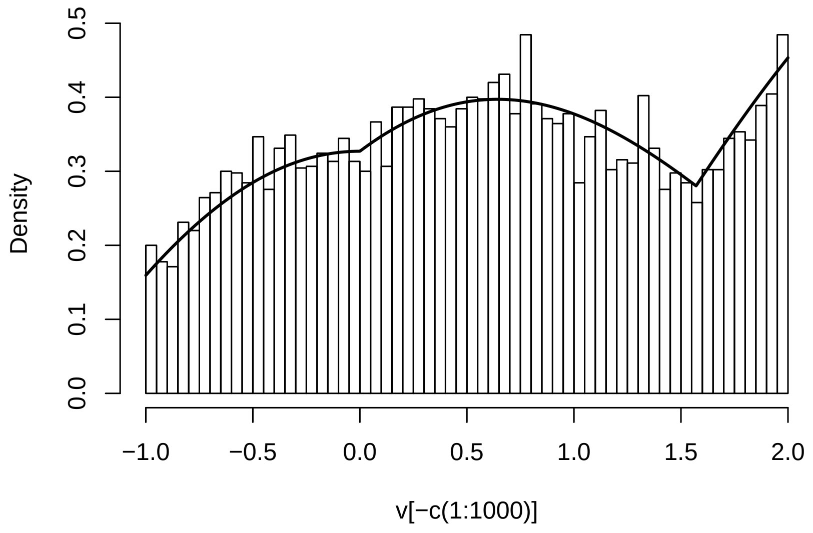
As long as calculating the function values is cheap, we can just do that numerically by calculating y's on a grid of x values, and randomly selecting those x values with $y > u$.

```
B <- 1e4
v <- rep(x[500], B)
u <- rep(y[500], B)
for(i in 2:B) {
  u[i] <- runif(1, 0, nor3(v[i-1]))
  xinf <- x[fx>u[i]]
  v[i] <- sample(xinf, 1)
}
hist(v[-c(1:1000)], 100, freq=FALSE, main="")
I <- integrate(nor3, 0, 1)$value
lines(x, fx/I, lwd=2)
```



And now we have basically a general event generator in one dimension:

```
f <- function(x) abs(sin(x))+3*abs(cos(x))+x
x <- seq(-1, 2, length=1000)
fx <- f(x)
v <- rep(x[500], B)
u <- rep(y[500], B)
for(i in 2:B) {
  u[i] <- runif(1, 0, f(v[i-1]))
  xinf <- x[fx>u[i]]
  v[i] <- sample(xinf, 1)
}
hist(v[-c(1:1000)], 100, freq=FALSE, main="")
I <- integrate(f, -1, 2)$value
lines(x, fx/I, lwd=2)
```



The problem of having to find an inverse can sometimes be overcome by a generalization of the 2D slice sampler. Say we want to sample from a density f which can be written as

$$f(x) = \prod_{i=1}^{k} f_i(x)$$

where the $f_i(x)$ need not be densities. Then we have the general

Algorithm (Slice sampler)

At iteration i, simulate

1: $w_1^{(i+1)} \sim U[0, f_1(x^{(i)})]$

...
k: $w_1^{(k+1)} \sim U[0, f_k(x^{(k)})]$

k+1: $x^{(i+1)} \sim U[A^{(i+1)}]$

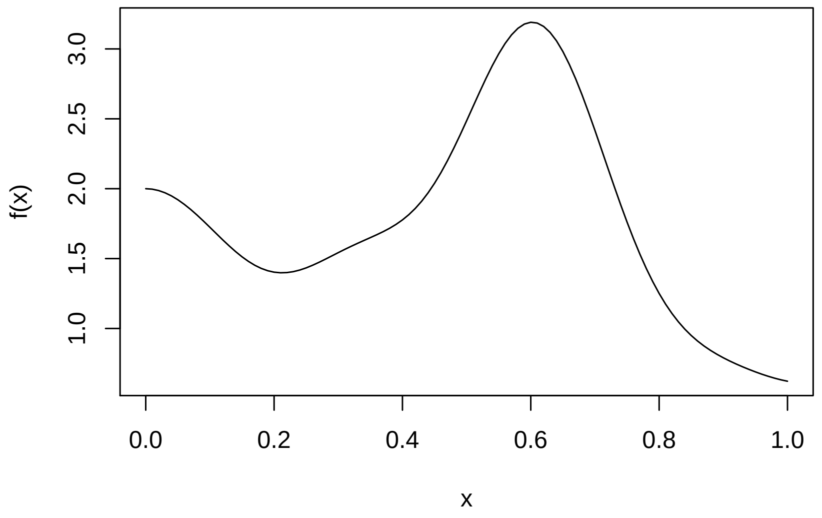
where

$A^{(i+1)} = \left\{ y : f_j(y) \geq w_j^{(i+1)}; j = 1, .., k \right\}$

Example: say

$$f(x) = (1 + \sin(3x)^2)(1 + \cos(5x)^4) exp(-x^2/2)$$

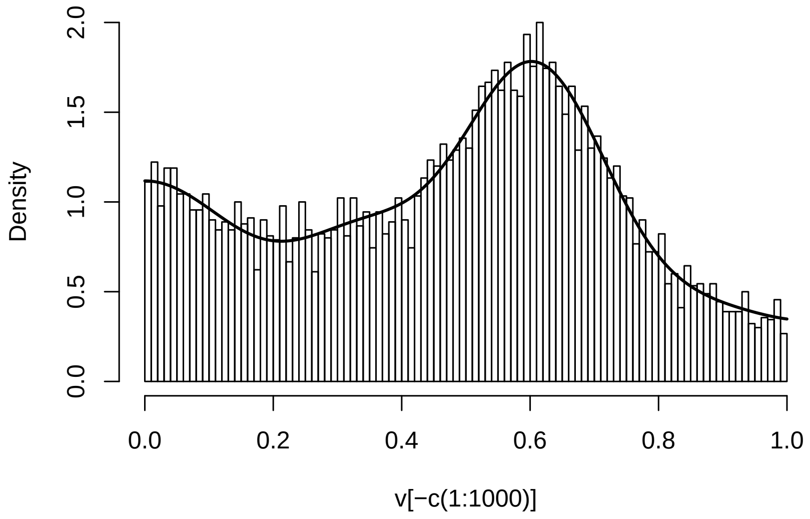
```
f <- function(x) (1+ sin(3*x)^2)*(1+cos(5*x)^4)*exp(-x^2/2)
curve(f, 0, 1)
```



now

```r
x <- seq(0, 1, length=1000)
f1 <- function(x) 1+sin(3*x)^2
fx1 <- f1(x)
w1 <- rep(0.5, B)
f2 <- function(x)  1+cos(5*x)^4
fx2 <- f2(x)
w2 <- rep(0.5, B)
f3 <- function(x)  exp(-x^2/2)
fx3 <- f3(x)
w3 <- rep(0.5, B)
v <- rep(x[500], B)

for(i in 2:B) {
  w1[i] <- runif(1, 0, f1(v[i-1]))
  w2[i] <- runif(1, 0, f2(v[i-1]))
  w3[i] <- runif(1, 0, f3(v[i-1]))
  tmpfx2 <- fx2[ fx1 > w1[i] ]
  tmpfx3 <- fx3[ fx1 > w1[i] ]
  xinf <- x[ fx1 > w1[i] ]
  tmpfx3 <- tmpfx3[ tmpfx2 > w2[i] ]
  xinf <- xinf[tmpfx2 > w2[i]]
  xinf <- xinf[tmpfx3 > w3[i]]

  v[i] <- sample(xinf, 1)
}
hist(v[-c(1:1000)], 100, freq=FALSE, main="")
```

```
I <- integrate(f, 0, 1)$value
lines(x, fx1*fx2*fx3/I, lwd=2)
```



This version is of course especially useful if we want to simulate from a Bayesian posterior distribution:

Example: say we have observations $X_1, .., X_n$ from an exponential distribution with rate λ. We want to estimate λ as the mean of the posterior distribution. If we use as a prior $\pi(\lambda) \sim 1/\lambda$, $\lambda > 0$ we find

$$f(x_1,..,x_n,\lambda) = \left[\prod_{i=1}^{n} \lambda \exp(-\lambda x_i)\right]\frac{1}{\lambda} =$$

$$\lambda^{n-1}\exp(-\lambda S)$$

where $S = \sum_{i=1}^{n} X_i$

$$m(\mathbf{x}) = \int_0^\infty \lambda^{n-1}\exp(-\lambda S)d\lambda =$$

$$\frac{\Gamma(n)}{S^n}\int_0^\infty \frac{S^n}{\Gamma(n)}\lambda^{n-1}\exp(-S\lambda)d\lambda = \frac{\Gamma(n)}{S^n}$$

$$f(\lambda|\mathbf{x}) = \frac{f(\mathbf{x},\lambda)}{m(\mathbf{x})} = \frac{S^n\lambda^{n-1}\exp(-\lambda S)}{\Gamma(n)}$$

$$\lambda|\mathbf{x}\sim\Gamma(n,S)$$

$$E[\lambda|\mathbf{x}] = \frac{n}{S} = \frac{1}{\bar{X}}$$

but what if we want to use $\pi(\lambda) \sim \lambda/(\lambda+1)^2$? Now m(x) can not be calculated but we can simulate from the posterior distribution:

$$f(t) = t^n \exp(-St)t/(t+1)^2$$

Let

$$f_1(t) = t^n \exp(-St) f_2(t) = t/(t+1)^2$$

```
n <- 50
x.sample <- rexp(n, 2)
S <- sum(x.sample)
x <- seq(0, 5, length=1000)
```

```
f1 <- function(x) x^n*exp(-S*x)
fx1 <- f1(x)
w1 <- rep(mean(x.sample), B)
f2 <- function(x)  x/(x+1)^2
fx2 <- f2(x)
w2 <- rep(mean(x.sample), B)
v <- rep(mean(x.sample), B)

for(i in 2:B) {
  w1[i] <- runif(1, 0, f1(v[i-1]))
  w2[i] <- runif(1, 0, f2(v[i-1]))
  tmpfx2 <- fx2[ fx1 > w1[i] ]
  xinf <- x[ fx1 > w1[i] ]
  xinf <- xinf[tmpfx2 > w2[i]]
  v[i] <- sample(xinf, 1)
}
out <- round( quantile(v[-c(1:1000)], c(0.025, 0.975)), 2)
cat("Bayesian 95% credible interval for lambda: (", out[1], ", ", out[2], ")\n")
```

```
## Bayesian 95% credible interval for lambda: ( 1.79 ,  3.1 )
```

Example: let's consider the normal mixture model, that is if $\phi(x;\mu,\sigma)$ denotes the normal density with mean μ and standard deviation σ, and if $0\le\alpha\le 1$, then

$$\begin{gathered}f(x;\alpha,mu_1,\sigma_1,\mu_2,\sigma_2)=\\ \alpha\phi(x,\mu_1,\sigma_1)+(1-\alpha)\phi(x;\mu_2,\sigma_2)\end{gathered}$$

Let's say we have a sample $X_1,..,X_n$ from f and we want to find a 90% credible interval for α. As priors we will use flat priors on α, μ_1 and μ_2, and $g(x)\sim 1/x$ for σ_1 and σ_2.

with this we find the posterior distribution to be

$$\begin{gathered}f(\alpha,mu_1,\sigma_1,\mu_2,\sigma_2;\mathbf{x})=\\ \prod_1^n\left(\alpha\phi(x_i,\mu_1,\sigma_1)+(1-\alpha)\phi(x_i;\mu_2,\sigma_2)\right)\frac{1}{\sigma_1\sigma_2}\end{gathered}$$

so we need to sample from this density. Now the obvious choice is to use

$$\begin{aligned}&f_i(\alpha,mu_1,\sigma_1,\mu_2,\sigma_2;\mathbf{x})=\\ &\alpha\phi(x_i,\mu_1,\sigma_1)+(1-\alpha)\phi(x_i;\mu_2,\sigma_2)\\ &i=1,..,n\\ &f_{n+1}(\alpha,mu_1,\sigma_1,\mu_2,\sigma_2;\mathbf{x})=\frac{1}{\sigma_1}\\ &f_{n+2}(\alpha,mu_1,\sigma_1,\mu_2,\sigma_2;\mathbf{x})=\frac{1}{\sigma_2}\end{aligned}$$

if n is large, clearly this will be very slow!

5.5 Case Study: Bayesian Inference for a Normal Distribution

Say we have a sample $\boldsymbol{X} = (X_1, .., X_n)$ with $X_i \sim N(\mu, \sigma)$.

For now we assume σ is known. We want to find a 95% Bayesian credible interval for μ with the prior distribution $\pi(\mu)$.

$$
\begin{aligned}
&f(\mathbf{x}, \mu) = \prod \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x_i-\mu)^2}{2\sigma^2}\right\}\pi(\mu) = \\
&(2\pi\sigma^2)^{-n/2} \exp\left\{-\frac{1}{2\sigma^2}\sum(x_i-\mu)^2\right\}\pi(\mu)
\end{aligned}
$$

Now for the usual arithmetic:

$$
\begin{aligned}
&\sum(x_i-\mu)^2 = \\
&\sum(x_i-\overline{x}+\overline{x}-\mu)^2 = \\
&\sum(x_i-\overline{x})^2 + 2\sum(x_i-\overline{x})(\overline{x}-\mu) + \sum(\overline{x}-\mu)^2 = \\
&\sum(x_i-\overline{x})^2 + n(\overline{x}-\mu)^2
\end{aligned}
$$

so we have

$$
f(\mathbf{x}, \mu) = K \exp\left\{-\frac{n}{2\sigma^2}(\overline{x}-\mu)^2\right\}\pi(\mu)
$$

and we see that inference for μ can be based on the sample mean.

Let's first say we have $\pi(\mu) = 1$, then the posterior distribution of $\mu|\mathbf{X} = \mathbf{x}$ is found by

$$
\begin{aligned}
&\pi_{\mu|\overline{X}=x}(\mu|x) = \frac{f(\mathbf{x},\mu)}{m(\mathbf{x})} \\
&m(\mathbf{x}) = \int_{-\infty}^{\infty} \sqrt{\frac{n}{2\pi\sigma^2}} \exp\left\{-\frac{n}{2\sigma^2}(x-\mu)^2\right\} d\mu = \\
&\int_{-\infty}^{\infty} \sqrt{\frac{n}{2\pi\sigma^2}} \exp\left\{-\frac{n}{2\sigma^2}(\mu-x)^2\right\} d\mu = 1 \\
&\pi_{\mu|\overline{X}=x}(\mu|x) = \sqrt{\frac{n}{2\pi\sigma^2}} \exp\left\{-\frac{n}{2\sigma^2}(\mu-x)^2\right\}
\end{aligned}
$$

and so $\mu|\overline{X} = x \sim N(x, 1/\sqrt{n})$.

As an example throughout this section consider the following data set:

```
x.sample <-
c(-7.2, -2.72, -2.61, -1.87, -1.84, -1.17, -0.89, -0.33, -0.07,
0.26, 0.32, 0.51, 0.71, 0.85, 0.89, 1, 1.16, 1.3, 1.84, 2.8,
3.08, 3.18, 3.53, 4.14, 4.15)
n <- length(x.sample)
```

Let's say we know $\sigma = 2$, then

```
B <- 1e4
sigma <- 2
samplemean.x <- mean(x.sample)
out <- round(samplemean.x + c(-1,1)*qnorm(0.95)*sigma/sqrt(n), 2)
cat("Frequentist Confidence Interval:  (", out[1],
    " ," ,out[2],")\n")
```

```
## Frequentist Confidence Interval:  ( -0.22  , 1.1 )
```

```
out <- round(qnorm(c(0.025, 0.975),
                   samplemean.x, sigma/sqrt(n)), 2)
cat("Bayesian Credible Interval, exact calculation: (",
    out[1], " ," ,out[2],")\n")
```

```
## Bayesian Credible Interval, exact calculation: ( -0.34  , 1.22 )
```

```
sample.post <- rnorm(B, samplemean.x, sigma/sqrt(n))
out <-round(quantile(sample.post, c(0.025, 0.975)), 2)
cat("Bayesian Credible Interval, direct simulation: (",
    out[1], " ," ,out[2],")\n")
```

```
## Bayesian Credible Interval, direct simulation: ( -0.35  , 1.23 )
```

Let's do the simulation using the Metropolis-Hastings algorithm. This means we want to sample from

$$g(\mu) = \exp(-\frac{1}{2\sigma^2}(\mu - \overline{x})^2)$$

let's use as a proposal distribution $q_{xy} \sim U[y-1, y+1]$, then we have

```
fun <- function(old.mu, new.mu)
  dnorm(new.mu, samplemean.x, sigma/sqrt(n))/dnorm(old.mu, samplemean.x, sigma/sqrt(n))
mu.x <- rep(samplemean.x, B)
for(i in 2:B) {
  new.mu <- runif(1, mu.x[i-1]-1,mu.x[i-1]+1)
  if(runif(1)<fun(mu.x[i-1], new.mu)) mu.x[i] <- new.mu
  else mu.x[i] <- mu.x[i-1]
}
out <-round(quantile(mu.x[-c(1:1000)], c(0.025, 0.975)), 2)
cat("Metropolis-Hastings: (", out[1], " ," ,out[2],")\n")
```

```
## Metropolis-Hastings: ( -0.36  , 1.25 )
```

If we wanted to use the Gibbs sample, what would that mean? We again need the marginals, but in fact we already have them:

$$\overline{X}|\mu \sim N(\mu, \sigma/\sqrt{n})$$
$$\mu|\overline{X} = x \sim N(x, \sigma/\sqrt{n})$$

and now the simulation of the posterior can be done with:

```
x.mu <- rep(1, B)
mu.x <- rep(1, B)
for(i in 2:B) {
  mu.x[i] <- rnorm(1, samplemean.x, sigma/sqrt(n))
  x.mu[i] <- rnorm(1, mu.x[i], sigma/sqrt(n))
}
out <- round(quantile(mu.x[-c(1:1000)], c(0.025, 0.975)), 2)
cat("Gibbs Sampler: (", out[1], " ," ,out[2],")\n")
```

```
## Gibbs Sampler: ( -0.36  , 1.24 )
```

Now let's say we also know that $\mu > 0$. How can we include that information in our analysis?

For a Frequentist this is rather difficult (though not impossible). For a Bayesian it is easy: all we need to do is use a prior that has support on the positive numbers, for example $\pi(\mu) = I_{(0,\infty)}(\mu)$. Now the joint distribution becomes

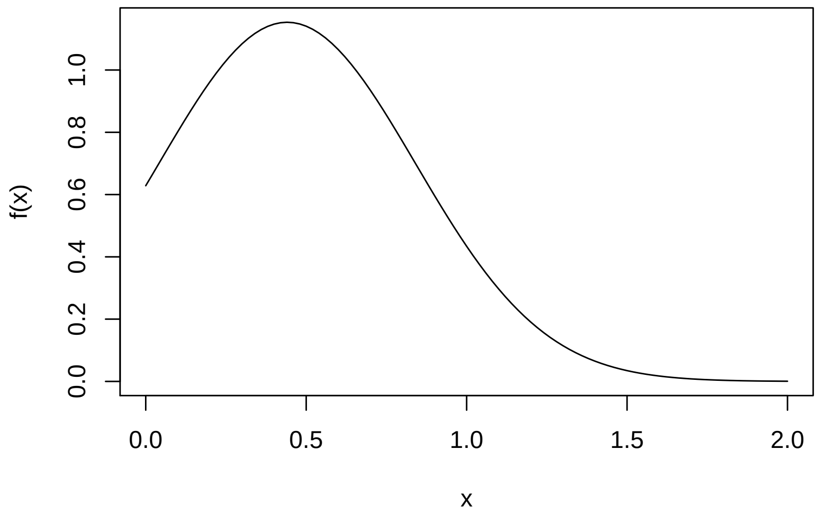
$$f(\mathbf{x}, \mu) = K \exp\left\{-\frac{n}{2\sigma^2}(\overline{x} - \mu)^2\right\} I_{(0,\infty)}(\mu)$$

Doing it analytically means finding the marginal:

$$m(\mathbf{x}) = \int_0^\infty K \exp\left\{-\frac{n}{2\sigma^2}(\overline{x} - \mu)^2\right\}$$

which is not possible, so we have to proceed numerically. For the cdf we could also use the *integrate* command. Here I do a simple numerical integration based on Riemann sums.

```
m.x <- 1
f <- function(mu) dnorm(mu, samplemean.x, sigma/sqrt(n))/m.x
m.x <- integrate(f, 0, Inf)$value
curve(f, 0, 2)
```



```
t <- seq(0, 2, length=1000)
ft <- f(t)
Ft <- cumsum(ft)*(t[2]-t[1])
Left <- t[abs(Ft-0.025)==min(abs(Ft-0.025))]
Right <- t[abs(Ft-0.975)==min(abs(Ft-0.975))]
out <- round(c(Left, Right), 4)
cat("Positive mu, numerically: (", out[1], " ," ,out[2],")\n")
```

```
## Positive mu, numerically: ( 0.036  , 1.2432 )
```

How about using Metropolis-Hastings? In fact the same algorithm as above works fine, except we need to change the proposal distribution so it only allows positive values:

```
fun <- function(old.mu, new.mu)
  dnorm(new.mu, samplemean.x, sigma/sqrt(n))/dnorm(old.mu, samplemean.x, sigma/sqrt(n))
mu.x <- rep(samplemean.x, B)
for(i in 2:B) {
  new.mu <- runif(1, max(0, mu.x[i-1]-1),mu.x[i-1]+1)
  if(runif(1)<fun(mu.x[i-1], new.mu)) mu.x[i] <- new.mu
  else mu.x[i] <- mu.x[i-1]
}
out <-round(quantile(mu.x[-c(1:1000)], c(0.025, 0.975)), 2)
cat("Metropolis-Hastings: (", out[1], " ," ,out[2],")\n")
```

```
## Metropolis-Hastings: ( 0.05  , 1.3 )
```

How about the Gibbs sampler? Again we will need the marginals, but this time they can not be found analytically.

Next we will consider the case of an unknown standard deviation. We then will need a prior on σ as well. We will use $\sigma \sim 1/\sigma^2$. With this we find

$$
\begin{aligned}
&f(\mathbf{x}, \mu, \sigma) = \\
&(2\pi\sigma^2)^{-n/2} \exp\left\{-\frac{1}{2\sigma^2}\sum(x_i-\mu)^2\right\}\pi(\mu)\frac{1}{\sigma^2} = \\
&(2\pi)^{-n/2}\sigma^{-n-2} \exp\left\{-\frac{1}{2\sigma^2}\left(\sum(x_i-\overline{x})^2 + n(\overline{x}-\mu)^2\right)\right\}\pi(\mu) = \\
&(2\pi)^{-n/2}\sigma^{-n-2} \exp\left\{-\frac{1}{2\sigma^2}\left((n-1)s^2 + n(\overline{x}-\mu)^2\right)\right\}\pi(\mu) = \\
&(2\pi)^{-n/2}\sigma^{-n-2} \exp\left\{-\frac{(n-1)s^2}{2\sigma^2}\right\}\exp\left\{-\frac{n(\overline{x}-\mu)^2}{2\sigma^2}\right\}\pi(\mu)
\end{aligned}
$$

Again we start with $\pi(\mu) = 1$. For the MH algorithm we now need also a proposal distribution for σ:

```r
S2 <- var(x.sample)
f1 <- function(x) exp(-(n-1)*S2/2/x)/x^(n/2+1)
f2 <- function(x, a) dnorm(samplemean.x, x, sqrt(a/n))
fun <- function(old, new)
    f1(new[2])*f2(new[1], new[2])/(f1(old[2])*f2(old[1], old[2]))
mu.x <- rep(samplemean.x, B)
sigma.x <- rep(sd(x.sample), B)
new <- rep(0, 2)
for(i in 2:B) {
  new[1] <- runif(1, mu.x[i-1]-1, mu.x[i-1]+1)
  new[2] <- runif(1, max(0, sigma.x[i-1]-1), sigma.x[i-1]+1)
  if(runif(1)<fun(c(mu.x[i-1], sigma.x[i-1]), new)) {
      mu.x[i] <- new[1]
      sigma.x[i] <- new[2]
  }
  else {
      mu.x[i] <- mu.x[i-1]
      sigma.x[i] <- sigma.x[i-1]
  }
}
out <-round(quantile(mu.x[-c(1:1000)], c(0.025, 0.975)), 2)
cat("Metropolis-Hastings: (", out[1], " ," ,out[2],")\n")
```

```
## Metropolis-Hastings: ( -0.53  , 1.43 )
```

notice that we are working here with the variance σ^2 as the parameter, not the standard deviation σ.

For the Gibbs sampler we find

$$f(\mu|\mathbf{x},\sigma^2) = K\exp\left\{-\frac{n(\overline{x}-\mu)^2}{2\sigma^2}\right\}$$
$$f(\sigma^2|\mathbf{x},\mu) = K(\sigma^2)^{-n/2-1}\exp\left\{-\frac{(n-1)s^2}{2\sigma^2}\right\}$$

$$f(\sigma|\mathbf{x},\mu) = K(\sigma^2)^{-n/2-1}\exp\left\{-\frac{(n-1)s^2}{2\sigma^2}\right\}$$

What is this second density? It is called a scaled inverse chisquare distribution with n df and scale nS^2:

$$f(x) = (n/2)^{n/2}/(\Gamma(n/2))S^n(1/x)^{(n/2)+1}\exp[-(nS^2)/(2x)]$$

the routine *rinvchisq* is part of the *geoR* package. So now we can use the Gibbs Sampler:

```r
library(geoR)
mu.var<- rep(1, B)
var.mu <- rep(4, B)
for(i in 2:B) {
  mu.var[i] <- rnorm(1, samplemean.x, sqrt(var.mu[i-1]/n))
  var.mu[i] <- rinvchisq(1, df=n-1, scale=S2)
}
out <- round(quantile(mu.var[-c(1:1000)], c(0.025, 0.975)), 2)
cat("Gibbs Sampler: (", out[1], " ," ,out[2],")\n")
```

```
## Gibbs Sampler: ( -0.58  , 1.5 )
```

Finally again the case $\pi > 0$:

```r
mu.x <- rep(samplemean.x, B)
sigma.x <- rep(sd(x.sample), B)
new <- rep(0, 2)
for(i in 2:B) {
  new[1] <- runif(1, max(0, mu.x[i-1]-1), mu.x[i-1]+1)
  new[2] <- runif(1, max(0, sigma.x[i-1]-1), sigma.x[i-1]+1)
  if(runif(1)<fun(c(mu.x[i-1], sigma.x[i-1]), new)) {
      mu.x[i] <- new[1]
      sigma.x[i] <- new[2]
  }
  else {
      mu.x[i] <- mu.x[i-1]
      sigma.x[i] <- sigma.x[i-1]
  }
}
out <-round(quantile(mu.x[-c(1:1000)], c(0.025, 0.975)), 2)
cat("Metropolis-Hastings: (", out[1], " ," ,out[2],")\n")
```

```
## Metropolis-Hastings: ( 0.06  , 1.59 )
```

6 Variance Reduction Methods

6.1 Antithetic and Control Variables

6.1.1 Antithetic Variables

Suppose we want to estimate a parameter $\theta = E[X]$ and suppose we have generated X_1 and X_2, two rv's with mean θ. Then

$$\begin{aligned} &Var(\overline{X}) = Var\left(\frac{X_1 + X_2}{2}\right) = \\ &\frac{1}{4}Var(X_1) + \frac{1}{4}Var(X_2) + \frac{1}{2}Cov(X_1, X_2) \end{aligned}$$

Obviously we want to do this so that we minimize our simulation error, that is the variance of our estimator. If we generate X_1 and X_2 independently we have $Cov(X_1, X_2) = 0$, but we could do even better if we generated them so that $Cov(X_1, X_2) < 0$

####**Example** (Estimating an Integral) Let's say we want to use simulation to estimate the integral

$$\int_0^1 e^x dx$$

(not that we need simulation to do this, but it illustrates the idea). Now

$$\int_0^1 e^x dx = E[U] = \theta$$

where $U \sim U[0, 1]$.

Now the straightforward approach would be to generate U_1, U_2 iid $U[0, 1]$ and to estimate the integral by $(\exp(U_1) + \exp(U_2))/2$. This would have variance

$$Var(\frac{1}{2}\left(e^{U_1} + e^{U_2}\right))$$

Now

$$\begin{aligned} E[e^U] &= \int_0^1 e^x dx = e - 1 \\ E[e^{2U}] &= \int_0^1 e^{2x} dx = \frac{1}{2}(e^2 - 1) \\ Var(e^U) &= \frac{1}{2}(e^2 - 1) - (e-1)^2 = 0.242 \\ Var(\frac{1}{2}\left(e^{U_1} + e^{U_2}\right)) &= \frac{1}{2}Var(e^U) = 0.121 \end{aligned}$$

How can we generate X_1 and X_2 so that $E[X_1] = E[X_2] = \theta$ and $Cov(X_1, X_2) < 0$? One idea (due to Rubinstein) is to use

$$X_1 = \exp(U) X_2 = \exp(1 - U)$$

Of course $1 - U \sim U[0, 1]$, so $E[X_1] = E[X_2] = \theta$.

Now

$$
\begin{aligned}
&Cov(X_1, X_2) = Cov(e^U, e^{1-U}) = \\
&E[e^U e^{1-U}[-E[e^U]E[e^{1-U}] = \\
&e - (e-1)^2 = -0.234
\end{aligned}
$$

and so

$$
\begin{aligned}
&Var\left(\frac{e^U + e^{1-U}}{2}\right) = \\
&\frac{1}{4}Var(e^U) + \frac{1}{4}Var(e^{1-U}) + \frac{1}{2}Cov(e^U, e^{1-U}) = \\
&0.242 + 0.242 + \frac{1}{2}(-0.234) = 0.0039
\end{aligned}
$$

Let's use simulation to verify this:

```
B <- 1e5
U <- runif(B)
out <- round( var(exp(U)), 4)
cat("Direct Simulation: ", out,"\n")
```

```
## Direct Simulation:  0.2427
```

```
U2 <- runif(B)
out <- round( var((exp(U)+exp(U2))/2), 4)
cat("Averaging: ", out,"\n")
```

```
## Averaging:  0.1214
```

```
out <- round( var((exp(U)+exp(1-U))/2), 4)
cat("Rubinstein: ", out,"\n")
```

```
## Rubinstein:  0.0039
```

####**Example** say we want to find

$$
I = \int_0^{\pi/4} \int_0^{\pi/4} x^2 y^2 \sin(x+y) \log(x+y) dx dy
$$

We can use R and the *integrate* routine:

```
dblInt <-
function(f, low = c(0, 0), high = c(Inf, Inf))
{
    integrate(function(y) {
        sapply(y, function(y) {
            integrate(function(x) f(x, y), low[1],
                      high[1])$value
        })
    }, low[2], high[2])$value
}
```

```r
f <- function(x, y) {
        x^2*y^2*sin(x+y)*log(x+y)
}
out <- round(dblInt(f, c(0, 0), c(pi,pi)/4), 4)
cat("Numeric Integration: ", out,"\n")
```

```
## Numeric Integration:  0
```

Now to use simulation we need to write the integral as in terms of expected values:

$$
\begin{aligned}
&f(x,y) = x^2y^2\sin(x+y)\log(x+y)dxdy \\
&I = \left(\frac{\pi}{2}\right)^2 \int_0^{\pi/4}\int_0^{\pi/4} f(x,y)\frac{4}{\pi}\frac{4}{\pi}dxdy = \\
&\left(\frac{\pi}{2}\right)^2 E[f(\frac{\pi}{4}U, \frac{\pi}{4}V)]
\end{aligned}
$$

```r
k <- (pi/4)^2
u <- runif(B)
v <- runif(B)
z <- k * f(pi * u/4, pi * v/4)
out <- round(c(mean(z), sd(z)), 4)
cat("Standard Simulation: ", out,"\n")
```

```
## Standard Simulation:  0 0
```

```r
u <- runif(B/2)
v <- runif(B/2)
z <- k * (f(pi * u/4, pi * v/4) + f(pi * (1 - u)/4, pi * (1 - v)/4))/2
out <- round(c(mean(z), sd(z)), 4)
cat("Antithetic Simulation: ", out,"\n")
```

```
## Antithetic Simulation:  0 0
```

Until now we use 1-U as an antithetic variable. There are other options, however

####**Example** say $X \sim N(10,1)$ and we want to find $E[\log(X)]$.

Obviously we can generate rnorm(n, 10, 1) and then find mean(x) but notice that if $X \sim N(\mu,\sigma)$, then $Y = 2\mu - X \sim N(\mu,\sigma)$, so Y is an antithetic variable again and

$$
\begin{aligned}
&Cov(X,Y) = Cov(X, 2\mu - X) = \\
&Var(X) - 2\mu Var(X) = \\
&(1-2\mu)Var(X) < 0 \\
&1 - 2\mu < 0 \text{ or } \mu > \frac{1}{2}
\end{aligned}
$$

```r
x <- rnorm(B, 10, 1)
out <- round(c(mean(log(x)), sd(log(x))), 3)
cat("Standard Simulation: ", out,"\n")
```

```
## Standard Simulation:  2.297 0.101
```

```
x <- rnorm(B/2, 10, 1)
y <- 20 - x
z <- (log(x) + log(y))/2
out <- round(c(mean(z), sd(z)), 3)
cat("Antithetic Simulation: ", out,"\n")
```

```
## Antithetic Simulation:  2.298 0.007
```

Let's we have $X \sim N(1/4, 1)$ and want to find $E[X^2]$. Here we have an example where there is no variance reduction because $\mu < 1/2$:

```
x <- rnorm(B, 1/4, 1)
out <- round(c(mean(x^2), sd(x^2)), 3)
cat("Standard Simulation: ", out,"\n")
```

```
## Standard Simulation:  1.053 1.491
```

```
x <- rnorm(B/2, 1/4, 1)
y <- 1/2 - x
z <- (x^2 + y^2)/2
out <- round(c(mean(z), sd(z)), 3)
cat("Antithetic Simulation: ", out,"\n")
```

```
## Antithetic Simulation:  1.051 1.392
```

6.1.2 Control Variables

Again we want to estimate a parameter $\theta = E[X]$. Now suppose that for some other output variable, say Y, we have $E[Y] = \mu$, and μ is known. Then for any constant c

$$E[X + c \cdot (Y - \mu)] = E[X] + cE(Y - \mu) = E[X] = \theta$$

so $X + c \cdot (Y - \mu)$ is an unbiased estimate of θ.

Of course the optimal c is the one that minimizes the variance, and so

$$\begin{aligned} &Var\left(X + c \cdot (Y - \mu)\right) = \\ &Var\left(X + c \cdot Y\right) = \\ &Var(X) + c^2 Var(Y) + 2cCov(X, Y) \end{aligned}$$

so

$$c_{opt} = -\frac{Cov(X, Y)}{Var(Y)}$$

and with this value we have

$$\begin{aligned} &Var(X) + \left(-\frac{Cov(X, Y)}{Var(Y)}\right)^2 Var(Y) + 2\left(-\frac{Cov(X, Y)}{Var(Y)}\right) Cov(X, Y) = \\ &Var(X) - \frac{Cov(X, Y)^2}{Var(Y)} \end{aligned}$$

The quantity Y is called a *control* variable for the simulation variable X.

Let's see how this works. Let's assume X and Y are positively correlated, that is large values of X go with large values of Y.

So if we see a large value of Y (relative to μ) then it is probably true that X is also large (relative to θ) and we should lower our estimate a bit. But this is exactly what happens because now c_{opt} is negative.

####**Example** let's do again the integral example above. We have $X = \exp(U)$. An obvious choice for Y is U itself. We know

$$E[Y] = \frac{1}{2} \text{ , } Var(Y) = \frac{1}{12}$$

and

$$\begin{aligned}
&Cov(U, e^U) = E[Ue^U] - E[U]E[e^U] = \\
&\int_0^1 ue^u du - \frac{1}{2}(e-1) = 0.1409 \\
&Var(X) - \frac{Cov(X,Y)^2}{Var(Y)} = \\
&0.2420 - 0.2380 = 0.0039
\end{aligned}$$

```
u <- runif(B)
x <- exp(u)
c_opt <- -cov(x, u)/var(u)
z <- x + c_opt*(u - 0.5)
out <- round(c(mean(z), var(z)), 4)
cat("Control Variable: ", out,"\n")
```

```
## Control Variable:  1.7184 0.0039
```

####**Example** let's find

$$I = \int_0^2 \exp(-x^2)dx$$

First note that by a change of variable

$$I = 2\int_0^1 \exp(-(2u)^2)du$$

Let $Y = \exp(-2U)$, then

$$\begin{aligned}
&F_Y(y) = P(Y < y) = P(2\exp(-2U) < y) = \\
&P(-2U < \log(y/2)) = \\
&P(U > -\frac{1}{2}\log(y/2)) = \\
&1 - P(U < -\frac{1}{2}\log(y/2)) = \\
&1 + \frac{1}{2}\log(y/2) \text{ for } \frac{2}{e^2} < y < 2 \\
&f_Y(y) = \frac{1}{2y} \\
&E[Y] = \int_{2/e^2}^{2} y\frac{1}{2y}dy = 1 - \frac{1}{e^2}
\end{aligned}$$

```
f <- function(x) exp(-x^2)
out <- round(integrate(f, 0, 2)$value, 2)
cat("Numerical Integration: ", out, "\n")
```

```
## Numerical Integration:  0.88
```

```
u <- runif(B)
z <- 2 * f(2 * u)
out <- round(c(mean(z), sd(z)), 4)
cat("Standard Simulation: ", out, "\n")
```

```
## Standard Simulation:  0.8813 0.6908
```

```
u <- runif(B)
x <- 2 * f(2 * u)
y <- 2 * exp(-2 * u)
mu_y <- 1 - exp(-2)
c_opt <- -cov(x, y)/var(y)
z <- x + c_opt * (y - mu_y)
out <- round(c(mean(z), sd(z)), 4)
cat("Control Variable Simulation: ", out, "\n")
```

```
## Control Variable Simulation:  0.882 0.128
```

####**Example** Consider the following problem: as part of an "online" computer program we need to find the following integral, for different values of $(t_1, .., t_k)$:

$$g_k(\mathbf{t}) = \int_0^1 ... \int_0^1 \sin\left(\prod_{i-1}^{k}[x_i + t_i]\right) dx_1..dx_k$$

It is necessary to find this integral with a precision of ± 0.01, that is a 95% CI for $g_k(\mathbf{t})$ should have a length of no more than 0.002. Each time this integral needs to be found the computer program has to wait and so solving it as fast as possible is important.

Let's first do this using a simple simulation. Generate $U_1, .., U_k \sim U[0,1]$ and calculate $X_1 = \sin(\prod[U_i + t_i])$.

Repeat this n times and then use $\overline{X}$ as an estimate of $g_k(\mathbf{t})$. Also calculate the sample standard deviation s. By the CLT $\overline{X}$ should be approximately normal with mean $g_k(\mathbf{t})$ and standard deviation σ, so a 95% CI for $g_k(\mathbf{t})$ is given by

$$\overline{X} \pm 1.96\sigma/\sqrt{n}$$

so the error is

$$2 \cdot 1.96\sigma/\sqrt{n} = 3.92\sigma/\sqrt{n} = 0.002$$

so $\sigma/\sqrt{n} = 0.0005$.

In other words we need n large enough to ensure $\sigma/\sqrt{n} = 0.0005$. But we don't know σ! We can (approximately) do this as follows:

1) Do a "small" trial run, say 1000 simulations. Based on these you can calculate an estimate of $\sigma = s$
2) If 1000 was already enough ($s < 0.0005$) you are done, otherwise we need $n = (s/0.0005)^2$ and now run the simulation again.

Note: you only need n-1000 runs, you already have 1000.

```r
n <- 1000
t <- c(0.3, 0.3, 0.3)
k <- length(t)
u <- matrix(runif(k * n), n, k)
for (j in 1:k) u[, j] = u[, j] + t[j]
I <- apply(sin(u), 1, prod)
out <- round(c(mean(I), sd(I)), 4)
if( sd(I)/sqrt(n) > 0.0005) {
    n <- round((sd(I)/0.0005)^2, -2)
    cat("n=", n, "\n")
    u <- matrix(runif(k * n), n, k)
    for (j in 1:k) u[, j] = u[, j] + t[j]
    I <- apply(sin(u), 1, prod)
    out <- round(c(mean(I)+c(1-.9, 1.96)*sd(I)/sqrt(n)), 5)
}
```

```
## n= 122600
```

```r
cat("Standard Simulation: ", out, "\n")
```

```
## Standard Simulation:  0.32507 0.32597
```

Now, can we speed this up? let's try the control variable approach.

the obvious choice here is $Y = \prod U_i$, but in order to use it we need to know $\mu = E[Y]$:

$$P(Y < y) = P(\prod_{i=1}^{k} U_i < y) =$$

$$P(\log(\prod_{i=1}^{k} U_i) < \log y) =$$

$$P(\sum_{i=1}^{k} \log U_i < \log y) =$$

$$P(\sum_{i=1}^{k} -\log U_i > -\log y) =$$

$$1 - P(\sum_{i=1}^{k} -\log U_i < -\log y)$$

Now $U \sim U[0,1]$ so $-\log(U) \sim Exp(1)$

and $\sum_{i=1}^{k} (-\log U_i) \sim \Gamma(k,1)$, so

$$P(Y < y) = 1 - \int_0^{-\log y} \frac{1}{\Gamma(k)1^k} t^{k-1} e^{-t/1} dt =$$

$$1 - \int_0^{-\log y} \frac{1}{(k-1)!} t^{k-1} e^{-t} dt, \; y > 0$$

$$f_Y(y) = \frac{d}{dy}\left[1 - \int_0^{-\log y} \frac{1}{(k-1)!} t^{k-1} e^{-t} dt\right] =$$

$$-\frac{1}{(k-1)!}(-\log y)^{k-1} e^{-\log y} \cdot (-\tfrac{1}{y}) =$$

$$\frac{1}{(k-1)!}(-\log y)^{k-1}, \; 0<y<1$$

$$\mu_k = E[Y] = \int_0^1 y \frac{1}{(k-1)!}(-\log y)^{k-1} dy =$$

$$\frac{1}{(k-1)!}\left(\frac{y^2}{2}(-\log y)^{k-1}\Big|_0^1 - \int_0^1 \frac{y^2}{2}(k-1)(-\log y)^{k-2}(-\tfrac{1}{y})dy\right) =$$

$$\frac{1}{2}\int_0^1 y \frac{1}{(k-2)!}(-\log y)^{k-2} dy = \frac{\mu_{k-1}}{2} = .. = \frac{\mu_1}{2^{k-1}} = \frac{1}{2^k}$$

so our new estimator is as $\overline{X} - cov(X,Y)/Var[Y](Y - 2^{-k})$

```
n <- 1000
t <- c(0.3, 0.3, 0.3)
k <- length(t)
u <- matrix(runif(k * n), n, k)
y <- apply(u, 1, prod)
for (j in 1:k) u[, j] = u[, j] + t[j]
x <- apply(sin(u), 1, prod)
c_opt <- -cov(x, y)/var(y)
I <- x + c_opt * (y - (1/2)^k)
out <- round(c(mean(I), sd(I)), 4)
if( sd(I)/sqrt(n) > 0.0005) {
    n <- round((sd(I)/0.0005)^2, -2)
    cat("n=", n, "\n")
    u <- matrix(runif(k * n), n, k)
    y <- apply(u, 1, prod)
    for (j in 1:k) u[, j] = u[, j] + t[j]
    x <- apply(sin(u), 1, prod)
    c_opt <- -cov(x, y)/var(y)
    I <- x + c_opt * (y - (1/2)^k)
    out <- round(c(mean(I)+c(1-.9, 1.96)*sd(I)/sqrt(n)), 5)
}
```

```
## n= 9600
```

```
cat("Control Variable Simulation: ", out, "\n")
```

```
## Control Variable Simulation:  0.32574 0.32669
```

now n=10000 is enough!

####**Example** (The Barbershop)

Many application of variance reduction techniques can be found in the study of queuing systems. As a simple example, consider the case of a barbershop where the barber opens for business every day at 9am and closes at 6pm. He is the only barber in the shop and he is considering hiring another barber to share the workload.

First, however, he would like to estimate the mean total time that customers spend waiting on a given day.

Assume customers arrive at the barbershop according to a non-homogeneous Poisson process, $N(t)$, with intensity $\lambda(t)$, and let W_i denote the waiting time of the i^{th} customer. Then, noting that the barber has a 9-hour work day, the quantity that he wants to estimate is $\mu = E[Y]$ where

$$Y = \sum_{j=1}^{N(9)} W_j$$

Assume also that the service times of customers are IID with CDF, F(:), and that they are also independent of the arrival process, N(t).

The usual simulation method for estimating μ would be to simulate n days of operation in the barbershop, thereby obtaining n samples, $Y_1, ..., Y_n$, and then finding the mean of the Y's.

However, a better estimate could be obtained by using a control variate. In particular, let Z denote the total time customers on a given day spend in service so that

$$Z = \sum_{j=1}^{N(9)} S_j$$

where S_j is the service time of the jth customer. Then, since services times are IID and independent of the arrival process, it is easy to see that

$$E[Z] = E[S]E[N(9)]$$

which should be easily computable. Intuition suggests that Z should be positively correlated with Y and therefore it would also be a good candidate to use as a control variate.

6.2 Conditioning and Importance Sampling

6.2.1 Conditioning

We have previously seen a famous formula for conditional expectations:

$$E\{E[X|Y]\} = E[X]$$

####**Example** Say (X,Y) is a discrete rv with joint density given by

	0	1
0	0.1	0.0
1	0.1	0.2
2	0.0	0.6

So the marginal of X is

x	P(X=x)
0	0.1
1	0.3
2	0.6

and so

$$E[X] = 0 \cdot 0.1 + 1 \cdot 0.3 + 2 \cdot 0.6 = 1.5$$

Also the marginal of Y is

y	P(Y=y)
0	0.2
1	0.8

the conditional density of $X|Y = 0$ is

x	P(X=x|Y=0)
0	0.5
1	0.5
2	0.0

so

$$E[X|Y=0] = 0 \cdot 0.5 + 1 \cdot 0.5 + 2 \cdot 0 = 0.5$$

and the conditional density of $X|Y=1$ is

x	P(X=x|Y=1)
0	0.00
1	0.25
2	0.75

$$E[X|Y=1] = 0 \cdot 0 + 1 \cdot 0.25 + 2 \cdot 0.75 = 1.75.$$

Now let the rv $Z = E[X|Y]$, then

z	P(Z=z)
0.50	0.2
1.75	0.8

and finally

$$E\{E[X|Y]\} = E[Z] = 0.5 \cdot 0.2 + 1.75 \cdot 0.8 = 1.5 = E[X]$$

There is also an equivalent formula for the conditional variance:

$$Var[X] = E[Var(X|Y)] + Var[E(X|Y)]$$

Let's see:

$$Var[X] = E[X^2] - E[X]^2 = 0^2 \cdot 0.1 + 1^2 \cdot 0.3 + 2^2 \cdot 0.6 - 1.5^2 = 0.45$$

Now

$$Var[E(X|Y)] = Var[Z] = 0.5^2 \cdot 0.2 + 1.75^2 \cdot 0.8 - 1.5^2 = 2.5 - 2.25 = 0.25$$

Also Var[X|Y] is a rv (just like E[X|Y]) with density

$$Var[X|Y=0] = E[X^2|Y=0] - E[X|Y=0]^2$$
$$Var[X|Y=1] = E[X^2|Y=1] - E[X|Y=1]^2$$

so if we set $Z_1 = Var[X|Y]$ we have

z	P(Z1=z)
0.250	0.2
1.875	0.8

and

$$E[Var(X|Y)] = E[Z_1] = 0.25 \cdot 0.2 + 0.1875 \cdot 0.8 = 0.2$$

and so

$$Var(E[X|Y]) + E[Var(X|Y)] = 0.2 + 0.25 = 0.45$$

So, how can we use this formula for reducing the variance of our simulation estimators? Because $Var(X|Y) > 0$ always we have

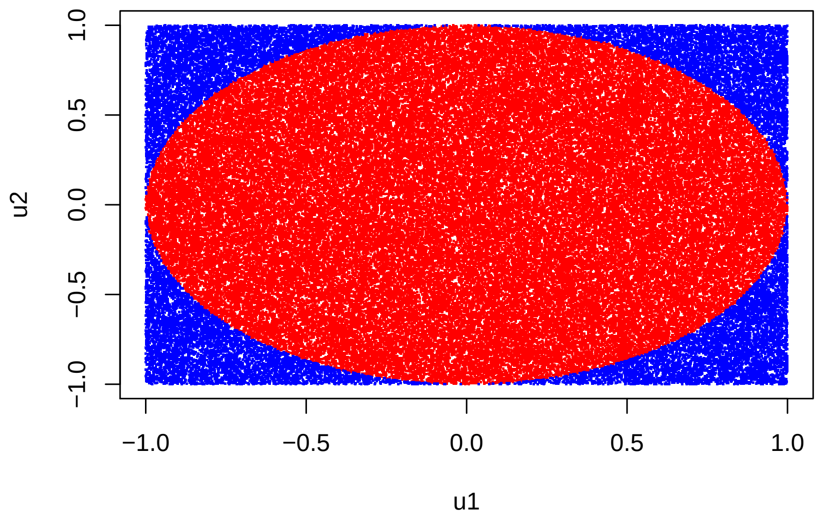
$$Var(X) \geq Var[E(X|Y)]$$

for any rv Y. So say we run a simulation yielding a rv X with $E[X] = \theta$ and the simulation yields a second rv Y, such that $E[X|Y]$ is known. Since $E\{E[X|Y]\} = E[X] = \theta$ it follows that $E[X|Y]$ is also an unbiased estimator of θ and has a variance not larger than X itself.

####**Example** Say we would like to use simulation to estimate the value of π (=3.14…). A straight-forward simulation is as follows:

generate V_1, V_2 iid $U[-1,1]$. If $V_1^2 + V_2^2 \leq 1$ set $Z_i = 1$, otherwise 0. Run the simulation n times, then $4(\sum Z_i)/\text{n}$ is an estimator of π

```
B <- 1e5
u1 <- 2*runif(B)-1
u2 <- 2*runif(B)-1
z <- ifelse(u1^2 + u2^2 < 1, 1, 0)
plot(u1, u2, xlim = c(-1, 1), type = "n",
     ylim = c(-1, 1), pch = ".")
points(u1[z==1], u2[z==1], col = "red", pch = ".")
points(u1[z==0], u2[z==0], col = "blue", pch = ".")
```



```r
z <- 4*z
out <- round(c(mean(z), sd(z)) ,4)
cat("Standard Simulation :", out,  "\n")
```

```
## Standard Simulation : 3.1484 1.6375
```

Now let's use the estimator $E[Z|V_1]$ instead of Z. Note

$$
\begin{aligned}
&E[Z|V_1 = v] = P(V_1^2 + V_2^2 \le 1|V_1 = v) = \\
&P(v^2 + V_2^2 \le 1|V_1 = v) = \\
&P(V_2^2 \le 1 - v^2|V_1 = v) = \\
&P(V_2^2 \le 1 - v^2) = \\
&P(-\sqrt{1-v^2} \le V_2 \le \sqrt{1-v^2}) = \\
&\int_{-\sqrt{1-v^2}}^{\sqrt{1-v^2}} \frac{1}{2} dx = \sqrt{1-v^2}
\end{aligned}
$$

so

$$E[Z|V_1] = \sqrt{1-V_1^2}$$

and so $\sqrt{1-V_1^2}$ is a better estimator than Z alone.

```r
u <- 2 * runif(B) - 1
z <- 4 * sqrt(1 - u^2)
out <- round(c(mean(z), sd(z)) ,4)
cat("Conditional Simulation :", out, "\n")
```

```
## Conditional Simulation : 3.1389 0.8933
```

Note that this new estimator has another advantage: it needs only one $U[0,1]$ per simulation run.

How much better is it? Let's see:

$$
\begin{aligned}
&Z \sim \text{Ber}(\frac{\pi}{4}) \\
&Var(Z) = \frac{\pi}{4}(1-\frac{\pi}{4}) = 0.1686 \\
&Var(\sqrt{1-V_1^2}) = E[(\sqrt{1-V_1^2})^2] - E[\sqrt{1-V_1^2}]^2 = \\
&E[1-V_1^2] - (\frac{\pi}{4})^2 = \\
&1 - E[V_1^2] - (\frac{\pi}{4})^2 = \\
&1 - (\frac{\pi}{4})^2 - \left(Var(V_1) + (E[V_1])^2\right) = \\
&1 - (\frac{\pi}{4})^2 - \left(\frac{1^2-(-1)^2}{12} + (0)^2\right) = 0.0498
\end{aligned}
$$

and so

$$
Var(Z) = Var(4(\sqrt{1-V_1^2})] = 16 * 0.0498 = 0.7968
$$

####**Example** Say $X \sim \text{Exp}(1)$, $Z \sim \text{Exp}(1/2)$, independent and we want to find $p = P(X+Z \ge 4)$

$$
\begin{aligned}
&P(X+Z \ge 4) = E[I_{[4,\infty)}(X+Z)] = \\
&E\left\{E[I_{[4,\infty)}(X+Z)]|Z]\right\} = \\
&E[I_{[4,\infty)}(X+Z)]|Z=z] = \\
&E[I_{[4,\infty)}(X+z)]|Z=z] = \\
&P(X > 4-z) = 1 - P(X < 4-z) = \\
&\exp(-(4-z)) = \exp(z-4)
\end{aligned}
$$

if $z < 4$ and 0 otherwise. So

```
B <- 1e5
x <- rexp(B, 1)
z <- rexp(B, 2)
v <- ifelse(x+z>4, 1, 0)
cat("Standard Simulation :", mean(v), "  sd :", sd(v), "\n")
```

```
## Standard Simulation : 0.03622   sd : 0.1868381
```

```
v <- ifelse(z<4, exp(z-4), 1)
cat("Conditioning Simulation :", mean(v), "  sd :", sd(v), "\n")
```

```
## Conditioning Simulation : 0.03613685   sd : 0.0406666
```

####**Example** say we want to find

$$I = \int_0^\infty \int_0^1 \sqrt{x+y}\; e^{-x} dx$$

Now $I = E[\sqrt{X+U}]$ where $X \sim \text{Exp}(1)$ and $U \sim U[0,1]$.
Let $V = E[\sqrt{X+U}|X]$, then

$$\begin{aligned}
&E[\sqrt{X+U}|X=x] = \\
&E[\sqrt{x+U}] = \int_0^1 \sqrt{x+u}du = \\
&\frac{2}{3}\sqrt{(x+u)^3}|_0^1 = \\
&\frac{2}{3}\left(\sqrt{(x+1)^3} - \sqrt{x^3}\right)
\end{aligned}$$

```r
B <- 1e5
x <- rexp(B, 1)
u <- runif(B)
v <- sqrt(x+u)
cat("Standard Simulation :", mean(v), "  sd :", sd(v), "\n")
```

```
## Standard Simulation : 1.158874   sd : 0.3931618
```

```r
x = rexp(B, 1)
v <- 2/3*(sqrt((x+1)^3) - sqrt((x)^3))
cat("Conditioning Simulation :", mean(v), "  sd :", sd(v), "\n")
```

```
## Conditioning Simulation : 1.161367   sd : 0.36729
```

6.2.2 Importance Sampling

####**Example** say we have a rv X geometric with $p = 0.5$. We want to find $P(\log(X!) > 50)$.

Let's try to solve this problem analytically. First, $\log(x!)$ is an increasing function of x, so there exists x_{50} such that $\log(x!) > 50$ iff $x > x_{50}$, so that

$$P(\log(X!) > 50) = P(X \geq x_{50})$$

Finding x_{50} analytically is hopeless, though. We can do it with R by trial and error: using *log(factorial(n))** for different values of n:

```r
log(factorial(10))
```

```
## [1] 15.10441
```

```r
log(factorial(20))
```

```
## [1] 42.33562
```

```r
log(factorial(30))
```

```
## [1] 74.65824
```

```r
log(factorial(25))
```

```
## [1] 58.00361
```

```r
log(factorial(22.5))
```

```
## [1] 50.03349
```

We find n=22.5, so

$$P(X \geq 23) = 1 - P(X \leq 22) = 1 - \sum_{i=1}^{22} \left(\frac{1}{2}\right)^i = 1 - \left(\sum_{i=0}^{22} \left(\frac{1}{2}\right)^i - 1\right) = 2 - \frac{1 - 0.5^{23}}{1 - 0.5}$$

or about 2.38×10^{-7}

How about an R check? The problem with this is that the probability p we want to find is very small, so in a simple simulation as shown in we can expect the outcome of interest only about every 1 in 4.2million runs. In order to get some reasonably good estimate we probably need to run the simulation with $n = 10^9$.

Here is an idea: the problem is that our event of interest, $\log(X!) > 50$, is very rare, it almost never happens. Let's instead sample from a distribution Y which has large values much more often, so that $\log(Y!) > 50$ happens more often. For example, let's try Y geometric with $p = 0.05$:

```r
B <- 1e5
y <- rgeom(B, 0.05)+1
logfac_y <- 0.918938533205 + (y+0.5)*log(y)-y
sum(logfac_y>50)/B
```

```
## [1] 0.32314
```

Note the calculation of $\log(y!)$ this is based on Stirlings' Formula:

$$n! \approx \sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n}$$

so

$$\log(n!) \approx \log(\sqrt{2\pi}) + (n + \frac{1}{2}) \log(n) - n$$

this is to avoid problem with numbers that are bigger than R can handle!

So $P(\log(Y!) > 50) = 0.35$. But what good is that? I want X! Well:

$$P(\log(X!) > 50) = E[I_{[\log(X!)>50]}(X)] =$$

$$\sum_{x=1}^{\infty} I_{[\log(x!)>50]}(x) f_X(x) =$$

$$\sum_{x=1}^{\infty} I_{[\log(x!)>50]}(x) f_X(x) \cdot \frac{f_Y(x)}{f_Y(x)} =$$

$$\sum_{x=1}^{\infty} \left[I_{[\log(x!)>50]}(x) \frac{f_X(x)}{f_Y(x)} \right] f_Y(x) =$$

$$E\left[I_{[\log(x!)>50]}(Y) \frac{f_X(Y)}{f_Y(Y)} \right] \simeq$$

$$\frac{1}{n} \sum_{i=1}^{n} \left[I_{[\log(x!)>50]}(Y_i) \frac{f_X(Y_i)}{f_Y(Y_i)} \right]$$

so if we sample from Y and find the sum we can still get an estimate of the probability for X:

```r
y <- y[logfac_y >= 50]
w <- dgeom(y - 1, 0.5)/dgeom(y - 1, 0.05)
return(sum(w)/B)
```

```
## [1] 2.390011e-07
```

In general we have the following: Let X be a rv' with density f and and Y a rv' with density g. Say we want to find $E[h(X)]$. Then

$$E[h(X)] = \sum_{i=1}^{\infty} h(x_i)f(x_i) =$$

$$\sum_{i=1}^{\infty} h(x_i)f(x_i)\frac{g(x_i)}{g(x_i)} =$$

$$\sum_{i=1}^{\infty}\left[h(x_i)\frac{f(x_i)}{g(x_i)}\right]g(x_i) =$$

$$E\left[h(Y)\frac{f(Y)}{g(Y)}\right] = E[h(Y)w(Y)]$$

where

$$w(y) = \frac{f(y)}{g(y)}$$

are called the "weights"

Note this was done for discrete rv's but it works just as well for continuous ones.

Note how to choose Y? Obviously we need Y such that it can't happen that $P(Y = x) > 0$ and $P(X = x) = 0$. In general we should choose a Y with the same *support* as X, that is $P(X = x) > 0$ iff $P(Y = x) > 0$.

It is not necessary to have a Y that "looks like" X. For example in the case above we could have chosen Y with density

$$f_Y(x) = 6/(\pi^2 x^2)\ ,\ x = 1, 2, ..$$

It is also a good idea to choose Y such that the event of interest , here $\log(Y!) > 50$, happens about 50% of the time.

####**Example** say X, Y and Z have a standard normal distribution. Find $P(|XYZ| > K)$, for example $K = 10$

Now there is no way to do this analytically, and again the probability is very small. So we will use IS with X', Y' and Z' generated from normal distributions with mean 0 and standard deviation s. For our case of $K = 10$ $s = 3.5$ works good. In general, for some K play around a bit to find a good s.

```
B <- 1e5
K <- 10
s <- 3.5
x <- rnorm(B, 0, s)
y <- rnorm(B, 0, s)
z <- rnorm(B, 0, s)
T <- abs(x*y*z)
I <- c(1:B)[T > K]
print(length(I)/B)
```

```
## [1] 0.45999
```

```
w <- dnorm(x[I])/dnorm(x[I], 0, s)*dnorm(y[I])/dnorm(y[I],  0, s)*dnorm(z[I])/dnorm(z[I
sum(w)/B
```

```
## [1] 0.0003760303
```

####**Example** (From a book by Robert and Casella) let $X \sim$ Cauchy and we want to use simulation to estimate $\tau = P(X > 2)$

$$\tau = P(X > 2) =$$

$$\int_2^\infty \frac{1}{\pi(1+x^2)} dx = \frac{1}{\pi} \tan^{-1}(x)|_2^\infty =$$

$$\frac{1}{\pi}\left(\frac{\pi}{2} - \tan^{-1}(2)\right) = \frac{1}{2} - \frac{\tan^{-1}2}{\pi} = 0.1476$$

Method 1: (Direct Simulation) generate $X_1, .., X_n$ iid Cauchy, estimate $\tau = 1/n \sum I_{[2,\infty)}(X_i)$

$$Var\left(\frac{1}{n}\sum_{i=1}^n I_{[2,\infty)}(X_i)\right) = \frac{1}{n} Var[I_{[2,\infty)}(X)]$$

$I_{[2,\infty)}(X) \sim Ber(\tau)$, so $Var[I_{[2,\infty)}(X)] = \tau(1-\tau)$

and

$$Var\left(\frac{1}{n}\sum_{i=1}^n I_{[2,\infty)}(X_i)\right) = \frac{0.1476(1-0.1476)}{n} = \frac{0.1258}{n}$$

```
x <- rcauchy(B)
z <- ifelse(x > 2, 1, 0)
```

```r
c(mean(z), sd(z))
```

```
## [1] 0.1455600 0.3526663
```

Method 2: (Direct Simulation, using a special feature of the problem) Make use of the fact that a Cauchy is symmetric around 0, so

$$P(X > 2) = \frac{1}{2}P(|X| > 2)$$

so generate $X_1, .., X_n$ iid Cauchy, estimate

$$\tau = \frac{1}{2n}\sum I_{[2,\infty)}(|X|_i)$$

$$Var\left(\frac{1}{2n}\sum_{i=1}^{n} I_{[2,\infty)}(|X_i|)\right) = \frac{1}{4n} Var[I_{[2,\infty)}(|X|)]$$

$I_{[2,\infty)}(|X|) \sim Ber(2\tau)$, so $Var[I_{[2,\infty)}(|X|)] = 2\tau(1-2\tau)$

and

$$Var\left(\frac{1}{n}\sum_{i=1}^{n} I_{[2,\infty)}(|X_i|)\right) = \frac{2\cdot 0.1476(1-2\cdot 0.1476)}{4n} = \frac{0.052}{n}$$

```r
z <- ifelse(abs(x) > 2, 1, 0)/2
c(mean(z), sd(z))
```

```
## [1] 0.1462700 0.2274656
```

Method 3: (Direct Simulation, using a special feature of the problem) Make use of the fact that

$\tau = \int_2^\infty \frac{1}{\pi(1+x^2)}dx = \frac{1}{2} - \int_0^2 \frac{1}{\pi(1+x^2)}dx$

so say $U_1,..,U_n \sim U[0,2]$

set $\hat{\tau} = \frac{1}{2} - \frac{2}{n}\sum_{i=1}^n \frac{1}{\pi(1+U_i^2)}$ then $E[\hat{\tau}] = \tau$

$Var[\hat{\tau}] = Var\left[\frac{1}{2} - \frac{2}{n}\sum_{i=1}^n \frac{1}{\pi(1+U_i^2)}\right] = \frac{4}{\pi^2 n}Var\left[\frac{1}{1+U_1^2}\right]$

$E[\frac{1}{1+U_1^2}] = \int_0^2 \frac{1}{1+t^2}\frac{1}{2}dt = \frac{1}{2}\arctan(t)|_0^2 = \frac{1}{2}\arctan(2) = 0.5535744$

$E[\left(\frac{1}{1+U_1^2}\right)^2] = \int_0^2 \frac{1}{(1+t^2)^2}\frac{1}{2}dt =$

$\frac{1}{2}[\frac{t}{2(1+t^2)} + \frac{1}{2}\arctan(t)|_0^{1/2}] = 0.3767872$

$Var[\hat{\tau}] = \frac{4}{\pi^2 n}(0.3767872 - 0.5535744^2) = 0.0285/n$

```r
x <- runif(B, 0, 2)
z <- 1/2-2/pi/(1 + x^2)
c(mean(z), sd(z))
```

```
## [1] -5.392704  4.324530
```

Method 4: (Direct Simulation, using a special feature of the problem)

$$\tau = \int_2^\infty \frac{1}{\pi(1+x^2)}dx$$

Let's do the change of variables $y = 1/x$

$$dx = -\frac{1}{x^2}dy$$

$$\tau = \int_{1/2}^{0} \frac{1}{\pi(1+(1/y)^2}\left(-\frac{1}{y^2}\right)dy = \int_0^{1/2} \frac{1/y^2}{\pi(1+1/y^2)}dy =$$

$$\int_0^{1/2} \frac{1}{\pi(1+y^2)}dy$$

so let $U_1,..,U_n \sim U[0,1/2]$

$$\hat{\tau} = \frac{1}{2\pi n}\sum_{i=1}^n \frac{1}{1+U_i^2}$$

Now

$$Var[\hat{\tau}] = \frac{1}{4\pi^2 n}Var[\frac{1}{1+U_1^2}]$$

$$E[\frac{1}{1+U_1^2}] = \int_0^{1/2}\frac{1}{1+t^2}2dt = 2\arctan(t)|_0^{1/2} = 2\arctan(1/2) = 0.9272952$$

$$E[\left(\frac{1}{1+U_1^2}\right)^2] = \int_0^{1/2}\frac{1}{(1+t^2)^2}2dt =$$

$$2[\frac{t}{2(1+t^2)} + \frac{1}{2}\arctan(t)|_0^{1/2}] = 0.8636476$$

$$Var[\hat{\tau}] = \frac{1}{4\pi^2 n}(0.8636476 - 0.9272952^2) = 9.5526 \cdot 10^{-5}/n$$

```
x <- runif(B, 0, 0.5)
z <- 1/2/pi/(1 + x^2)
c(mean(z), sd(z))
```

```
## [1] 2.471424 1.248377
```

Method 5: (Importance sampling) Let's use the rv Y with density $g(x) = 2/x$, $x > 2$. Note

$$G(x) = \int_{-\infty}^{x} g(x)dx = \int_{2}^{x} 2/t^2 dt = -2/t|_2^x = 1 - \frac{2}{x},\ x > 2$$

$y = G(x) = 1 - \frac{2}{x}$, so $x = \frac{2}{1-y} = G^{-1}(y)$,

so we can generate rv's from G with $Y = \frac{2}{U}$ where $U \sim U[0,1]$

$$w(y) = \frac{f(y)}{g(y)} = \frac{\frac{1}{\pi(1+y^2)}}{2/y^2} = \frac{y^2}{2\pi(1+y^2)}$$

$$\hat{\tau} = \frac{1}{n}\sum_{i=1}^{n} \frac{(2/U)^2}{2\pi(1+(2/U)^2)} = \frac{1}{n\pi}\sum_{i=1}^{n} \frac{2}{(4+U^2)}$$

Note if $U \sim U[0,1]$, then $U' = U/2 \sim U[0,1/2]$ and

$$\frac{1}{n\pi}\sum_{i=1}^{n} \frac{2}{4+U^2} = \frac{1}{n\pi}\sum_{i=1}^{n} \frac{2}{(4+(2U')^2)} = \frac{1}{2n\pi}\sum_{i=1}^{n} \frac{1}{1+U'^2}$$

so this is actually the same as Method 4, with the same variance.

```r
x <- runif(B)
z <- 2/(4 + x^2)/pi
c(mean(z), sd(z))
```

```
## [1] 2.472509 1.249050
```

####**Example** Say we have the following problem: we have $X_1, .., X_n$ iid Pois(λ) and we want to test

$H_0 : \lambda = \lambda_0$ vs. $H_a : \lambda \neq \lambda_0$

we decide to use a Wald-type test, that is a test based on the CLT. Of course by the CLT

$$T_n = \frac{\sum X_i - n\lambda}{\sqrt{n\lambda}} \sim N(0,1)$$

and so we have a test of the form

reject H_0 if $|T_n| > z_{\alpha/2}$

Now this is based on the CLT, and so we need to worry whether is works for our n and λ_0, say $n = 100$ and $\lambda_0 = 2.0$. Easy enough, we do a simulation:

- generate rpois(100, 2.0)
- calculate T_n
- check whether $|T_n| > z_{\alpha/2}$
- repeat B times

```r
alpha <- 0.05
lambda <- 2
```

```r
n <- 100
B <- 10^5
x <- rpois(B, n * lambda)
T_n = (x - n * lambda)/sqrt(n * lambda)
sum(abs(T_n) > qnorm(1 - alpha/2))/B
```

```
## [1] 0.05286
```

and so the test works as it should.

Note that we can use the fact that $\sum X_i \sim \text{Pois}(n\lambda)$.

Now in most fields hypothesis tests are done with $\alpha = 0.05$ or so. In High Energy Physics, though, they use $\alpha = 2.87 \times 10^{-7}$! (this strange number happens to be pnorm(-5), so they are looking for a "5-sigma effect") . The reason for such a small α is that in HEP we have a very serious simultaneous inference problem.

So now we should check whether this test still works if we use $\alpha = 2.87 \times 10^{-7}$. But even if it does $|T_n| > 5$ will only happen every 3.5 million runs or so $(1/\alpha)$, so to get some reasonable estimate we would need $B = 10^9$ or so.

Let's use IS instead. Again we need to generate data from an rv where $|T_n| > 5$ happens more often. Say we use $Y \sim \text{Pois}(n\tau)$. Now

$$
\begin{aligned}
&w(y) = \text{dpois}(y, n\lambda)/\text{dpois}(y, n\tau) \\
&T_n = \frac{y - n\lambda}{n\lambda} \\
&I_n(y) = 1 if |T_n| > 5, \text{ 0 otherwise} \\
&P(|T_n| > 5) = \text{Mean}(I_n w)
\end{aligned}
$$

For example, if $n = 100$ and $\lambda = 2.0$, use $\tau = 2.7$.

```r
tau <- 2.7
alpha <- pnorm(-5)
y <- rpois(B, n * tau)
T_n <- ifelse(abs((y - n * lambda)/sqrt(n * lambda)) > qnorm(1 - alpha/2), 1, 0)
 w <- dpois(y, n * lambda)/dpois(y, n * tau)
alphahat = mean(w * T_n)
c(truealpha = alphahat, sigmas = qnorm(1 - 2 * alphahat), percentage = sum(T_n)/B)
```

```
##    truealpha       sigmas   percentage
## 5.679983e-07 4.727591e+00 4.363600e-01
```

finds that the actual type I error probability is a about twice what it is supposed to be.

7 Additional Topics

7.1 Optimization

In this section we will study methods for finding maxima and minima of a function f. Of course the first try will always be via calculus, that is finding f' and solving $f'(x) = 0$. This, though, only works if the function is fairly simple and the zeros of f' can be found analytically.

7.1.1 Numerical Optimization

If that is not the case we can try and use numerical methods. The most famous of them is the *Newton-Raphson* algorithm. It chooses a starting point x_0 and then iteratively calculates

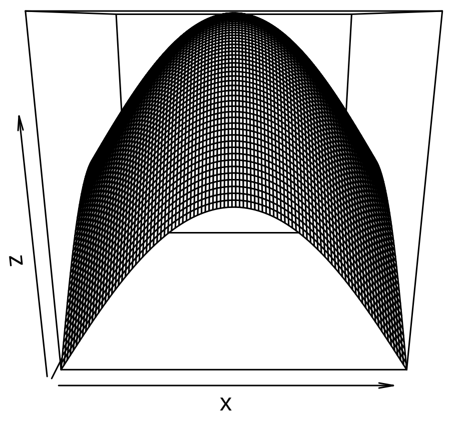
$$x_n = x_{n-1} - H^{-1}\nabla$$

where H is the Hessian matrix and ∇ is the gradient of f evaluated at x_{n-1}.

####**Example**

say $f(x, y) = \sin(x) + \sin(y)$, $0 < x, y < \pi$

```
f <- function(x, y) sin(x)+sin(y)
x <- seq(0, pi, length=100)
y <- x
z <- matrix(0, 100, 100)
for(i in 1:100) z[ ,i] <- f(x[i], y)
persp(x, y, z)
```



```
ij <- which(z == max(z), arr.ind = TRUE)[1, ]
round(c(x[ij[1]], y[ij[2]], z[ij[1], ij[2]]), 2)
```

```
## [1] 1.55 1.55 2.00
```

$$f(x,y) = \sin(x) + \sin(y)$$

$$\frac{df}{dx} = \cos(x)$$

$$\frac{df}{dy} = \cos(y)$$

$$H_{xx} = -\sin(x)$$

$$H_{xy} = H_{yx} = 0$$

$$H_{yy} = -\sin(y)$$

$$\begin{aligned}
&f(x,y) = \sin(x) + \sin(y) \\
&\nabla_x = \frac{df}{dx}(x,y) = \cos(x) \\
&\nabla_y = \frac{df}{dy}(x,y) = \cos(y) \\
&H_{1,1} = \frac{d^2 f}{dy^2}(x,y) = -\sin(y) \\
&H_{1,2} = H_{2,1} = \frac{d^2 f}{dydx}(x,y) = 0 \\
&H_{2,2} = \frac{d^2 f}{dx^2}(x,y) = -\sin(x)
\end{aligned}$$

```r
newp <- c(1, 1)
repeat {
  oldp <- newp
  grad <- cbind(cos(oldp))
  H <- matrix(c(-sin(oldp[1]), 0, 0, -sin(oldp[2])), 2, 2)
  newp <- oldp - solve(H)%*%grad
  print(round(c(newp, f(newp[1], newp[2])), 4))
  if(sum(abs(oldp-newp))<0.0001) break
}
```

```
## [1] 1.6421 1.6421 1.9949
## [1] 1.5707 1.5707 2.0000
## [1] 1.5708 1.5708 2.0000
## [1] 1.5708 1.5708 2.0000
```

7.1.2 Direct Simulation

Here we randomly pick points in some area, evaluate the function and pick the points which have the maxima

####**Example** $f(x, y) = \sin(x) + \sin(y)$ $0 < x, y < \pi$

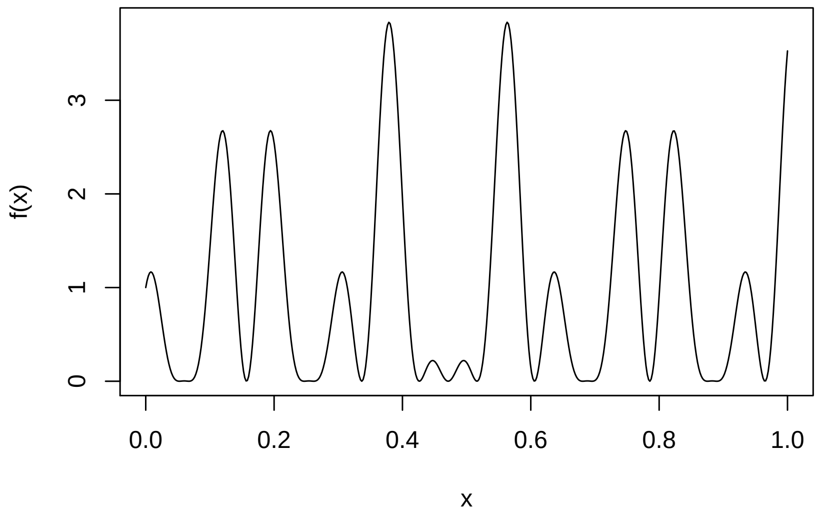
```r
x <- runif(100, 0, pi)
y <- runif(100, 0, pi)
z <- matrix(0, 100, 100)
for(i in 1:100) z[ ,i] <- f(x[i], y)
ij <- which(z == max(z), arr.ind = TRUE)[1, ]
round(c(x[ij[1]], y[ij[2]], z[ij[1], ij[2]]), 4)
```

```
## [1] 1.2508 1.6075 2.0000
```

Example consider the function

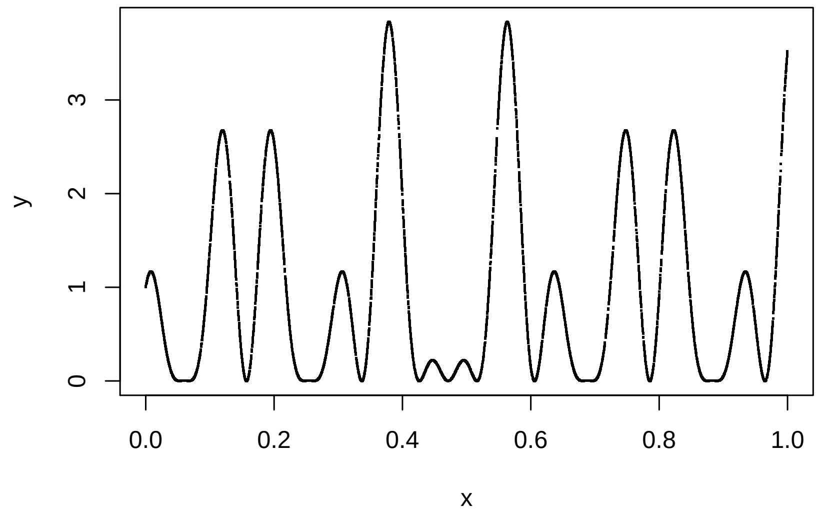
$$f(x) = [\cos(50x) + \sin(20x)]^2, \; x \in [0, 1]$$

```r
f <- function(x) (cos(50*x) + sin(20*x))^2
curve(f, 0, 1, n=500)
```



we see it has many maxima and minima. Here using Newton-Raphson is almost certainly going to fail because the starting point would have to be almost at the maximum.

```
B <- 1e4
x <- runif(B)
y = f(x)
plot(x, y, pch = ".")
```



```
round(c(x[y == max(y)], max(y)), 2)
```

```
## [1] 0.56 3.83
```

This does work, but we do need a lot of U's because the peak at the maximum is very sharp.

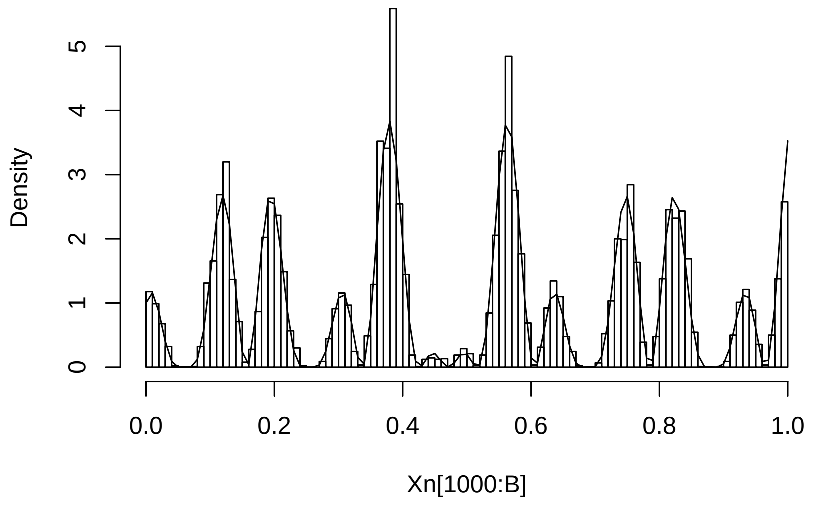
Here is another idea:

Because f is a continuous function on a finite interval there exists a constant c such that $c \cdot f$ is a density.

Moreover using the Hastings-Metropolis algorithm we don't even need to know c.

One problem is to extract the maximum from the generated data. We can use a histogram to estimate the density and pick the maximum. In general a non-parametric density estimator would be better and would need far fewer points.

```r
B <- 1e4
Xn <- rep(0, B)
Xn[1] <- 0.5
for (i in 2:B) {
  X <- runif(1)
  if (runif(1) < f(X)/f(Xn[i-1]))
    Xn[i] = X
  else Xn[i] = Xn[i-1]
}
hist(Xn[1000:B], breaks = 100, freq = FALSE, main="")
curve(f, 0, 1, add=TRUE)
```



```r
a <- hist(Xn[100:B], plot = FALSE)
a$breaks[which.max(a$counts)]
```

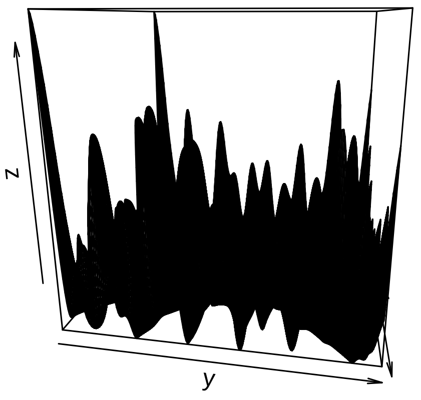
```
## [1] 0.35
```

####**Example**

Consider the function

$$\begin{aligned}f(x,y) =& \\ &(x\sin(20y)+y\sin(20x))^2\cosh(\sin(10x)x)+\\ &(x\cos(10y)-y\sin(10x)^2\cosh(\cos(20y)y)\\ &-1<x,y<1\end{aligned}$$

```
f <- function(x, y)
    (x*sin(20*y) + y*sin(20*x))^2 * acos(sin(10*x)*x) +
    (x*cos(10*y) - y*sin(10*x))^2 * acos(cos(20*y)*y)
n <- 250
x <- seq(-1, 1, length = n)
y <- x
z <- matrix(0, n, n)
for (i in 1:n) z[i, ] = f(x[i], y)
persp(x, y, z, theta = 100)
```



using the simple simulation approach is easy:

```
B <- 1e4
x <- runif(B, -1, 1)
y <- runif(B, -1, 1)
z <- f(x, y)
I <- c(1:B)[which.max(y)]
round(c(x[I], y[I], z[I]), 2)
```

```
## [1] 0.14 1.00 1.67
```

The solution via the histogram/non-parametric density estimate again is doable but needs a bit of work.

7.1.3 Simulated Annealing

this algorithm was actually also introduced by Metropolis, in the same 1953 paper where he first showed the "Metropolis" version of the Metropolis-Hastings algorithm. The fundamental idea is that a change of scale, called *temperature*, allows for faster moves on the surface of the function f to maximize, whose negative is called the energy.

Therefore rescaling partially avoids the trapping of the algorithm in local minima/maxima. Given a temperature parameter T0 a sample of

$\theta_1(T)$, $\theta_2(T)$,..

is generated from the distribution

$$\pi(\theta) = c \cdot \exp(f(\theta)/T)$$

Notice that

$$\begin{aligned}&\pi'(\theta) = c \cdot \exp(f(\theta)/T)f'(\theta)/T = 0\\ &\text{iff}\\ &f'(\theta) = 0\end{aligned}$$

so $\pi(\theta)$ has a maximum iff $f(\theta)$ has a maximum.

Moreover even if $\int f(x)dx = \infty$, $\int \exp(f(x))dx$ is often finite and so there exists a constant c which makes π a density.

Here is one popular version of the simulated annealing algorithm:

1) simulate Y from a distribution with the same support as f, say with density $g(|y - \theta_i|)$
2) accept $\theta_{i+1} = Y$ with probability

   $$p = \min\{\exp[(f(Y) - f(\theta_i))/T_i), 1\}$$

   take $\theta_{i+1} = \theta_i$ otherwise
3) update T_i to T_{i+1}

Notice the similarities between this algorithm and the Hastings-Metropolis one: in each case we draw observations from a "proposal distribution" which depends on the current state x, and accept it as a new observation for X with a certain probability.

Example

$f(x) = [\cos(50x) + \sin(20x)]^2$ on [0,1]. For this one implementation of the algorithm is as follows:

at iteration i the algorithm is at $(x^i, f(x^i))$

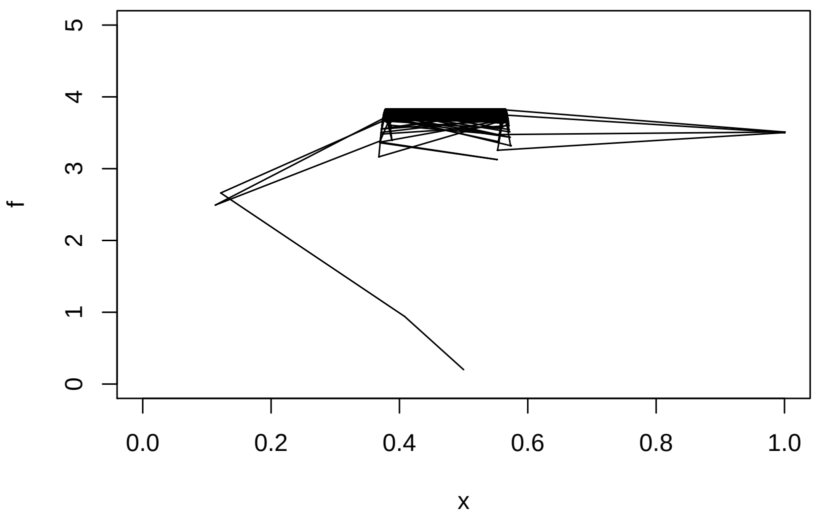
1) simulate $U \sim U[a^i, b^i]$ where $a^i = \max(x^i - 0.5, 0)$ and $b^i = \min(x^i + 0.05, 1)$
2) accept $x^{i+1} = U$ with probability

$p^i = \min\{\exp[(f(U) - f(x^i))/T_i], 1\}$

otherwise set $x^{i+1} = x^i$

3) set $T^{i+1} = 1/log(i+1)$

```r
f = function(x) (cos(50*x) + sin(20*x))^2
B <- 1e4
x <- rep(0.5, B)
y <- rep(f(x[1]), B)
M <- c(x[1], y[1])
plot(c(0, 1), c(0, 5),
     type = "n", xlab = "x", ylab = "f")
for (i in 2:B) {
  U <- runif(1, max(x[i-1]-0.5, 0), min(x[i-1]+0.5, 1))
  y[i] <- f(U)
  p <- min(exp((y[i] - y[i-1])* log(i+1)), 1)
  if(runif(1) < p)
     x[i] <- U
  else {
    x[i] <- x[i-1]
    y[i] <- y[i-1]
  }
  if(y[i]>M[2])
      M <- c(x[i], y[i])
      segments(x[i-1], y[i-1], x[i], y[i])
  }
```



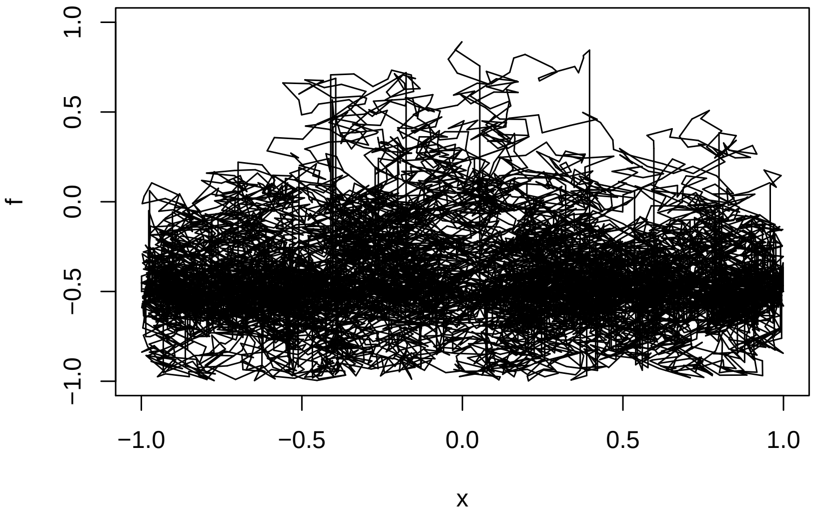
```r
M
```

```
## [1] 0.379145 3.832544
```

####**Example**

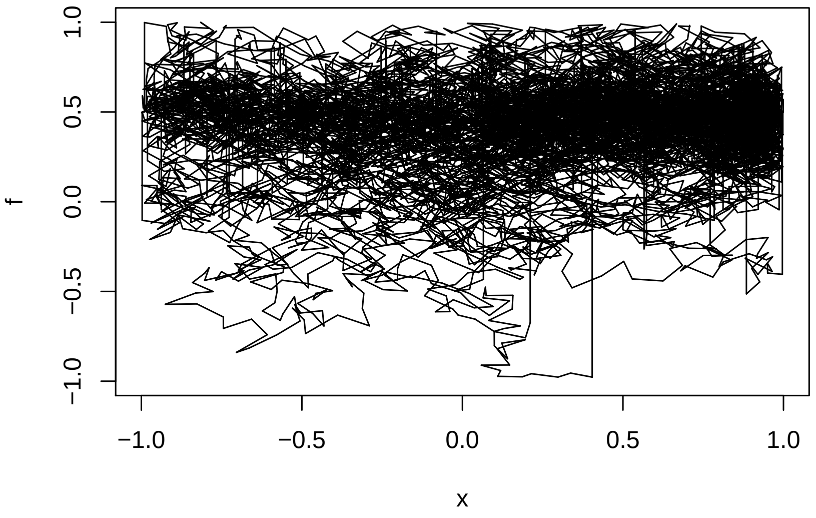
$$
\begin{aligned}
f(x,y) = & \\
& (x\sin(20y) + y\sin(20x))^2\cosh(\sin(10x)x) + \\
& (x\cos(10y) - y\sin(10x)^2\cosh(\cos(20y)y) \\
& -1 < x, y < 1
\end{aligned}
$$

```r
f <- function(x, y)
    (x*sin(20*y) + y*sin(20*x))^2 * acos(sin(10*x)*x) +
    (x*cos(10*y) - y*sin(10*x))^2 * acos(cos(20*y)*y)
f_opt <- function(start, eps = 0.025) {
  x <- rep(start[1], B)
  y <- rep(start[2], B)
  fun <- rep(f(x[1], y[1]), B)
  M <- c(x[1], y[1], fun[1])
  plot(c(-1, 1), c(-1, 1), type = "n", xlab = "x", ylab = "f")
  for (i in 2:B) {
    U1 <- runif(1, max(x[i-1]-0.1, -1), min(x[i-1]+0.1, 1))
    U2 <- runif(1, max(y[i-1]-0.1, -1), min(y[i-1]+0.1, 1))
    fun[i] <- f(U1, U2)
    p <- min(exp((fun[i]-fun[i-1])*eps*log(i+1)), 1)
    if(runif(1)<p) {x[i] <- U1; y[i] <- U2}
    else {
        x[i] <- x[i-1]
        y[i] <- y[i]
        fun[i] <- fun[i-1]
      }
      if(fun[i]>M[3])
      M <- c(x[i], y[i], fun[i])
      segments(x[i-1], y[i-1], x[i], y[i])
    }
  M
}
f_opt(c(-0.5, -0.5))
```



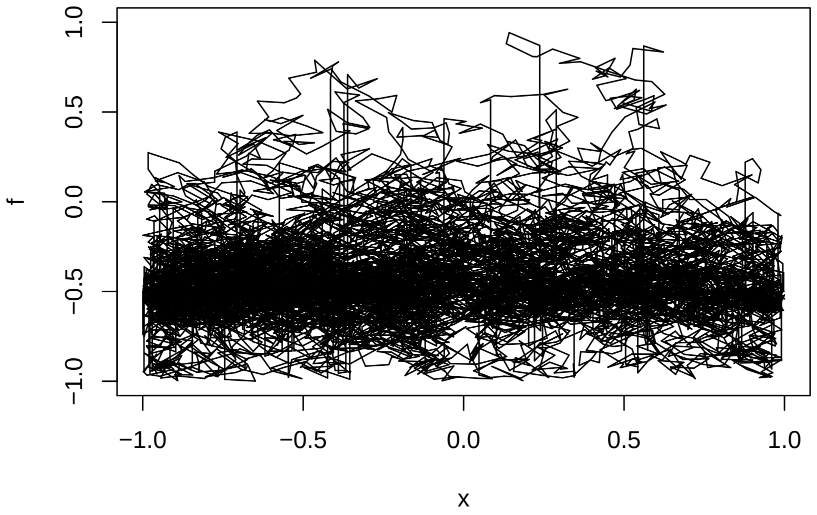
```
## [1] -0.5268550 -0.9345497  6.5037106
```

```
f_opt(c(-0.5, 0.5))
```



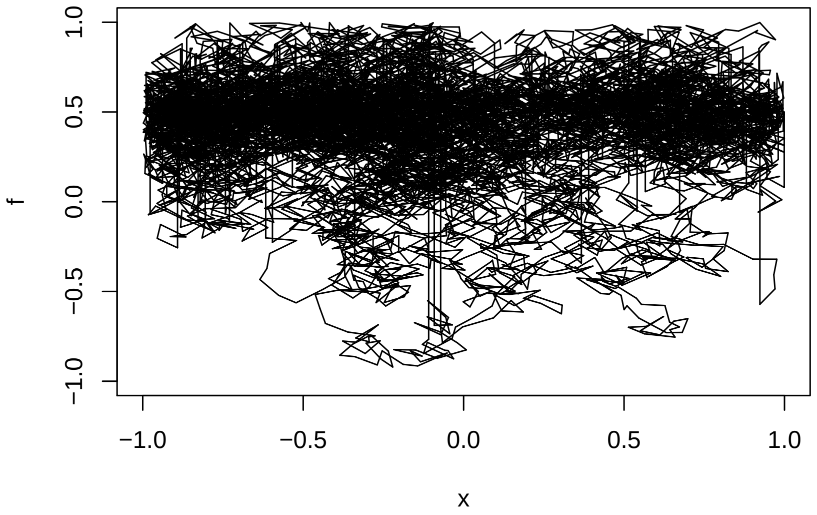
```
## [1] -0.9901463  0.9980210  6.0729629
```

```
f_opt(c(0.5, -0.5))
```



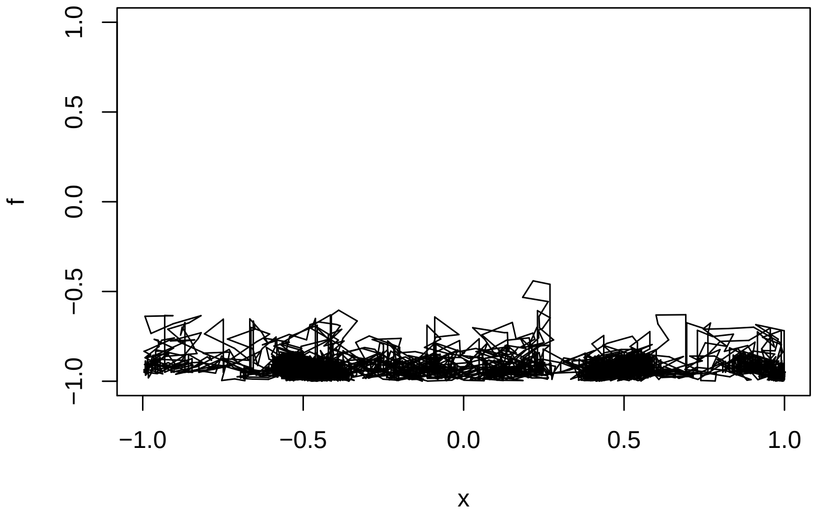
```
## [1] -0.9985411 -0.9462792  8.1523719
```

```r
f_opt(c(0.5, 0.5))
```



```
## [1] -0.9927066  0.7062291  6.9319666
```

```r
f_opt(c(-0.99, -0.95), eps=0.1)
```



```
## [1]  0.9996405 -0.9906784 10.7457202
```

Different values of eps change the temperature T. Also, rerunning the routine from different starting points is often a good idea!

7.2 EM (Expectation-Maximization)

The EM algorithm seems at first to solve a very specific problem but it turns out to be quite useful in general.

####**Example** Let's return to the normal mixture model we considered earlier:

$$\begin{aligned}
&Y_1 \sim N(\mu_1, \sigma_1)\\
&Y_2 \sim N(\mu_2, \sigma_2)\\
&Z \sim \text{Ber}(p)\\
&X = (1-Z)Y_1 + ZY_2
\end{aligned}$$

Let's assume for the moment that in addition to X we also observe Z. Then

$$P(X \le x, Z = z) =$$

$$P(X \le x | Z = z) P(Z = z) =$$

$$\begin{cases} (1-p)\Phi(x,\mu_1,\sigma_1) & \text{if } z = 0 \\ p\Phi(x,\mu_2,\sigma_2) & \text{if } z = 1 \end{cases} =$$

$$[(1-p)\Phi(x,\mu_1,\sigma_1)]^{1-z} \cdot [p\Phi(x,\mu_2,\sigma_2)]^{z}$$

so

$$f(x,z) = [(1-p)\varphi(x,\mu_1,\sigma_1)]^{1-z} \cdot [p\varphi(x,\mu_2,\sigma_2)]^{z}$$

$$L(p,\mu_1,\sigma_1,\mu_2,\sigma_2) = f(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^{n} f(x_i,z_i) =$$

$$\prod_{i=1}^{n} \Big[[(1-p)\varphi(x_i,\mu_1,\sigma_1)]^{1-z_i} \cdot [p\varphi(x_i,\mu_2,\sigma_2)]^{z_i} \Big]$$

$$l(p,\mu_1,\sigma_1,\mu_2,\sigma_2) = \log(L(p,\mu_1,\sigma_1,\mu_2,\sigma_2)) =$$

$$\sum_{i=1}^{n} ((1-z_i)(\log(1-p) + \log\varphi(x_i,\mu_1,\sigma_1)) +$$

$$z_i(\log p + \log\varphi(x_i,\mu_2,\sigma_2))) =$$

$$\sum_{i=1}^{n} ((1-z_i)\log(1-p) + z_i \log p) +$$

$$\sum_{i=1}^{n} ((1-z_i)\log\varphi(x_i,\mu_1,\sigma_1) + z_i \log\varphi(x_i,\mu_2,\sigma_2)$$

to simplify a bit let's assume $\sigma_1 = \sigma_2 = 1$, then

$$l(\mu_1, \mu_2) =$$
$$\sum (1 - z_i) \log \varphi(x_i; \mu_1) + \sum z_i \log \varphi(x_i; \mu_2)$$

$$\frac{dl}{d\mu_1} = \sum (1 - z_i) \frac{d}{d\mu_1} \left(\log \frac{1}{\sqrt{2\pi}} - \frac{1}{2}(x_i - \mu_1)^2 \right)$$
$$\sum (1 - z_i)(x_i - \mu_1) = 0$$

$$\widehat{\mu_1} = \frac{\sum (1-z_i) x_i}{\sum (1-z_i)}$$

$$\widehat{\mu_2} = \frac{\sum z_i x_i}{\sum z_i}$$

Notice that $\hat{\mu}_1$ is just the mean of the observations from group 1, which we can identify because we know the z's. It is therefore easy to guess what will happen if we also let the σ's float: $\hat{\sigma}_i$ is the sample standard deviation of the events in group i.

So if we knew the z_i's this would be a simple problem. On the other hand,

$$\begin{aligned}
&E[Z|X = x] = \\
&0 \cdot P(Z = 0|X = x) + 1 \cdot P(Z = 1|X = x) = \\
&P(Z = 1|X = x) = \\
&\frac{P(Z = 1, X = x)}{P(X = x)} = \\
&\frac{p\phi(x; \mu_2, \sigma_2)}{(1 - p)\phi(x; \mu_1, \sigma_1) + p\phi(x; \mu_2, \sigma_2)}
\end{aligned}$$

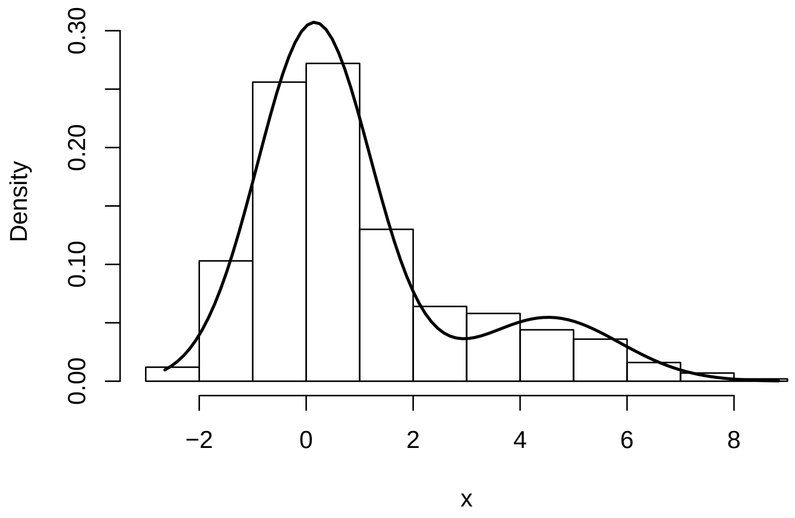
so if we knew the parameters we could estimate each of the z_i's. This is then the basic idea of the EM algorithm:

- in the M step assume you know the $z_1, .., z_n$ and estimate the parameters.
- in the E step use these parameters to estimate the $z_1, .., z_n$.

Here is the implementation:

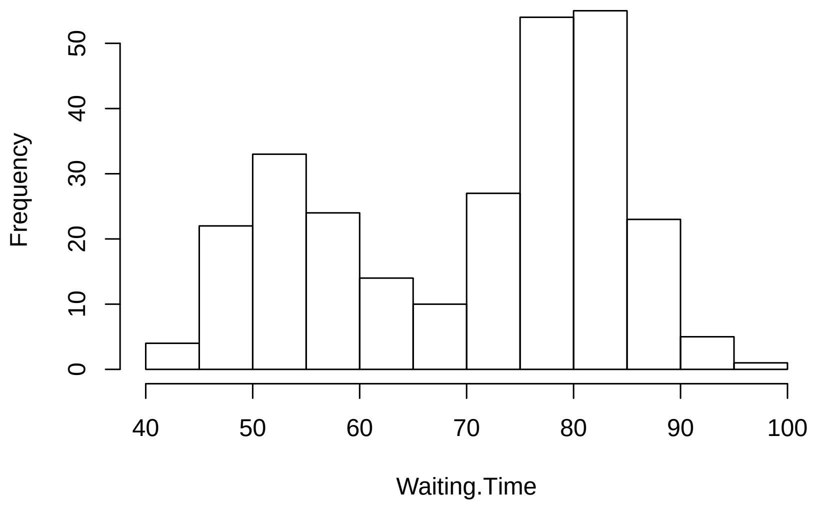
```
emNormalMix <- function(x,  p=0.5, mu=c(0, 3.5),
                sigma=c(1, 1), start=c(p, mu, sigma)) {
    loglike <- function(x, p, mu)
        sum(log((1-p)*dnorm(x, mu[1])+p*dnorm(x, mu[2])))
    n <- length(x)
    a <- start
    p <- a[1]
    mu <- a[2:3]
    sigma <- a[4:5]
    z <- ifelse(p*dnorm(x, mu[2])/
        ((1-p)*dnorm(x, mu[1])+p*dnorm(x, mu[2]))>0.5, 1, 0)
    print(round(c(a, loglike(x, p, mu)), 3))
    repeat {
        aold <- a
        z <- ifelse(p * dnorm(x, mu[2])/((1 - p)*dnorm(x, mu[1]) +
            p * dnorm(x, mu[2])) > 0.5, 1, 0)
        p <- sum(z)/n
        mu[1] <- mean(x[z == 0])
        sigma[1] <- sd(x[z == 0])
        mu[2] <- mean(x[z == 1])
        sigma[2] <- sd(x[z == 1])
        a <- c(p, mu, sigma)
        print(round(c(a, loglike(x, p, mu)), 3))
        if (sum(abs(a - aold)) < 1e-04)
            break
    }
    x.points <- seq(min(x), max(x), length = 100)
    y.points <- (1 - p) * dnorm(x.points, mu[1], sigma[1]) + p * dnorm(x.points, mu[2],
    hist(x, freq=FALSE, main="", ylim=c(0, max(y.points)))
    lines(x.points, y.points, lwd=2)

}
n <- 1000
p <- 0.3
mu <- c(0, 3.5)
sigma <- c(1, 2)
z <- sample(c(0, 1), size=n, replace=TRUE, prob=c(1-p, p))
x <- (1 - z) * rnorm(n, mu[1], sigma[1]) +
          z * rnorm(n, mu[2], sigma[2])
emNormalMix(x)
```

```
## [1]      0.500      0.000      3.500      1.000      1.000 -2189.494
## [1]      0.247     -0.024      3.908      0.911      1.542 -2016.582
## [1]      0.213      0.061      4.219      0.979      1.433 -1995.176
## [1]      0.195      0.112      4.393      1.025      1.373 -1990.086
## [1]      0.185      0.142      4.493      1.053      1.337 -1989.424
## [1]      0.181      0.154      4.535      1.064      1.322 -1989.632
## [1]      0.181      0.154      4.535      1.064      1.322 -1989.632
```



Let's apply the algorithm to a famous data set, the *Old Faithful* data, specifically the length of the Waiting.Time:

```
attach(faithful)
hist(Waiting.Time, main="")
```



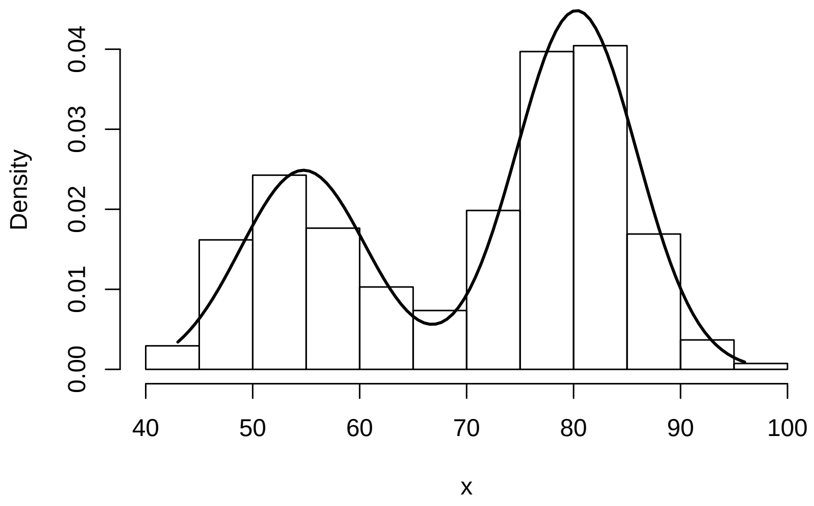
To run the routine we need some starting values. It seems that the two groups are those with data less than and more than 70, so

```
mu <- c(mean(Waiting.Time[Waiting.Time<70]), mean(Waiting.Time[Waiting.Time>70]))
sigma <- c(sd(Waiting.Time[Waiting.Time<70]), sd(Waiting.Time[Waiting.Time>70]))
print(c(mu,sigma), digits=2)
```

```
## [1] 55.2 80.7  6.3  5.3
```

```
emNormalMix(Waiting.Time, mu = mu, sigma = sigma)
```

```
## [1]     0.500    55.155    80.745     6.267     5.268 -4892.593
## [1]     0.632    54.750    80.285     5.895     5.627 -4856.739
## [1]     0.632    54.750    80.285     5.895     5.627 -4856.739
```



The EM algorithm was originally invented by Dempster in 1977 to deal with a common problem in Statistics called **censoring**:

say we are doing a study on survival of patients after cancer surgery. Any such study will have a time-limit after which we will have to start with the data analysis, but hopefully there will still be some patients who are alive, so we don't know their survival times, but we do know that the survival times are greater than the time that has past sofar. We say the data is censored at time T.

The number of patients with survival times >T is important information and should be used in the analysis. If we order the observations into (x_1, .., x_n) the uncensored observations (the survival times of those patients that are now dead) and (x_{n+1}, .., x_{n+m}) the censored data, the likelihood function can be written as

$$L(\theta|x) = [1 - F(T|\theta]^m \prod_{i=1}^{n} f(x_i|\theta)$$

because all we know of the censored data is that

$$P(X_i > T) = 1 - F(T|\theta)$$

If we had also observed the survival-times of the censored patients, say z=(z_{n+1}, .., z_{n+m}) we could have written the complete-data likelihood

$$L^c(\theta|\mathbf{x},\mathbf{z}) = \prod_{i=1}^{n} f(x_i|\theta) \prod_{i=n+1}^{m} f(z_i|\theta)$$

and again we can use the EM algorithm to estimate θ:

- in the M step assume you know the $z_1, .., z_n$ and estimate θ.
- in the E step use θ to estimate the $z_1, .., z_n$

####**Example** Say $X_i \sim \text{Exp}(\theta)$ and we have data $(x_1, .., x_n)$ and we know that m observations were censored at T. Now

$$L^c(\theta|x,z) = \prod_{i=1}^{n} f(x_i|\theta) \prod_{i=n+1}^{m} f(z_i|\theta) =$$

$$\prod_{i=1}^{n} \theta e^{-\theta x_i} \prod_{i=n+1}^{m} \theta e^{-\theta z_i} = \theta^{n+m} \exp[-\theta(\sum x_i + \sum z_i)]$$

$$l^c(\theta) = \log L^c(\theta|x,z) = (n+m)\log\theta - \theta(\sum x_i + \sum z_i)$$

$$\frac{dl^c(\theta)}{d\theta} = \frac{n+m}{\theta} - \sum x_i + \sum z_i = 0$$

so $\theta = \frac{n+m}{\sum x_i + \sum z_i}$

Recall that $F(x) = 1 - e^{-\theta x}$, so

$$P(Z < t|Z > T) = \frac{P(T<Z<t)}{P(Z>T)} = \frac{(1-e^{-\theta t})-(1-e^{-\theta T})}{1-(1-e^{-\theta T})} = \frac{e^{-\theta T}-e^{-\theta t}}{e^{-\theta T}}$$

$$f_{Z|Z>T}(z|t) = \frac{\theta e^{-\theta t}}{e^{-\theta T}},\ t > T$$

$$E[Z|Z > T] = \int_T^\infty t\frac{\theta e^{-\theta t}}{e^{-\theta T}}dt = \frac{1}{e^{-\theta T}}\int_T^\infty t\theta e^{-\theta t}dt =$$

$$\frac{1}{e^{-\theta T}}\int_0^\infty (x+T)\theta e^{-\theta(x+T)}dt =$$

$$\frac{e^{-\theta T}}{e^{-\theta T}}\left(\int_0^\infty x\theta e^{-\theta x}dt + T\int_0^\infty \theta e^{-\theta x}dt\right) = \frac{1}{\theta} + T$$

so the EM algorithm proceeds as follows:

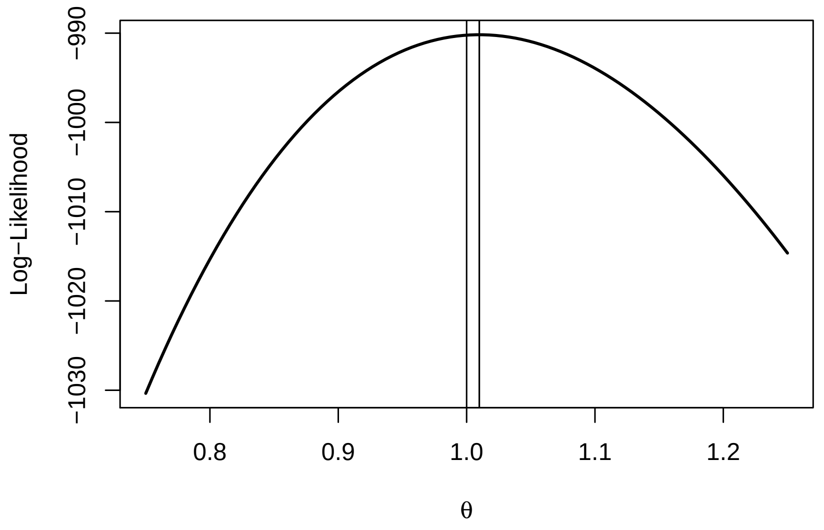
- in the M step assume you know the $z_1, .., z_n$ and estimate $\theta = 1/mean(x_1, .., x_n, z_{n+1}, .., z_{n+m})$.

- in the E step use θ to estimate the $z_1, .., z_n = 1/\theta + T$

```r
emCensExp <- function (n = 1000, T = 1, m = 0, theta = 1, start = theta)
{
    loglike <- function(x, theta, m, T) {
        -theta * T * m + sum(log(dexp(x, theta)))
    }
    x <- rexp(n, theta)
    u <- seq(theta * 0.75, 1.25 * theta, length = 100)
    ll <- rep(0, 100)
    for (i in 1:100) ll[i] = loglike(x, u[i], m, T)
    plot(u, ll, type = "l", lwd = 2, xlab = expression(theta),
        ylab = "Log-Likelihood")
    truetheta <- theta
    theta <- start
    print(round(c(theta, loglike(x, theta, m, T)), 3))
    abline(v = theta)
    repeat {
        thetaold <- theta
        z <- rep(1/theta + T, m)
        theta <- 1/mean(c(x, z))
        print(round(c(theta, loglike(x, theta, m, T)), 3))
        abline(v = theta)
        if (abs(theta - thetaold) < 1e-04)
            break
    }
    theta
}
```

Let's first check the case without censoring:

```r
emCensExp()
```

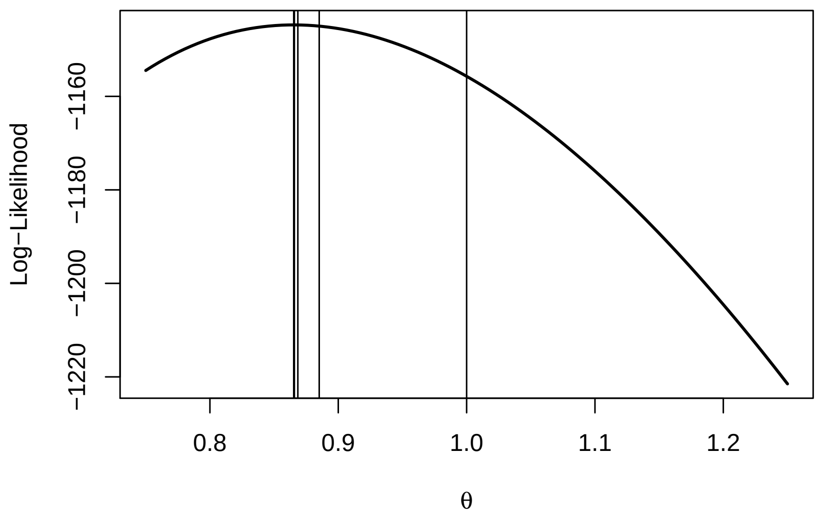


```
## [1]    1.000 -990.224
## [1]    1.010 -990.176
## [1]    1.010 -990.176
```

```
## [1] 1.009872
```

And now with 200 censored events:

```
emCensExp(m=200)
```



```
## [1]     1.000 -1155.709
## [1]     0.885 -1144.974
## [1]     0.869 -1144.721
## [1]     0.866 -1144.715
## [1]     0.865 -1144.714
## [1]     0.865 -1144.714
```

```
## [1] 0.8652844
```

####**Example**

nonparametric density estimation using Bernstein polynomials.

say we have data (X_1, .., X_n) from some continuous but unknown density f, and we want to estimate f(x) for any x. One idea to do this is to approximate the function f by a polynomial of some degree d, denoted by $p_d(x)$, with the coefficients estimated via maximum likelihood. A big problem when doing this is that polynomials are not natural choices for densities because they easily have negative values, and just finding out where we have $p_d(x)<0$ is a nontrivial problem if d>2. One way around this issue is to use polynomials that are naturally non-negative, and a popular choice are so called *Bernstein polynomials*:

$$x^k(1-x)^{d-k}$$

if $0<x<1$ and $k=0,..,d$

of course these are essentially Beta densities, which leads to another nice feature, namely it is easy to normalize the polynomials so they are proper densities:

$$b(k,d,x)=\frac{(d+1)!}{k!(d-k)!}x^k(1-x)^{d-k}$$

It can be shown that any density on [0,1] can be approximated uniformly by a linear combination of Bernstein polynomials, that is for any $\epsilon>0$ there exists a d and numbers a_0,..,a_d with a_0+..+a_d=1 such that

$$\max\left\{|f(x)-\sum_{k=0}^d a_k b(k,d,x)| : 0<x<1\right\}<\epsilon$$

Bernstein polynomials are defined on [0,1], if the density f is positive on the interval [A,B] we need to first use the transform $y=(x-A)/(B-A)$.

If f is defined on $[A,\infty)$ or $(-\infty,\infty)$, other transforms can be used but we won't discuss that here.

Let's set

$$p(x;a_0,..,a_d)=\sum_{k=0}^d a_k b(k,d,x)$$

so, how can we find a_0, .., a_d as well as the smallest d for which this is true? Let's assume for a moment that d is known, then we can estimated a_0, .., a_d via maximum likelihood, that is we we need to find

$$\max\left\{\sum_x \log(p(x;a_0,..,a_d)); 0<a_0,..,a_d<1 \text{ and } a_0+..+a_d=1\right\}$$

In calculus we have the method of Lagrange multipliers for this type of constraint maximization, but here (if d>1) this leads to a nonlinear system of equations which can not be solved analytically.

Moreover, this is also a difficult problem numerically, because most standard minimization algorithm (such as Newton-Raphson) do not allow for these types of contraints.

Instead we can use the EM algorithm. Even easier, because the Bernstein polynomials do not have parameters we don't even need the M step!

The algorithm:

use as start value a = rep(1, d+1)/(d+1)

at each iteration set

$$w_k = \text{mean}(a_k b(k, d, x)/p(x))$$

$k = 0, .., d$

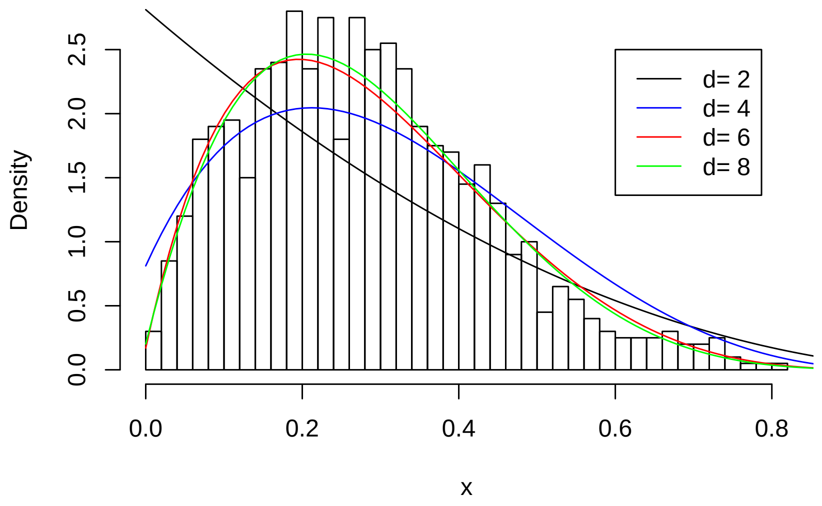
and stop when (say) $\sum |a_k - w_k| < 0.001$

```r
dBernstein <- function(x, a, returnMatrix=FALSE) {
  d <- length(a)-1
  n <- length(x)
  Z <- matrix(0, n, d+1)
  for(i in 0:d) Z[, i+1] <- a[i+1]*dbeta(x, i+1, d+1-i)
  if(returnMatrix) return(Z)
  apply(Z, 1, sum)
}

fitBernstein <- function(x, d) {
  a <- rep(1, d+1)/(d+1)
  k <- 0
  repeat {
    k <- k+1
    Z <- dBernstein(x, a, returnMatrix=TRUE)
    p <- apply(Z, 1, sum)
    for(i in 0:d) Z[, i+1] <- Z[, i+1]/p
    w <- apply(Z,  2, mean)
    if( sum(abs(a-w))<0.01) break
    a <- w
    if(k>100) break
  }
  a
}
```

Here is an example:

```r
x <- rbeta(1000, 2, 5)
hist(x, 50, freq=FALSE, main="")
t <- seq(0, 1, length=100)
cols <- c("black", "blue", "red", "green")
```

```
for(i in 1:4) {
  a <- fitBernstein(x, d=2*i)
  lines(t, dBernstein(t, a=a), col=cols[i])
}
legend(0.6, 2.5, legend=paste("d=", 2*1:4), lty=rep(1, 4), col = cols)
```



How can we find a good degree d? We can use the likelihood ratio test:

say we want to compare the fit of p_d with that of p_{d+1}. Let p_d^* be p_d evaluated at the data x using the respective mle's as coefficients. Then by the large sample theory of the likelihood ratio test

$$(-2)\left(\sum \log p_d^* - \sum \log p_{d+1}^*\right) \sim \chi^2(1)$$

so we will test $d = 1$ vs $d = 0$. If we reject we test $d = 2$ vs $d = 1$ and so on until we fail to reject the null.

```
a_0 <- fitBernstein(x, d=0)
p_0star <- dBernstein(x, a_0)
a_1 <- fitBernstein(x, d=1)
p_1star <- dBernstein(x, a_1)
chi2 <- (-2)*(sum(log(p_0star))-sum(log(p_1star)))
crit <- qchisq(0.9, 1)
cat("Critical value=", round(crit, 3), "\n")
```

```
## Critical value= 2.706
```

```
d <- 1
cat("d =", d-1, "Chisquare Statistic =", round(chi2, 3),"\n")
```

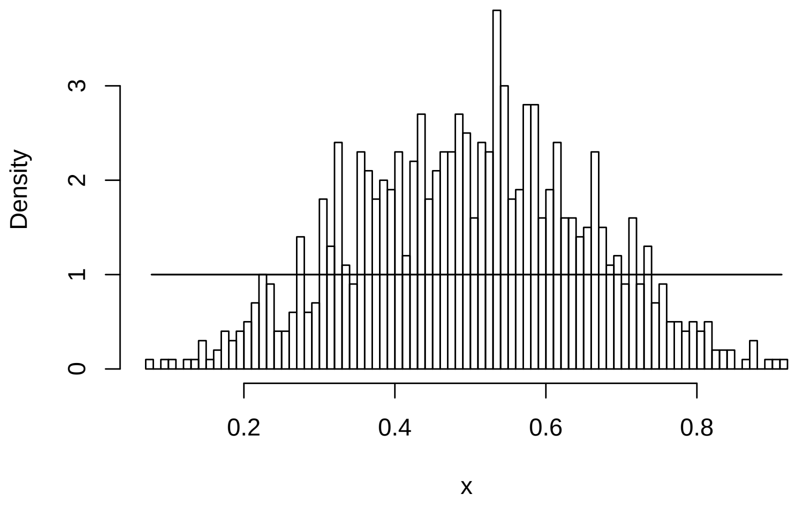
```
## d = 0 Chisquare Statistic = 660.45
```

```
repeat {
  d <- d+1
  p_0star <- p_1star
  p_1star <- dBernstein(x, fitBernstein(x, d=d))
  chi2 <- (-2)*(sum(log(p_0star))-sum(log(p_1star)))
  cat("d =", d-1, "Chisquare Statistic =", round(chi2, 3),"\n")
  if(chi2<crit) break
  if(d>20) break
}
```

```
## d = 1 Chisquare Statistic = 94.972
## d = 2 Chisquare Statistic = 49.175
## d = 3 Chisquare Statistic = 136.118
## d = 4 Chisquare Statistic = 65.305
## d = 5 Chisquare Statistic = 3.777
## d = 6 Chisquare Statistic = 1.483
```

There is a problem, though: consider this example:

```
x <- sort(rbeta(1000, 5, 5))
hist(x, 100, freq=FALSE, main="")
a_0 <- fitBernstein(x, d=0)
p_0star <- dBernstein(x, a_0)
lines(x, p_0star, type="l")
a_1 <- fitBernstein(x, d=1)
p_1star <- dBernstein(x, a_1)
lines(x, p_1star, type="l")
```



```r
chi2 <- (-2)*(sum(log(p_0star))-sum(log(p_1star)))
crit <- qchisq(0.9, 1)
cat("Critical value=", round(crit, 3), "\n")
```

```
## Critical value= 2.706
```

```r
d <- 1
cat("d =", d-1, "Chisquare Statistic =", round(chi2, 3),"\n")
```

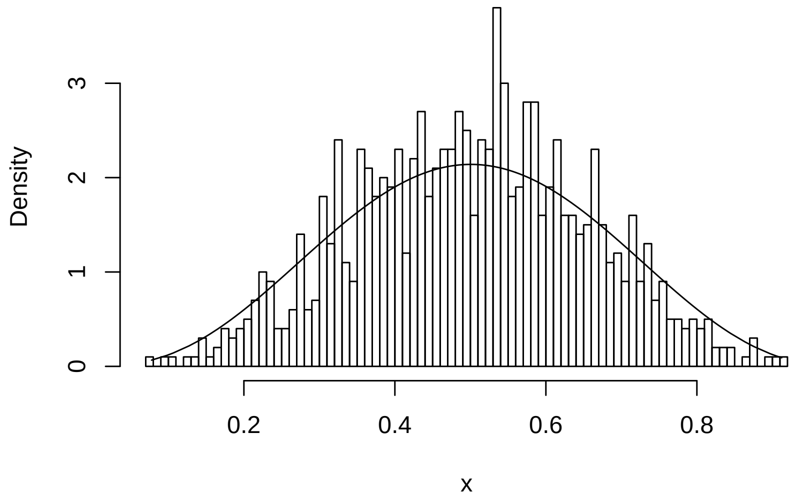
```
## d = 0 Chisquare Statistic = 0
```

```r
repeat {
  d <- d+1
  p_0star <- p_1star
  p_1star <- dBernstein(x, fitBernstein(x, d=d))
  chi2 <- (-2)*(sum(log(p_0star))-sum(log(p_1star)))
  cat("d =", d-1, "Chisquare Statistic =", round(chi2, 3),"\n")
#  if(chi2<crit) break
  if(d>10) break
}
```

```
## d = 1 Chisquare Statistic = 583.944
## d = 2 Chisquare Statistic = 0.696
## d = 3 Chisquare Statistic = 224.063
## d = 4 Chisquare Statistic = 1.928
## d = 5 Chisquare Statistic = 87.554
## d = 6 Chisquare Statistic = 0.165
## d = 7 Chisquare Statistic = 21.384
## d = 8 Chisquare Statistic = 1.57
## d = 9 Chisquare Statistic = 0.918
## d = 10 Chisquare Statistic = 1.74
```

so the routine would stop already at d=1 although although obviously both fits are very bad. That is actually the problem, both are equally bad! In general in addition to the hypothesis test we should also make a visual check to see that the fit looks reasonably ok. The next time the test rejects the null is for d=6:

```r
hist(x, 100, freq=FALSE, main="")
a <- fitBernstein(x, d=6)
p <- dBernstein(x, a)
lines(x, p, type="l")
```



and this looks quite alright!

7.3 The Symmetric Random Walk in R^d

The symmetric random walk in R^d is one of the classic stochastic processes. It works as follows: Let S_d be the integer lattice in R^d, that is

$S_d= \{(i_1,..,i_d): i_k \in Z\}$

and for the process $\{X_n, n=1,2,..\}$ we have

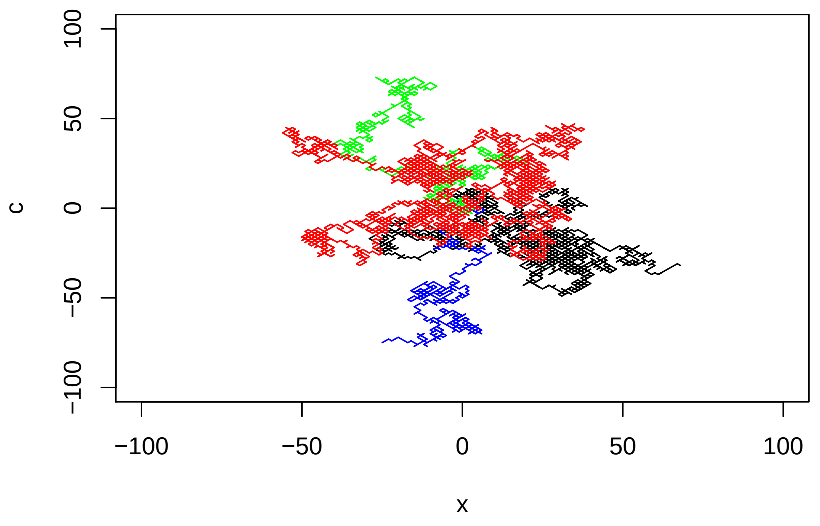
$P(X_{n+1}= (i_1,..,i_d) \mid X\sim n \sim= (j_1,..,j_d)) = 1/(2d)$

if $i_k = j_k \pm 1$ for one k=1,..,d and $i_l=j_l$ for all l=1,..,d, $l \neq k$

In other words if X_n is at some point of the lattice it randomly chooses a neighboring point and moves there.

Here is an illustration for d=2:

```r
rw2.plot<- function(A=100) {
    cols <- c("black", "blue", "green", "red")
    plot(c, xlim=c(-A, A), ylim=c(-A, A), type="n")
    for(k in 1:4) {
      x <- c(0, 0)
      repeat {
        y <- x + sample(c(-1, 1), size=2, replace=TRUE)
        segments(x[1], x[2], y[1], y[2], col=cols[k])
        x <- y
        if(sum(abs(y))>=100) break
      }
    }
}
rw2.plot()
```



There are a lot of interesting questions one can investigate for the random walk. We will consider the following: Let $N_{d,A}$ be the first time that the process, having started at the origin, is a distance A from the origin. What is $E[N_{d,A}]$?

"Distance" here is defined as the minimum number of jumps needed to get back to the origin. Say we are at the point $(i_1,..,i_d)$, then it is easy to see that the distance is

$D = |i_1|+\dots+|i_d|$

It is always a good idea to start with a simple case, so let's look at d=1. Here it is very easy way to generate an observation:

```
x <- 0
n <- 0
repeat {
  n <- n+1
  x <- x+sample(c(-1,1), 1)
  if(abs(x)>=A) break
}
```

and this is done for $M=10^4$ in

```
rwd1 <-
function (which = 1, A = 5, M = 10000)
{
    tm <- proc.time()
    N <- rep(0, M)
```

```r
    if (which == 1) {
        for (i in 1:M) {
            x <- 0
            repeat {
                N[i] <- N[i] + 1
                x <- x + sample(c(-1, 1), 1)
                if (abs(x) == A)
                  break
            }
        }
    }
    if (which == 2) {
        for (i in 1:M) {
            x0 <- 0
            repeat {
                x <- x0 + cumsum(sample(c(-1, 1),
                                        size = 4*A,
                                        replace = TRUE))
                if (max(abs(x)) >= A)
                  break
                x0 <- x[4 * A]
                N[i] <- N[i] + 4 * A
            }
            N[i] <- N[i] + seq_along(x)[abs(x) == A][1]
        }
    }
    if (which == 3) {
        for (i in 1:M) {
            N[i] <- rwd1C(A)
        }
    }
    if (which == 4)
        N <- rwd1aC(A, M)
    print(proc.time() - tm)
    mean(N)
}
rwd1()
```

```
##    user  system elapsed
##    3.11    0.23    3.67
```

```
## [1] 24.783
```

But there is a problem: even for just A=5 and M=10^4 this takes quite a while.

We probably should use M=10^5, and we need to run this for for (say) A=2:1:100, so this just is way to slow.

How can we speed things up? I will discuss four possible improvements:

7.3.1 Do the Math!

In general the best way to go is to do as much as possible theoretically. So we should really get a good book about Stochastic Processes and see what we can find.

7.3.2 Improve your R Routine

R is a fantastic language for writting programs, but it is not very fast. Still there are a number of tricks we can use to improve performance:

- Vectorize!

One major source of slow R are loops, and we actually have two of them, nested. So first we should try to get rid of some. Our inner loop looks like this:

```
repeat {
  N[i]=N[i]+1
  x=x+sample(c(-1,1),1)
  if(sum(abs(x))==A) break
}
```

Let's try the following: we find a sequence of ± 1's, and use cumsum to get x. If $|x| \geq A$ we find where that happens the first time (=N), otherwise generate another sequence and "add" it. Here is the routine:

```
x0 <- 0
repeat {
  x <- x0 + cumsum(sample(c(-1,1), size=4*A, replace=TRUE))
  if(max(abs(x)) >= A) break
  x0 ,- x[4*A]
  N[i] <- N[i]+4*A
}
N[i] <- N[i] + seq_along(x)[abs(x)==A][1]
```

this is done in *rwd1(2)*, and this is almost 10 times faster!

Notice I generate sequences of length 4A, one could play around with this and likely find an even better choice.

- Parallelize!

Many of today's computers have multiple cores (processors), but generally only one is used at a time. Also, simulation problems are usually *embarrasingly parallel*, that is they do the same thing (with different random numbers) over and over again. So we can speed up calculation by doing parallel processing. On a Windows machine, this is done using the "snow" library:

```
library(snow)
cl <- makeCluster(rep("localhost",6), type = "SOCK")
```

```r
clusterCall(cl, rwd1, which=2, M=1e5)
stopCluster(cl)
```

There is a lot of overhead in calling all the processors, so this is worth it only for routines that take at least a few minutes to run.

- Rcpp

Finally often we can speed things up dramatically by changing parts of the routine to C++. Let's look again at the inner loop:

```r
x <- 0
n <- 0
repeat {
  n <- n+1
  x <- x+sample(c(-1,1), 1)
  if(abs(x)>=A) break
}
```

It is easy to turn this into a C++ routine. We need to

- every variable is declared explicitly
- in C++ repeat is called do
- Rcpp has no sample command, so we use runif()<0.5
- make sure every line ends with ;

with this the code in C++ looks like this:

```
int k=0;
  int z=0;
NumericVector u;
do {
  &nbsp;&nbsp;&nbsp;k++;
  &nbsp;&nbsp;&nbsp;u=runif(1);
  &nbsp;&nbsp;&nbsp;if(u[0]<0.5) z--;
  &nbsp;&nbsp;&nbsp;else z++;
  } while (abs(z)<A);
return k;
```

Finally we need to add the usual stuff on top and save all of it in a file with the .cpp extension.

To make it available in R do

```r
library(Rcpp)
sourceCpp(paste(getwd(),"/rwd1.cpp",sep=""))
```

and it is run with *rwd1(3)*.

It turns out to be almost 4 times faster than rwd1(2)!

We can go another step and also replace the outer for loop, done in rwd1a.cpp and run with rwd1(4).

So compared to the first routine rwd1(1) rwd1(4) is about **75** times faster!

7.3.3 $\mathbf{R^d}$

Having written the basic routine in R, it is easy to change it to work in R^d. All we need to do is now first pick a coordinate and then do ± 1 that coordinate.

Running *rwd* it turns out to be actually a little faster than in R^1:

```r
rwd <-
function (which = 1, d = 1, A = 5, M = 10000)
{
    Dist <- function(x) sum(abs(x))
    tm <- proc.time()
    N <- rep(0, M)
    if (which == 1) {
        for (i in 1:M) {
            x <- rep(0, d)
            repeat {
                N[i] <- N[i] + 1
                k <- sample(1:d, 1)
                x[k] <- x[k] + sample(c(-1, 1), 1)
                if (Dist(x) == A)
                  break
            }
        }
    }
    if (which == 2) {
        jumps <- make_jumps(d)
        for (i in 1:M) {
            x0 <- rep(0, d)
            repeat {
                x <- x0 + t(apply(jumps[, sample(1:(2 * d), size = 4 *
                  A, replace = TRUE), drop = FALSE], 1, cumsum))
                D <- apply(x, 2, Dist)
                if (max(D) >= A)
                  break
                x0 <- c(x[, 4 * A])
                N[i] <- N[i] + 4 * A
            }
            N[i] <- N[i] + c(1:(4 * A))[D == A][1]
        }
    }
    if (which == 3) {
        for (i in 1:M) {
            N[i] <- rwdC(d, A)
        }
    }
    if (which == 4)
        N <- rwdaC(d, A, M)
    print(proc.time() - tm)
    mean(N)
}
rwd(1,d=1,5)
```

```
##    user  system elapsed
##    5.80    0.03    5.83
```

```
## [1] 24.76
```

```
rwd(1,d=2,5)
```

```
##    user  system elapsed
##    2.09    0.00    2.12
```

```
## [1] 15.1058
```

```
rwd(1,d=3,5)
```

```
##    user  system elapsed
##    1.52    0.02    1.55
```

```
## [1] 11.041
```

In R[1] it takes longer because we do one more calculation inside the repeat loop.

How about the vectorized version? This is a bit harder. First we need to have a matrix with all possible changes, jumps, see

```
make_jumps <- function(d) {
    jumps <- matrix(0, d, 2 * d)
    for (i in 1:d) jumps[i, ((i - 1) * 2 + 1:2)] <- c(-1, 1)
    jumps
}
```

Next we randomly select 4A of these jumps

```
jumps[, sample(1:(2*d), size=4*A, replace=TRUE)
```

Then we need do "add them up", again using cumsum:

```
x <- x0 + t(apply(jumps[, sample(1:(2*d),
                       size=4*A,
                      replace=TRUE), drop=FALSE], 1, cumsum))
```

Notice the drop=FALSE argument, which assures that the result is not turned into a vector in the case d=1, and the t(), which is necessary because the apply(„cumsum) inverts the matrix.

I also have a small routine dist which calculates the distance from the origin.

Now we find

```
rwd(1, 2, 5, M=1e5)
```

```
##    user  system elapsed
##   20.33    0.12   20.61
```

```
## [1] 15.01588
```

```
rwd(2, 2, 5, M=1e5)
```

```
##    user  system elapsed
##   13.89    0.05   14.00
```

```
## [1] 14.96574
```

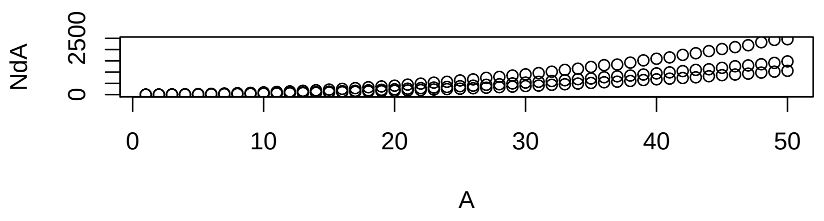
so there is actually not much gain here! The reason is that the apply(„cumsum) command is quite slow.

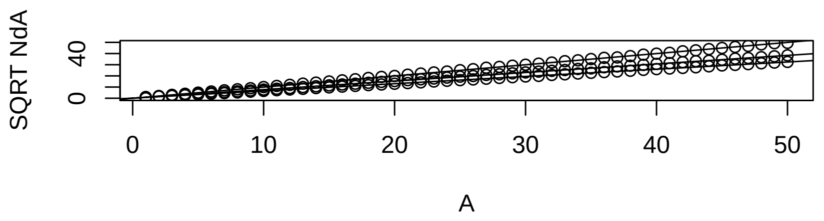
7.3.4 Back to $E[N_{d,A}]$

Now we can run this routine to simulate N_{dA}, and estimate $E[N_{d,A}]$. The numbers for d=1,..,6 are in NdA.

```
NdA <-
structure(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
48, 49, 50, 0, 4, 9, 15.8, 24.8, 35.9, 49.7, 64, 80, 99.9, 120.8,
143.2, 170.4, 194.8, 224.1, 255.7, 288.3, 328.2, 363.5, 396.8,
443.6, 488, 531.8, 570.3, 629, 675.2, 745.3, 785.7, 847.3, 899.2,
957, 1018.6, 1097.9, 1151.3, 1229.1, 1306.1, 1334.1, 1420.7,
1520.1, 1592.1, 1652.3, 1763.3, 1836.6, 1931.9, 2023.2, 2106.6,
2191.4, 2323.5, 2428.1, 2458.4, 1, 2.7, 5.6, 9.7, 15.2, 21.5,
29.1, 37.8, 47.7, 59.6, 71.6, 83.5, 99.4, 114.9, 132.9, 151.7,
169.1, 190.8, 213.6, 236.6, 262.9, 287.1, 312.6, 341.2, 373.6,
396.8, 435.5, 460.1, 497.5, 534.4, 568.6, 602.2, 637, 686, 714.4,
770.1, 812.2, 845.6, 891.6, 942.5, 987, 1043.1, 1098.1, 1125.2,
1188.9, 1258.8, 1295.5, 1348.3, 1408.6, 1474.4, 1, 2.4, 4.4,
7.4, 11.1, 15.8, 21.1, 27.3, 34.9, 43, 51.9, 61.3, 70.9, 82.7,
94.8, 108.2, 121.9, 138.1, 152.2, 168.7, 183.7, 202, 225.1, 244.5,
262.1, 283.1, 306.5, 330.1, 354.8, 379.6, 401, 434.6, 458.9,
489.7, 518.1, 547.4, 574.5, 605.5, 646, 678.8, 708.9, 737.2,
767.4, 815.5, 862.9, 891.6, 931.9, 977.3, 1019.9, 1056.9), .Dim = c(50L,
4L), .Dimnames = list(NULL, c("A", "d=1", "d=2", "d=3")))
```

```
ENdA <-
function (s)
{
    par(mfrow = c(2, 1))
    plot(NdA[, 1], NdA[, 2], ylab = "NdA", xlab = "A")
    points(NdA[, 1], NdA[, 3])
    points(NdA[, 1], NdA[, 4])
    fit1 <- lm(sqrt(NdA[, 2])~NdA[, 1])
    fit2 <- lm(sqrt(NdA[, 3])~NdA[, 1])
    fit3 <- lm(sqrt(NdA[, 4])~NdA[, 1])
    plot(NdA[, 1], sqrt(NdA[, 2]), ylab = "SQRT NdA", xlab = "A")
    abline(fit1)
    points(NdA[, 1], sqrt(NdA[, 3]))
    abline(fit2)
    points(NdA[, 1], sqrt(NdA[, 4]))
    abline(fit3)
    cbind(coef(fit1), coef(fit2), coef(fit3))
}
ENdA()
```





```
##                      [,1]        [,2]       [,3]
## (Intercept) -0.04585626 0.06462512 0.0869832
## NdA[, 1]     1.00019201 0.76570467 0.6470283
```

We find a somewhat quadratic relationship.

Plotting SQRT(NdA) vs A shows a linear relationship.

Finding the least squares regression equations using a no-intercept model and plotting the slopes vs d we again see some curve

This time log(slope) vs log(d) turns it into a straight line, and doing the regression we find the equation

$\text{slope} = 1/\text{d}^{0.41}$

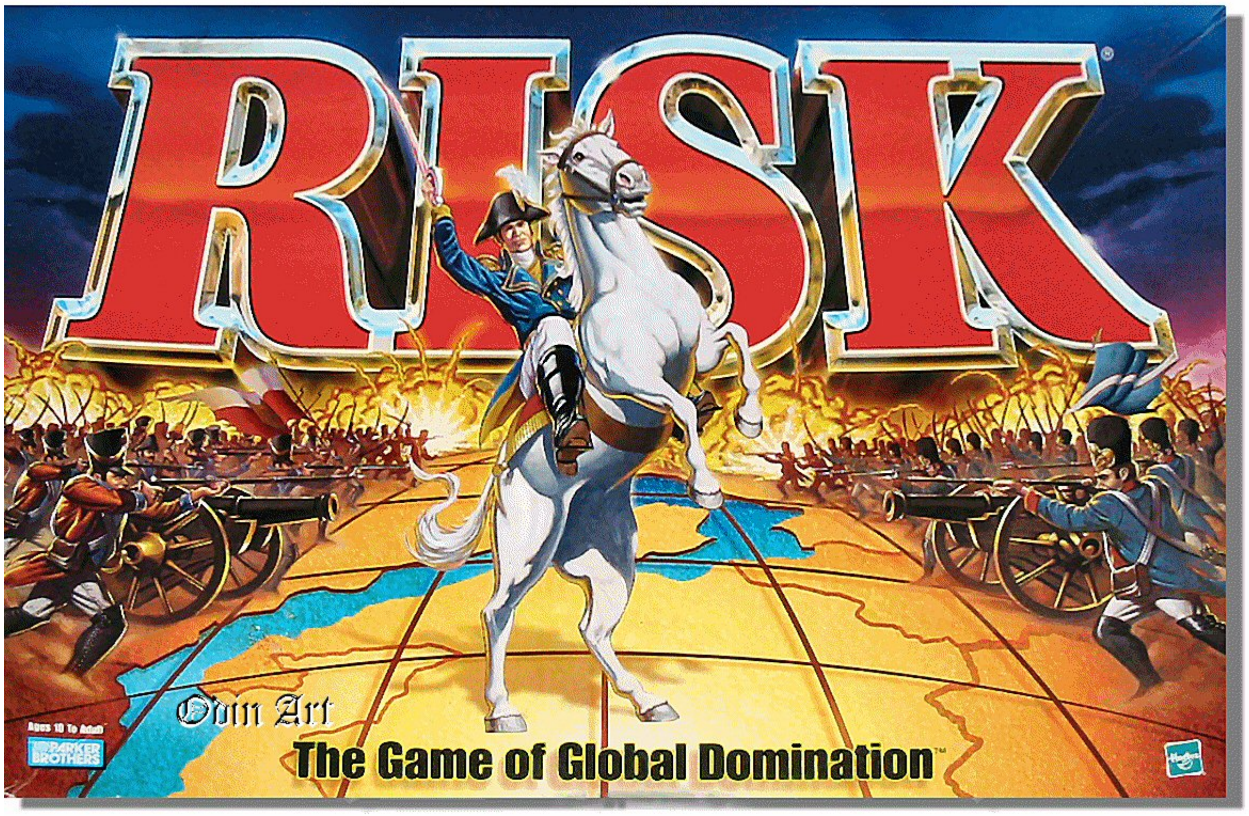
and putting it all together we get

$\text{E}[\text{N}_{\text{d,A}}] = (\text{A}/\text{d}^{0.41})^2 = \text{A}^2/\text{d}^{0.82}$

It is not a perfect fit, in order to improve it we would need to look for a better equation for the slope.

8 Student Exercises

8.1 Risk!



Risk is a strategy board game produced by Parker Brothers. It was invented by French film director Albert Lamorisse and originally released in 1957 as *La Conquête du Monde* ("The Conquest of the World") in France.

It was later bought by Parker Brothers and released in 1959 with some modifications to the rules as Risk: The Continental Game, then as Risk: The Game of Global Domination.

Risk is a turn-based game for two to six players. The standard version is played on a board depicting a political map of the Earth, divided into forty-two territories, which are grouped into six continents. The object of the game is to occupy every territory on the board and in doing so, eliminate the other players.

Players control armies with which they attempt to capture territories from other players, with results determined by dice rolls.

there are a lot of details to the game. We will investigate only one step, that of one country attacking another. This step is done by the owner of the attacking country and the owner of the defending country throwing dice.

Say the attacking country has n armies and the defending one has m. Then if $n \geq 4$ and $m \geq 3$ both throw 3 dice. These are ordered from largest to smallest and matched. For each match were the defending country is at least equal to the attacking country the attacking country looses one armee, otherwise the defending armee does.

Example A: 3 5 5 D: 2 5 6
Sort: A: 5 5 3 D 6 5 2
$5 < 6$ A looses one armee
$5 = 5$ A looses one armee
$3 > 2$ D looses one armee

so A looses 2 armees and D looses 1.

finally neither side can through more dice than they have available for fighting, and A always has to keep 1 armee to occupy the country, so for example if A has 3 armees and D has one, A throws 2 dice and D throws 1.

Example A 4 2 D 4, $4 \geq 4$, so A looses 1 armee.

If D has lost all his armees he looses the country and A takes it over. If A has only one armee left he can no longer attack. A can decide to stop attacking at any time.

Now here is our problem: we want to find a "simple" rule that we can use during an actual game that tells us the odds of winning if the Attacker has n armees for the attack and the defender has m.

8.2 Waiting Time

A call center is open from 8am in the morning until 5pm in the afternoon Monday to Friday. It is a small center with only one person answering phone calls. If a person is calling while the operator is busy they are put on hold. The call center decides to give a new caller some information on the expected waiting time. The information provided should be a message like this:

"Your call is very important to us. Our representative is currently busy with another customer. You are the next person in line, and your expected wait time is between 1 and 5 minutes"

or maybe

"Your call is very important to us. Our representative is currently busy with other customers. You are caller number 3 in line, and your expected wait time is between 15 and 27 minutes"

In order to figure out the wait times they keep track of all calls and waiting times for one month (24 working days).

You can get the data with

```
source("http://academic.uprm.edu/wrolke/esma5015/queuedata.txt")
```

Here it what the start looks like:

```
kable.nice(head(Days[[1]]))
```

Times	Events
8:21AM	C
8:27AM	C
8:31AM	F
8:36AM	F
8:38AM	C
8:44AM	F

this tells us that on day 1 the first call came in at 8:21 AM. It lasted until 8:31 AM. There was also a call coming in at 8:27 AM, so by the time the first call was finished there was 1 customer waiting in line.

At the end of day 1 we find

```
kable.nice(tail(Days[[1]]))
```

	Times	Events
95	4:16PM	F
96	4:22PM	F
97	4:36PM	C
98	4:44PM	C
99	5:01PM	F
100	5:14PM	F

As we can see, the last call was accepted just before 5 pm but those already on line were still getting taken care off (nice!)

Use the data provided and write a computer program that finds a 95% confidence interval for a person who is n^{th} in line and who calls in when the customer rep has already talked k minutes with the current customer.