Additional (advanced) exercises

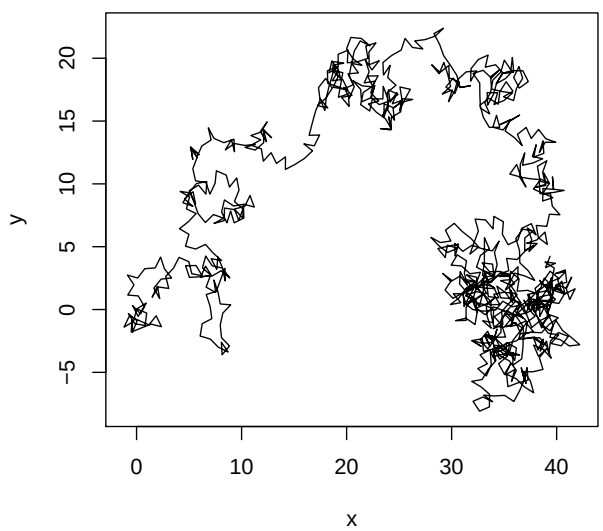
Unlike the exercises in the notes, whose solutions are provided in Chapter 19, you must attempt these questions yourself before I will share my answer with you. Please ask questions either in class or via Zoom. Then upload your solution to `http://doubleblind.dynu.net/GEOL0061` to download the model answer. Submissions won't be formally assessed, so there is no need to worry if your answer is wrong. Try, struggle, ask questions, and learn!

1 Random walk

Write a function that simulates a random walk:

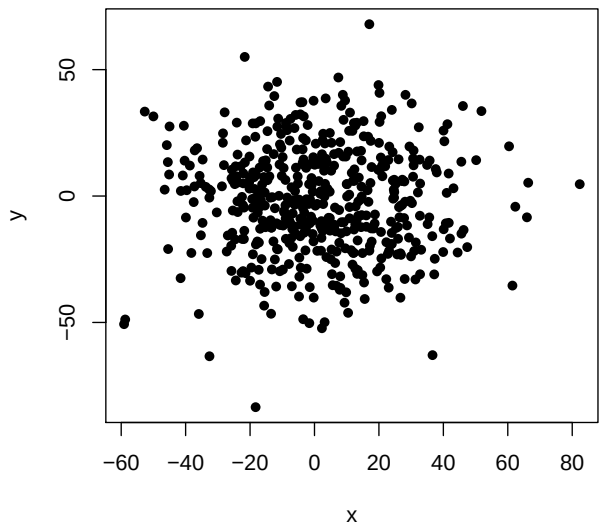
1. From a starting position of $x = 0$ and $y = 0$, move a virtual particle by a distance of 1 in a random direction.
2. Repeat $n = 1000$ times and plot the track of the particle as a line.

```
# random walk for one particle:
walk <- function(xy0=rep(0,2),n=1000){
   xy <- matrix(rep(xy0,n),ncol=2,byrow=TRUE)
   d <- runif(n,min=0,max=2*pi) # random directions
   for (i in 2:n){
      dx <- cos(d[i])
      dy <- sin(d[i])
      xy[i,1] <- xy[i-1,1] + dx
      xy[i,2] <- xy[i-1,2] + dy
   }
   return(xy)
}
xy <- walk()
plot(xy,type='l',xlab='x',ylab='y')
```



3. Repeat steps 1 and 2 for $N = 500$ virtual particles and visualise their final positions on a scatter plot.

```
n <- 1000
N <- 500
xyf <- matrix(NA,nrow=N,ncol=2)
for (i in 1:N){
   xyf[i,] <- walk(n=n)[n,]
}
plot(xyf,type='p',pch=16,xlab='x',ylab='y')
```

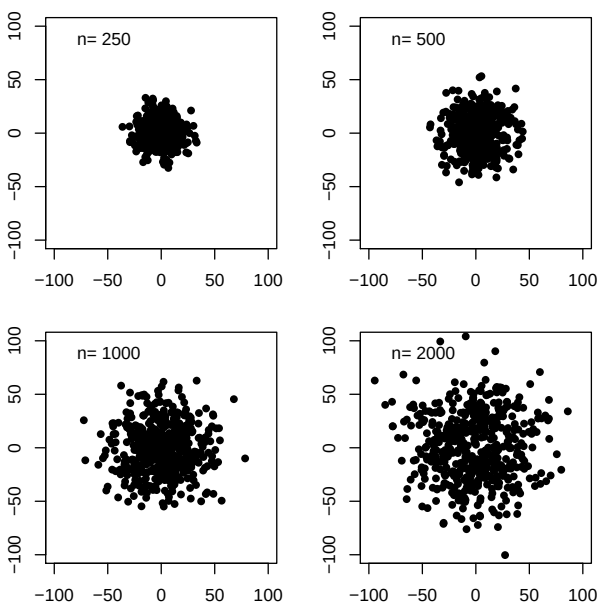

y
50
0
−50
−60
−40
−20
0
20
40
60
80
x

2 Diffusion

Using the code from exercise 1:

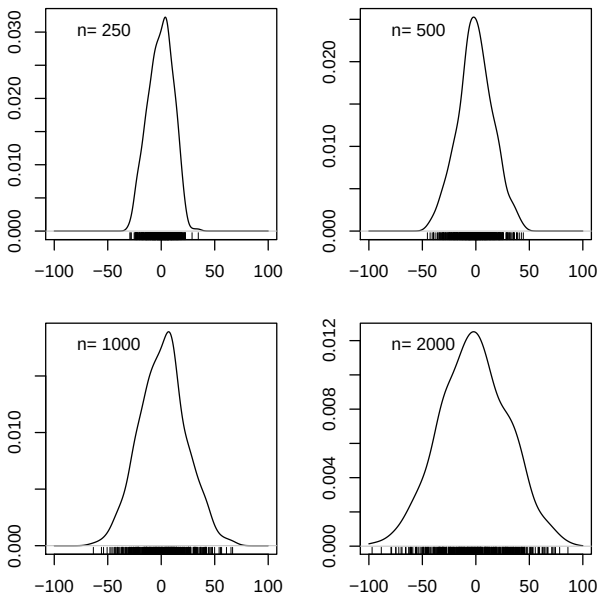
1. Repeat step 1.3 for $n = 250$, 500, 1000 and 2000 iterations and visualise the final positions on a 2×2 panel grid of scatter plots. Adjust the axis limits so that all four panels are plotted at the same scale.

```
cloud <- function(n=1000,N=500,plot=TRUE,...){
   xyf <- matrix(NA,nrow=N,ncol=2)
   for (i in 1:N){
      xyf[i,] <- walk(n=n)[n,]
   }
   if (plot) plot(xyf,pch=16,xlab='x',ylab='y',...)
   invisible(xyf) # returns the values without printing them at the console
}

par(mfrow=c(2,2),mar=rep(2,4))
lims <- c(-100,100)
ns <- c(250,500,1000,2000)
for (n in ns){
   cloud(n=n,xlim=lims,ylim=lims)
   legend('topleft',legend=paste('n=',n),bty='n')
}
```

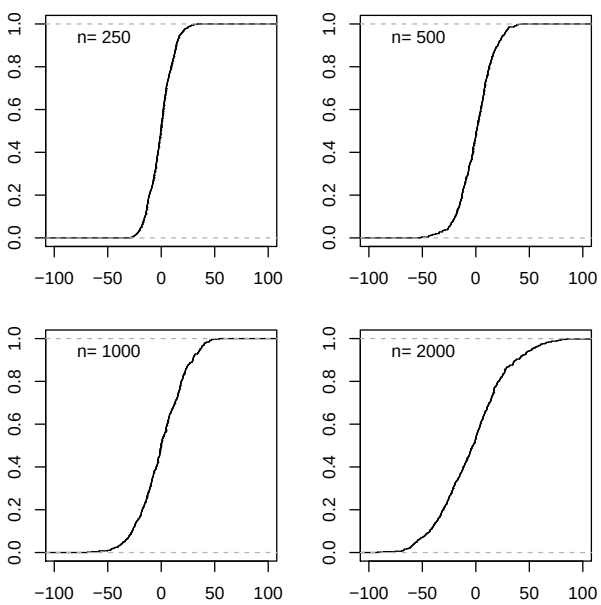


2. Plot the marginal distributions of the x-values as kernel density estimates and empirical cumulative distribution functions.

```
par(mfrow=c(2,2),mar=rep(2,4))
for (n in ns){
   xy <- cloud(n=n,plot=FALSE)
   d <- density(xy[,1],from=-100,to=100)
   plot(d,main='')
   rug(xy[,1])
   legend('topleft',legend=paste('n=',n),bty='n')
}
```

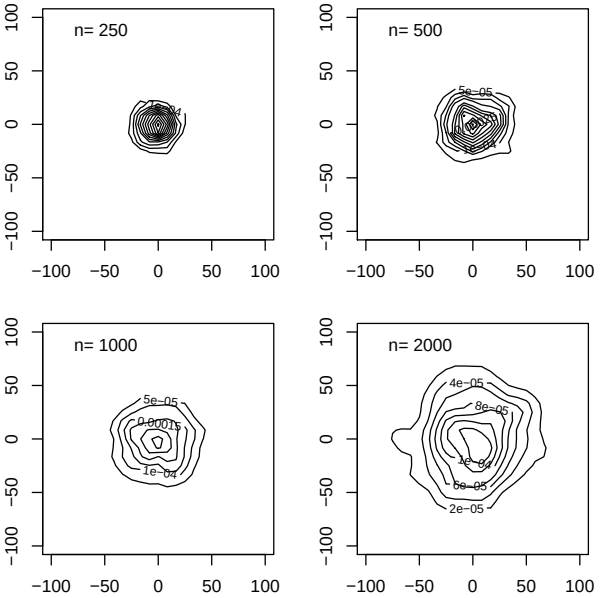



```r
for (n in ns){
   xy <- cloud(n=n,plot=FALSE)
   d <- ecdf(xy[,1])
   plot(d,xlim=c(-100,100),main='',verticals=TRUE,pch=NA)
   legend('topleft',legend=paste('n=',n),bty='n')
}
```



3. Visualise the bivariate datasets as 2-dimensional KDEs.

```r
par(mfrow=c(2,2),mar=rep(2,4))
lims <- c(-100,100)
for (n in ns){
   xy <- cloud(n=n,plot=FALSE)
   d2d <- MASS::kde2d(x=xy[,1],y=xy[,2],lims=rep(lims,2))
   contour(d2d,xlim=lims,ylim=lims)
   legend('topleft',legend=paste('n=',n),bty='n')
}
```


n= 250
n= 500
n= 1000
n= 2000
5e−05
0.00015
1e−04
4e−05
8e−05
2e−05
−100
−50
0
50
100

3 Summary statistics

Using the code from the previous exercises:

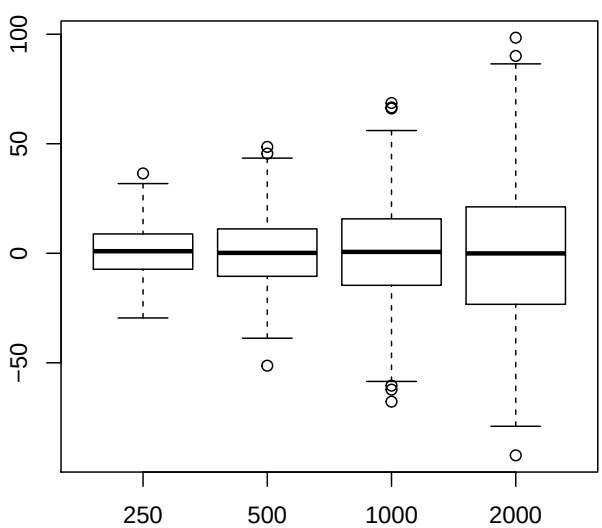
1. Construct a table with the means, medians, standard deviations and interquartile ranges of the synthetic datasets (x-values) of exercise 2.

```
nn <- length(ns)
tab <- data.frame(means=rep(0,nn),medians=rep(0,nn),sds=rep(0,nn),iqrs=rep(0,nn))
for (i in 1:nn){
   x <- cloud(n=ns[i],plot=FALSE)[,1]
   tab[i,'means'] <- mean(x)
   tab[i,'medians'] <- median(x)
   tab[i,'sds'] <- sd(x)
   tab[i,'iqrs'] <- IQR(x)
}
tab
```

```
##        means    medians      sds     iqrs
## 1 -0.7545556 -1.2704257 11.38422 16.22251
## 2 -0.8692892 -1.2742507 16.42722 21.73105
## 3  0.6363320  0.8041491 21.69073 29.26165
## 4  2.2655974  3.0979016 31.18737 43.14988
```

2. Plot the four datasets as box plots.

```
dat <- list()
for (i in 1:nn){
   dat[[i]] <- cloud(n=ns[i],plot=FALSE)[,1]
}
names(dat) <- ns
boxplot(dat)
```



4 Probability

Suppose that a password needs to contain exactly 8 characters and must include at least one lowercase letter, uppercase letter and digit. How many such passwords are possible?

Let $|x|$ stand for 'the number of outcomes for x'. Then

$$|\text{good passwords}| = |\text{all passwords}| - |\text{bad passwords}|$$

where

$$|\text{all passwords}| = (26 + 26 + 10)^8$$

The number of bad passwords requires the *inclusion-exclusion principle*, which is similar to the additive rule of probability (Equation 4.4 of the notes). Let u, l and d represent outcomes *lacking* uppercase letters, lowercase letters and digits, respectively. Then we are looking for all possible combinations of those three outcomes:

$$\begin{aligned}
|\text{bad passwords}| &= |u \cup l \cup d| \\
&= |u \cup l| + |d| - |u \cup l \cap d| \\
&= |u| + |l| + |d| - |u \cap l| - |(u \cup l) \cap d| \\
&= |u| + |l| + |d| - |u \cap l| - |u \cap d| - |l \cap d| + |u \cap l \cap d|
\end{aligned}$$

where $|u \cap l \cap d| = 0$ so that

$$|\text{good passwords}| = (26 + 26 + 10)^8 - (26 + 26)^8 - (26 + 10)^8 - (26 + 10)^8 + 26^8 + 26^8 + 10^8$$

In `R`:

```r
len <- 8 # length of the password
nupper <- 26 # A -> Z
nlower <- 26 # a -> z
ndigit <- 10 # 0 -> 9
# total number of 8-character strings:
N <- (nupper+nlower+ndigit)^len
# remove all the passwords with no uppercase letter, lowercase letter, or digit:
N <- N - (nlower+ndigit)^len - (nupper+ndigit)^len - (nlower+nupper)^len
# But then you removed some passwords twice. You must add back all passwords with
# no lowercase AND no uppercase, no lowercase AND no digit, or no uppercase AND no digit:
N <- N + ndigit^len + nupper^len + nlower^len
# hence the total number of passwords is
print(N)
```

```
## [1] 1.596559e+14
```

5 Bernoulli variables

1. Draw a random number from a uniform distribution between 0 and 1 and store this number in a variable (p, say).

```
set.seed(2) # to get reproducible results (comment out for truly random values)
p <- runif(1)
print(p)
```

```
## [1] 0.1848823
```

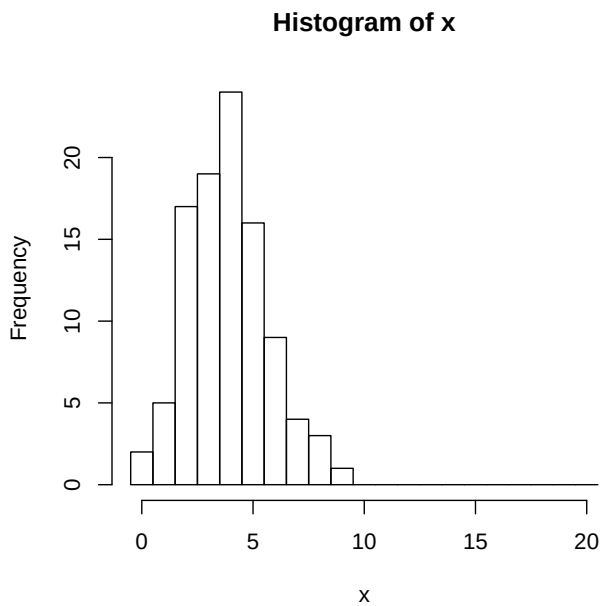
2. draw $n = 20$ additional numbers from the same uniform distribution and count how many of these values are less than p.

```
n <- 20
r <- runif(n)
print(sum(r<p))
```

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

3. Repeat steps 1 and 2 $N = 100$ times, store the results in a vector (x, say), and plot the results as a histogram.

```
N <- 100
m <- matrix(runif(n*N),nrow=N,ncol=n)
x <- rowSums(m<p)
h <- hist(x,breaks=seq(from=-0.5,to=20.5,by=1))
```



4. Suppose that you did not know the value of p, then you could estimate it (as a new variable $\hat{p}$, say) from the data x. To this end, compute the binomial density (or 'likelihood') of all values in x for $\hat{p} = 0.5$ and sample size n.

```
phat <- 0.5 # example value
d <- dbinom(x=x, size=n, prob=phat)
```

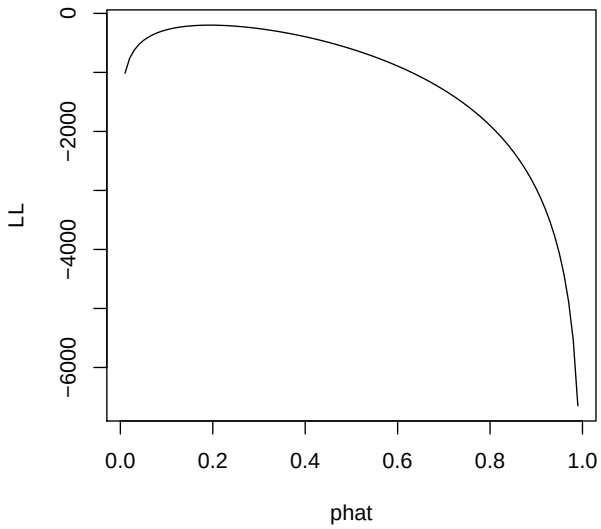
5. Now take the sum of the logarithms of the binomial likelihoods of x. Call this sum LL (for 'log-likelihood').

```r
d <- dbinom(x=x, size=n, prob=phat, log=TRUE)
LL <- sum(d)
print(LL)
```

```
## [1] -602.9585
```

6. Repeat step 5 for a regularly spaced sequence of $\hat{p}$-values. Then plot the resulting LL-values against those $\hat{p}$-values.

```r
phat <- seq(from=0.01,to=0.99,by=0.01) # values of 0 and 1 don't work
np <- length(phat)
LL <- rep(0,np)
for (i in 1:np){
d <- dbinom(x=x, size=n, prob=phat[i], log=TRUE)
LL[i] <- sum(d)
}
plot(phat,LL,type='l')
```



7. Approximately which value of $\hat{p}$ corresponds to the maximum value for LL? How does this compare to the true value of p?

```r
i <- which.max(LL)
message('phat=',phat[i],', p=',signif(p,2))
```

```
## phat=0.19, p=0.18
```

By completing this exercise, you have numerically extended the procedure described in Section 5.1 of the notes.

6 Type-2 errors

1. Draw a random number from a binomial distribution with $n = 20$ and $p = 0.52$.

```
n <- 20
p <- 0.52
r <- rbinom(n=1,size=n,prob=p)
```

2. Apply a binomial test comparing $H_\circ : p = 0.5$ with $H_a : p \neq 0.5$. Do the data pass the test?

```
alpha <- 0.05
p0 <- 0.5
h <- binom.test(x=r,n=n,p=p0)
message('The data ',ifelse(h$p.value<alpha,'fail','pass'),' the test')
```

```
## The data pass the test
```

3. Repeat steps 1 and 2 $N = 1000$ times. What percentage of the datasets pass the test?

```
N <- 1000
r <- rbinom(n=N,size=n,prob=p)
npass <- 0
for (i in 1:N){
   h <- binom.test(x=r[i],n=n,p=p0)
   if (h$p.value>alpha) npass <- npass + 1
}
prob <- npass/N
message(signif(100*prob,2),'%')
```

```
## 97%
```

4. Repeat steps 1 through 3 for $n = 200$.

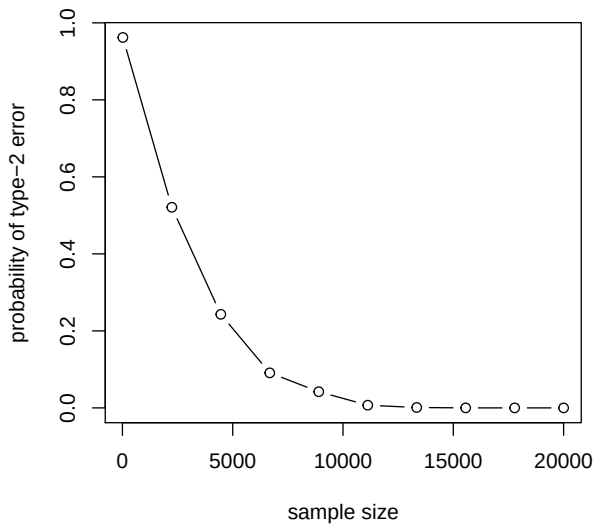
```
# write a function to avoid duplicate code:
btest <- function(n=20,N=1000,p=0.52,p0=0.5,alpha=0.05){
   r <- rbinom(n=N,size=n,prob=p)
   npass <- 0
   for (i in 1:N){
      h <- binom.test(x=r[i],n=n,p=p0)
      if (h$p.value>alpha) npass <- npass + 1
   }
   return(npass/N)
}

# use the function
prob <- btest(n=200,p=p)
message(signif(100*prob,2),'%')
```

```
## 92%
```

5. Repeat step 4 for a range of values between $n = 20$ and $n = 2000$. Plot the probability of rejection against n.

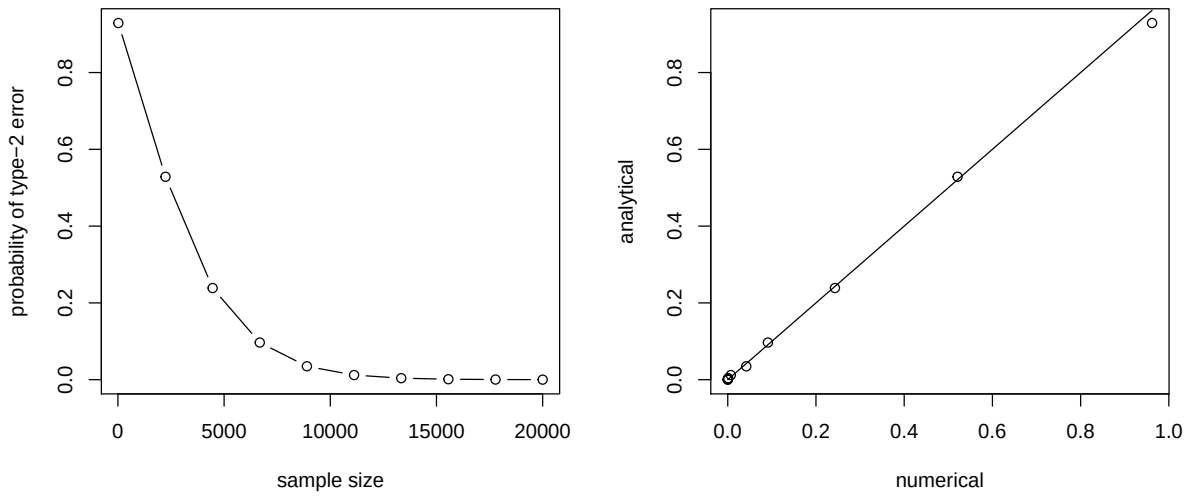
```
nt <- 10 # number of tests
nn <- seq(from=20,to=20000,length.out=nt)
pp <- rep(0,nt)
for (i in 1:nt){
   pp[i] <- btest(n=nn[i],p=p)
}
```

```r
plot(x=nn,y=pp,type='b',xlab='sample size',ylab='probability of type-2 error')
```



6. Compare the results of step 5 with an analytical calculation of the probability of committing a Type-2 error as a function of sample size.

```r
type2 <- function(n=20,p=0.52,p0=0.5){
   # rejection region of a two-sided binomial test for p=0.5
   ll <- qbinom(p=0.025,size=n,prob=p0) # lower limit
   ul <- qbinom(p=0.975,size=n,prob=p0) # upper limit
   # compute the probabilities of erroneously accepted values
   pu <- pbinom(q=ul,size=n,prob=p)
   pl <- pbinom(q=ll,size=n,prob=p)
   return(pu-pl)
}

pp2 <- rep(0,nt)
for (i in 1:nt){
   pp2[i] <- type2(n=nn[i],p=p)
}
par(mfrow=c(1,2))
plot(x=nn,y=pp2,type='b',xlab='sample size',ylab='probability of type-2 error')
plot(x=pp,y=pp2,xlab='numerical',ylab='analytical')
lines(range(pp),range(pp))
```


probability of type−2 error
0.0
0.2
0.4
0.6
0.8
0
5000
10000
15000
20000
sample size
analytical
0.0
0.2
0.4
0.6
0.8
1.0
numerical

7 Confidence intervals

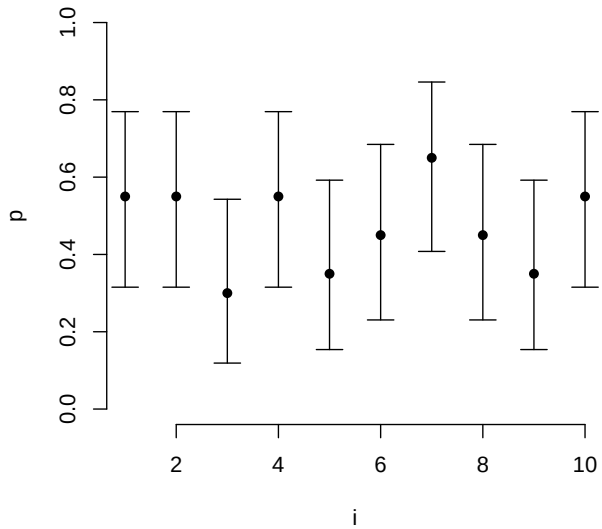
1. Draw $N = 10$ random numbers from a binomial distribution with $p = 0.55$ and $n = 20$. Construct 95% confidence intervals for p for each of these numbers.

```r
# define a function to avoid future duplication of code:
bci <- function(n=20,N=10,p=0.5){
   out <- matrix(0,nrow=3,ncol=N)
   rownames(out) <- c('p','ll','ul')
   k <- rbinom(n=N,size=n,prob=p)
   out['p',] <- k/n
   for (i in 1:N){
      h <- binom.test(k[i],n=n)
      out[c('ll','ul'),i] <- h$conf.int
   }
   return(out)
}

# use the new function:
ci <- bci(p=0.55)
print(ci)
```

```
##         [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## p  0.5500000 0.5500000 0.3000000 0.5500000 0.3500000 0.4500000 0.6500000
## ll 0.3152781 0.3152781 0.1189316 0.3152781 0.1539092 0.2305779 0.4078115
## ul 0.7694221 0.7694221 0.5427892 0.7694221 0.5921885 0.6847219 0.8460908
##         [,8]      [,9]     [,10]
## p  0.4500000 0.3500000 0.5500000
## ll 0.2305779 0.1539092 0.3152781
## ul 0.6847219 0.5921885 0.7694221
```

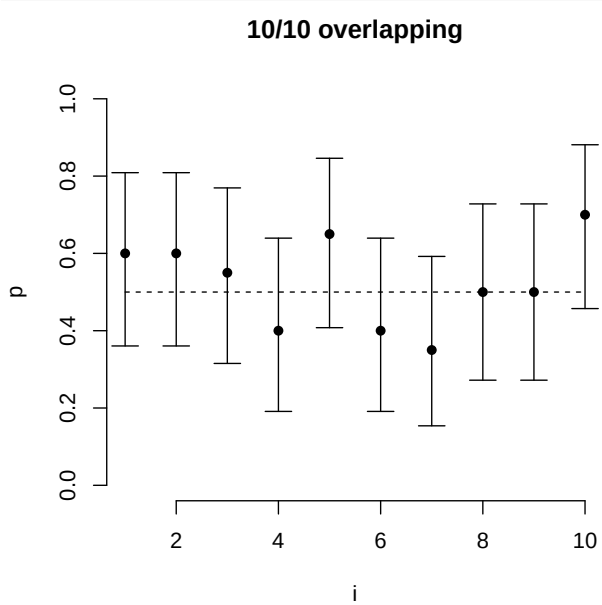
2. Plot these confidence intervals against n as error bars using `R`'s `arrows()` function.

```r
plotci <- function(ci,...){
   N <- ncol(ci)
   plot(1:N,ci['p',],pch=16,ylim=c(0,1),bty='n',xlab='i',ylab='p',...)
   arrows(x0=1:N,y0=ci['ll',],y1=ci['ul',],length=0.1,angle=90,code=3)
}
set.seed(1) # set the random seed for the sake of reproducibility
plotci(ci)
```



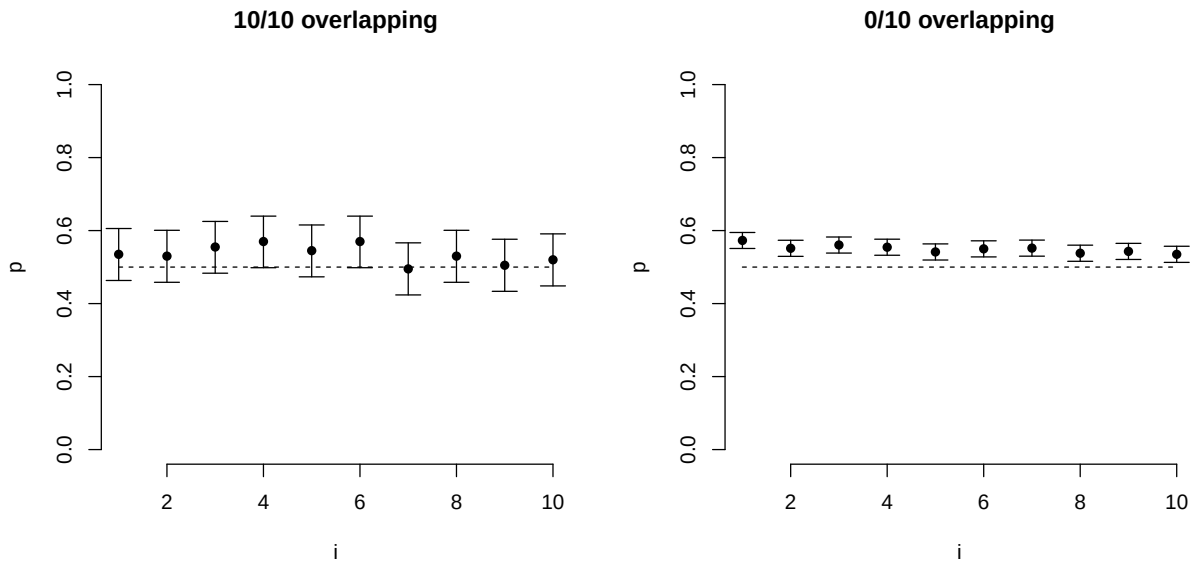
3. How many of the confidence intervals in step 2 overlap with $p_0 = 0.5$?

```
# write function to avoid future duplication:
bciexplorer <- function(n=20,N=10,p=0.5,p0=0.5){
   ci <- bci(n=n,N=N,p=p)
   overlap <- sum(p0>ci['ll',] & p0<ci['ul',])
   tit <- paste0(overlap,'/',N,' overlapping') # title
   plotci(ci,main=tit)
   lines(x=1:N,y=rep(p0,N),lty=2)
}

# use the function:
set.seed(1)
bciexplorer(n=20,N=10,p=0.55,p0=0.5)
```



4. Repeat step 2 for $n = 200$. How many of these new confidence intervals fall outside $p = 0.5$? Do so again for $n = 2000$.

```r
par(mfrow=c(1,2))
bciexplorer(n=200,N=10,p=0.55,p0=0.5)
bciexplorer(n=2000,N=10,p=0.55,p0=0.5)
```



8 Radioactive decay

Radioactive decay is the process whereby a *parent* isotope P transforms to a *daughter* isotope D at a rate that is controlled by the amount of parent and a *decay constant* λ_P:

$$\partial D/\partial t = -\partial P/\partial t = \lambda_P P$$

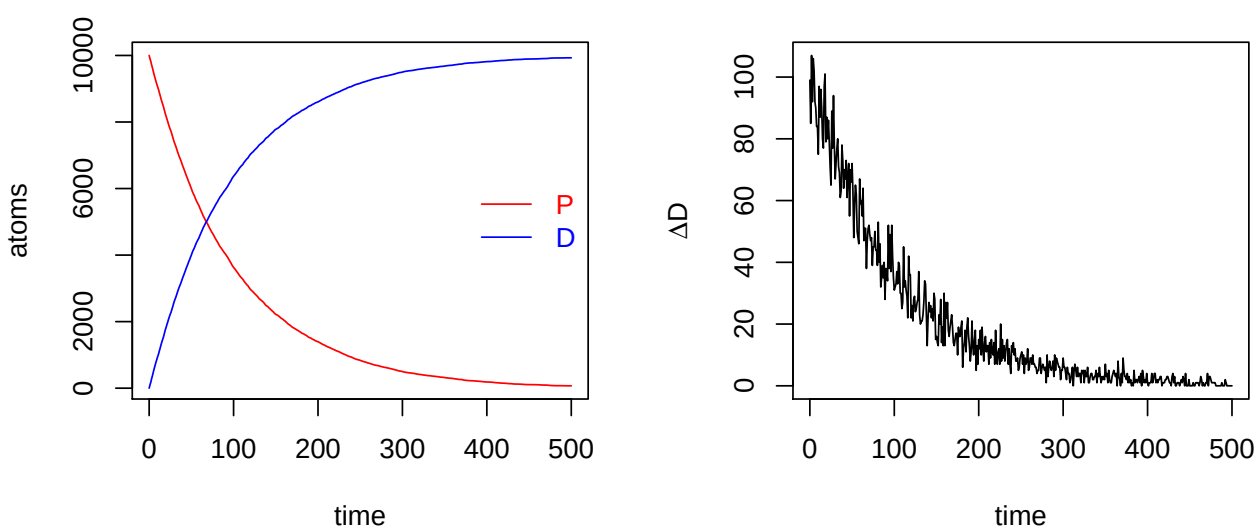
The rate of decay ($\partial D/\partial t$) can be measured with a Geyger counter. In this exercise, you will simulate radioactive decay using a finite differences approximation to the decay equation:

$$\Delta D/\Delta t = -\Delta P/\Delta t \approx \lambda_P P$$

Then the expected number of parent atoms that are lost (and daughter atoms that are gained) per time interval Δt follows a Poisson distribution with parameter $\lambda_P P \Delta t$.

Suppose that you start off with $P = 10000$ atoms of a parent isotope with decay constant $\lambda_P = 1/100$. Then simulate the evolution of P, D and ΔD from as a function of discretised time from $t = 0 \rightarrow 500$ with $\Delta t = 1$.

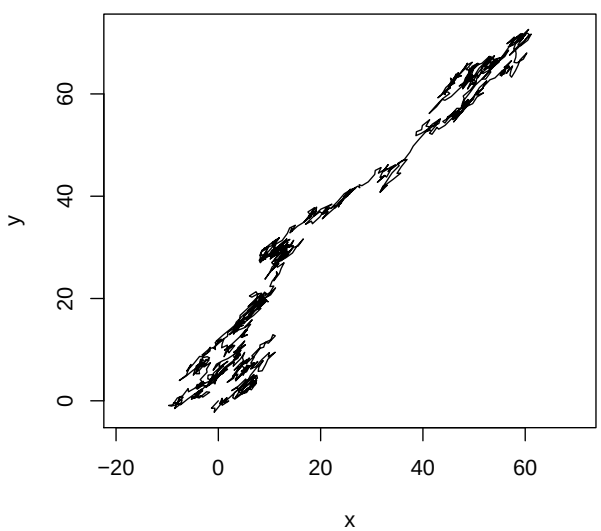
```
dt <- 1                                # time step
lamp <- 1/100                          # decay constant
tt <- seq(from=0,to=500,by=dt)         # time vector
nt <- length(tt)                       # number of steps
P <- rep(10000,nt)                     # parent atoms
D <- rep(0,nt)                         # daughter atoms
dD <- rep(0,nt)                        # lost daughter atoms per step
for (i in 1:(nt-1)){
   dD[i] <- rpois(n=1,lambda=lamp*P[i]*dt)
   D[i+1] <- D[i] + dD[i]
   P[i+1] <- P[i] - dD[i]
}
dD[nt] <- rpois(n=1,lambda=lamp*P[nt]*dt) # final step
par(mfrow=c(1,2))
cols <- c('red','blue')
plot(x=tt,y=P,type='l',col=cols[1],xlab='time',ylab='atoms')
lines(x=tt,y=D,col=cols[2])
legend('right',legend=c('P','D'),lty=1,col=cols,text.col=cols,bty='n')
plot(tt,dD,type='l',xlab='time',ylab=expression(Delta*'D'))
```



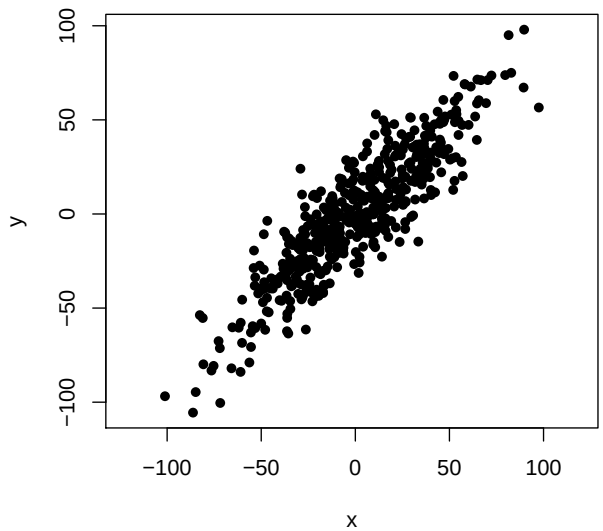
9 The normal distribution

1. Modify the random walk code of exercise 1 so that the random displacements are not defined by a unit circle but by an ellipse with major axis $a = 2$, minor axis $b = 0.5$ and rotation angle $\alpha = \pi/4$. See exercise 18.1.2 of the notes for the relevant equations.

```
ell <- function(n=1,a=1,b=1,alpha=0){
   beta <- runif(n,min=0,max=2*pi) # random directions
   dx <- a*cos(alpha)*cos(beta) - b*sin(alpha)*sin(beta)
   dy <- a*sin(alpha)*cos(beta) + b*cos(alpha)*sin(beta)
   cbind(dx,dy)
}

# random walk for one particle:
walk <- function(xy0=rep(0,2),n=1000,a=1,b=1,alpha=0){
   xy <- matrix(rep(xy0,n),ncol=2,byrow=TRUE)
   dxy <- ell(n=n,a=a,b=b,alpha=alpha)
   for (i in 2:n){
      xy[i,1] <- xy[i-1,1] + dxy[i,1]
      xy[i,2] <- xy[i-1,2] + dxy[i,2]
   }
   return(xy)
}
xy <- walk(a=2,b=0.5,alpha=pi/4)
plot(xy,type='l',xlab='x',ylab='y',asp=1)
```

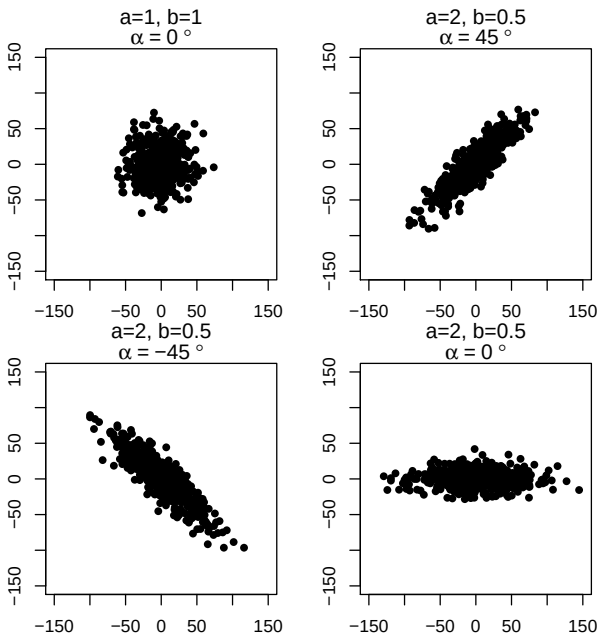



```
n <- 1000
N <- 500
xyf <- matrix(NA,nrow=N,ncol=2)
for (i in 1:N){
   xyf[i,] <- walk(n=n,a=2,b=0.5,alpha=pi/4)[n,]
}
plot(xyf,type='p',pch=16,xlab='x',ylab='y',asp=1)
```



2. Explore the effects of different values for a, b and α. No-one less than *Albert Einstein* showed that random walks (which are also known as 'Brownian motion') lead to diffusion, which gives rise to bivariate normal distributions. In exercise 2, this diffusion was *isotropic*. The elliptical modification is one way to simulate *anisotropic* diffusion.

```
cloud <- function(n=1000,N=500,plot=TRUE,a=1,b=1,alpha=0,...){
   xy <- matrix(NA,nrow=N,ncol=2)
   for (i in 1:N){
      xy[i,] <- walk(n=n,a=a,b=b,alpha=alpha)[n,]
   }
   # the optional plot argument will be used in exercise 11 below
   if (plot) plot(xy,pch=16,xlab='x',ylab='y',...)
   invisible(xy)
}

par(mfrow=c(2,2),mar=rep(2,4))
lims <- c(-150,150)
a <- c(1,2  ,2  ,2)
b <- c(1,0.5,0.5,0.5)
alpha <- c(0,pi/4,-pi/4,0)
n <- 1000
for (i in 1:4){
   cloud(n=n,xlim=lims,ylim=lims,a=a[i],b=b[i],alpha=alpha[i],asp=1)
   mtext(text=paste0('a=',a[i],', b=',b[i]),line=1)
   adeg <- signif(alpha[i]*180/pi,2)
   mtext(text=substitute(alpha~'='~a~degree,list(a=adeg)),line=0)
}
```


a=1, b=1
α = 0 °
a=2, b=0.5
α = 45 °
a=2, b=0.5
α = −45 °
a=2, b=0.5
α = 0 °
150
50
0
−50
−150
−150
−50
0
50
150

10 Error propagation

Consider a bivariate normal distribution with the following mean vector and covariance matrix:

$$\mu = \begin{bmatrix} x = -2 \\ y = 3 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & -2 \\ -2 & 5 \end{bmatrix}$$

1. Predict $z = x + 2y$ and estimate its standard error.

```
mu <- c(-2,3)
Sigma <- matrix(c(1,-2,-2,5),nrow=2,ncol=2)
x <- mu[1]
y <- mu[2]
sx <- sqrt(Sigma[1,1])
sy <- sqrt(Sigma[2,2])
sxy <- Sigma[1,2]
z <- x + 2*y
# using equation 8.10:
sz <- sqrt( sx^2 + (2*sy)^2 + 2*2*sxy )
message('z=',signif(z,2),', s[z]=',signif(sz,2))
```

```
## z=4, s[z]=3.6
```

2. Draw $n = 1000$ pairs of random numbers from the bivariate normal distribution. Compute z for each pair and calculate the mean and standard error of the resulting vector. How does it compare with your analytical solution?

```
# instead of loading the entire MASS package,
# you can also call one of its functions like this:
xy <- MASS::mvrnorm(n=1000,mu=mu,Sigma=Sigma)
Z <- xy[,1] + 2*xy[,2]
sZ <- sd(Z)
message('z=',signif(mean(Z),2),', s[z]=',signif(sZ,2))
```

```
## z=4, s[z]=3.6
```

3. Repeat steps 1 and 2 for $z = x^2y^3$.

```
# step 1:
z <- (x^2)*(y^3)
dzdx <- 2*x*y^3
dzdy <- 3*(x^2)*(y^2)
# using equation 8.8:
sz <- sqrt( (dzdx*sx)^2 + (dzdy*sy)^2 + 2*dzdx*dzdy*sxy )
message('z=',signif(z,2),', s[z]=',signif(sz,2))
```

```
## z=110, s[z]=340
```

```
# step 2:
xy <- MASS::mvrnorm(n=1000,mu=mu,Sigma=Sigma)
Z <- (xy[,1]^2)*(xy[,2]^3)
sZ <- sd(Z)
message('z=',signif(mean(Z),2),', s[z]=',signif(sZ,2))
```

```
## z=730, s[z]=1600
```

4. Repeat step 4 for

$$\Sigma = \begin{bmatrix} 0.01 & -0.02 \\ -0.02 & 0.05 \end{bmatrix}$$

```r
Sigma <- matrix(c(.01,-.02,-.02,.05),nrow=2,ncol=2)
sx <- sqrt(Sigma[1,1])
sy <- sqrt(Sigma[2,2])
sxy <- Sigma[1,2]
# step 1:
z <- (x^2)*(y^3)
dzdx <- 2*x*y^3
dzdy <- 3*(x^2)*(y^2)
# using equation 8.8:
sz <- sqrt( (dzdx*sx)^2 + (dzdy*sy)^2 + 2*dzdx*dzdy*sxy )
message('z=',signif(z,2),', s[z]=',signif(sz,2))
```

```
## z=110, s[z]=34
```

```r
# step 2:
xy <- MASS::mvrnorm(n=1000,mu=mu,Sigma=Sigma)
Z <- (xy[,1]^2)*(xy[,2]^3)
sZ <- sd(Z)
message('z=',signif(mean(Z),2),', s[z]=',signif(sZ,2))
```

```
## z=110, s[z]=35
```

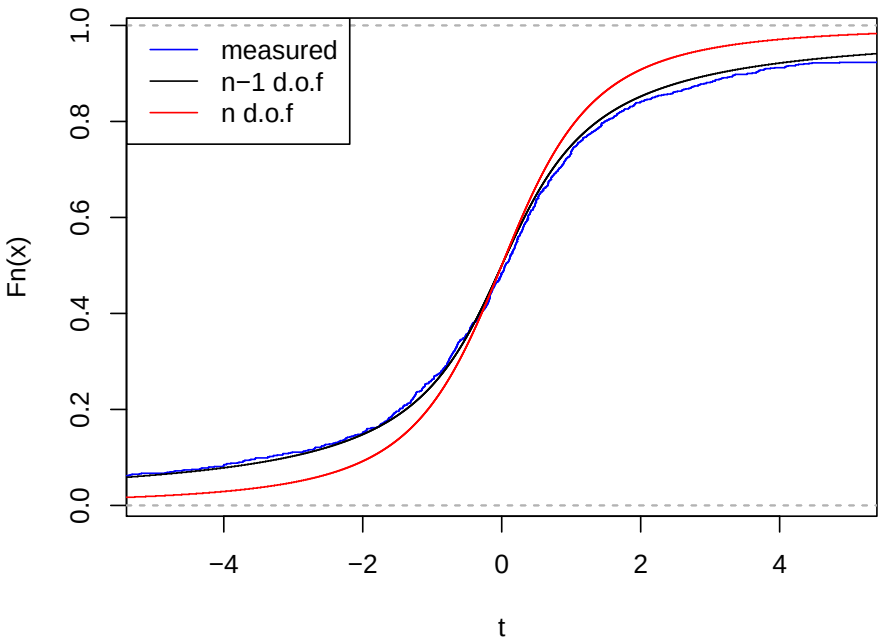
Conclusion: the error propagation formulas work well for linear functions but not for strongly non-linear ones, unless the measurements are precise.

11 Degrees of freedom

The degrees of freedom of a problem represents the number of independent ways in which a system can vary. In most cases, the degrees of freedom represents the number of measurements, adjusted for the number of parameters. In the context of the t-test, we use n measurements to estimate the one-sample t-statistic t. But to do so we also have to calculate $\bar{x}$. So there is some redundancy in the system, which is reducing the apparent dispersion of the t-statistic. To account for this, we subtract one degree of freedom, in exactly the same way as the Bessel correction, which is briefly discussed at the end of Chapter 7 of the notes. See Wikipedia for a derivation of the Bessel correction. It is a useful exercise to simulate the t-distribution on your computer:

1. Draw $N = 1000$ times $n = 2$ random numbers from a standard normal distribution and calculate their means.
2. Compute the t-statistics of the 1000 resulting values, using Equation 9.2 of the notes. Visualise as an ECDF.
3. Superimpose the CDF of the t-distribution with $n - 1$ degrees of freedom on the existing plot, and then again with n degrees of freedom. Which curve fits the simulated results best?

```
ns <- 1000      # number of samples
nv <- 2         # number of values per sample
# 1000 samples of 2 values drawn from a standard normal distribution:
obs <- matrix(rnorm(nv*ns),nrow=ns,ncol=nv)
tstat <- rep(NA,ns)  # initialise the t-statistic
for (i in 1:ns){     # loop through the samples
   tstat[i] <- sqrt(nv)*mean(obs[i,])/sd(obs[i,])  # equation 9.2 of the notes
}
# predicted quantiles of the t-distribution with nv-1 degrees of freedom:
pred1 <- qt(seq(from=0,to=1,length.out=ns),df=nv-1)
# predicted quantiles of the t-distribution with nv degrees of freedom:
pred2 <- qt(seq(from=0,to=1,length.out=ns),df=nv)
# plot the empirical cumulative distribution function of the 1000 t-statistics:
plot(ecdf(tstat),verticals=TRUE,pch=NA,col='blue',xlim=c(-5,5),xlab='t',main='')
# add the predicted distribution:
lines(ecdf(pred1),verticals=TRUE,pch=NA,col='black')
# add the second predicted distribution:
lines(ecdf(pred2),verticals=TRUE,pch=NA,col='red')
legend('topleft',legend=c('measured','n-1 d.o.f','n d.o.f'),
       lty=1,col=c('blue','black','red'))
```

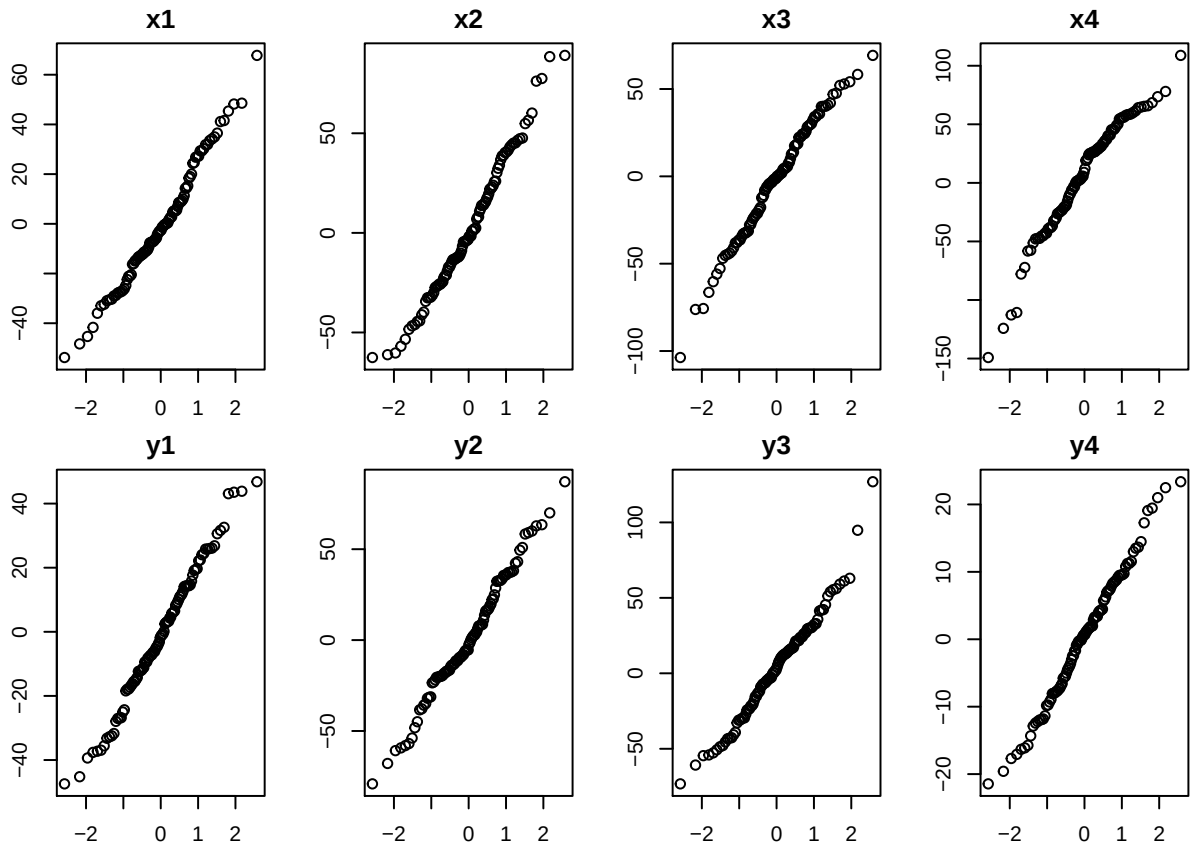


In the two-sample t-test (Equation 9.4), we have $n_1 + n_2$ measurements, and have to estimate two parameters, $\bar{x}_1$ and $\bar{x}_2$. Hence the number of degrees of freedom is $n_1 + n_2 - 2$. If you want you can modify this code for the two-sample case to verify that this requires $n_1 + n_2 - 2$ degrees of freedom.

12 Comparing distributions

1. Compare the marginal distributions of exercise 9 with a normal distribution using Q-Q plots.

```
a <- c(1,2 ,2 ,2)
b <- c(1,0.5,0.5,0.5)
alpha <- c(0,pi/4,-pi/4,0)
n <- 1000
N <- 100
nc <- length(a)
par(mfcol=c(2,nc),mar=rep(2,4))
xy <- list() # storing the matrices in a list will save trouble later
for (i in 1:nc){
   xy[[i]] <- cloud(n=n,N=N,a=a[i],b=b[i],alpha=alpha[i],plot=FALSE)
   qqnorm(xy[[i]][,1],main=paste0('x',i))
   qqnorm(xy[[i]][,2],main=paste0('y',i))
}
```



2. Formalise the comparisons using a Kolmogorov-Smirnov test. See the documentation of the `ks.test()` function for help. Note: you should *normalise* the data by specifying the mean and standard deviation of the marginal distributions to the `ks.test()` function. See the documentation for the ellipsis (`...`) in said documentation.

```
pvals <- matrix(0,nrow=2,ncol=nc)
rownames(pvals) <- c('x','y')
for (i in 1:nc){
   x <- xy[[i]][,1]
   y <- xy[[i]][,2]
```

```r
   ksx <- ks.test(x=x,y='pnorm',mean=mean(x),sd=sd(x))
   ksy <- ks.test(x=y,y='pnorm',mean=mean(y),sd=sd(y))
   pvals['x',i] <- ksx$p.value
   pvals['y',i] <- ksy$p.value
}
signif(pvals,2)
```

```
##   [,1] [,2] [,3] [,4]
## x 0.71 0.64 0.81 0.19
## y 1.00 0.57 0.91 0.99
```

Conclusion: the marginal distributions are Gaussian.

13 Effect size

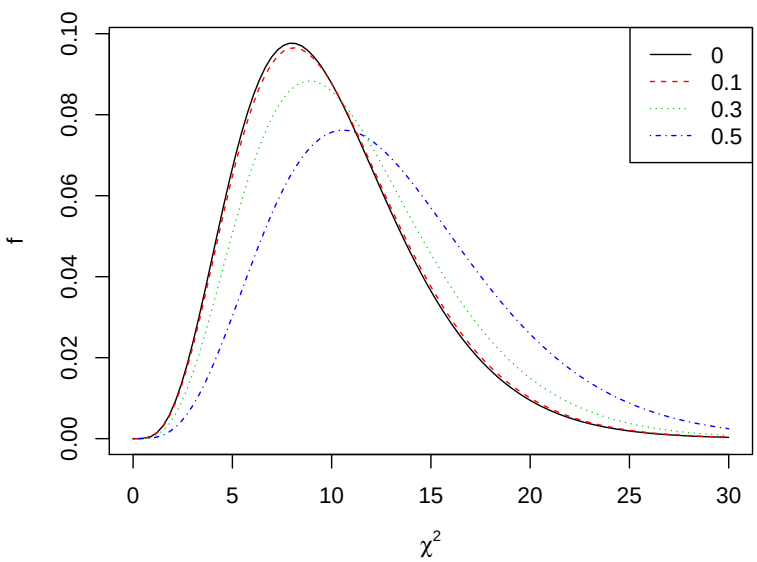
The optional arguments of R's `rchisq`, `dchisq`, `pchisq` and `qchisq` functions include a 'noncentrality parameter' (`ncp`), which is related to the effect size (w) as follows:

$\texttt{ncp} = n \times w^2$

where n is the sample size. By default, the noncentrality parameter is zero, which is generally used as a null distribution. Changing `ncp` to a positive number allows the user to explore different alternative distributions.

1. Plot the PDFs of (non-central) chi-square distributions with 10 degrees of freedom (i.e. $n = 12$) and effect sizes of 0, 0.1, 0.3 and 0.5.

```
# (a)
n <- 12
w <- c(0,0.1,0.3,0.5)
nw <- length(w)
nn <- 100
x <- seq(from=0,to=30,length.out=nn)
y <- matrix(0,nn,nw)
for (i in 1:nw){
y[,i] <- dchisq(x,df=n-2,ncp=n*w[i]^2)
}
matplot(x,y,type='l',xlab=expression(chi^2),ylab='f')
legend('topright',legend=w,lty=1:nw,col=1:nw)
```



1. What is the probability of committing a type-II error for $w \in \{0.1, 0.3, 0.5\}$ with $\alpha = 0.05$?
2. What is the probability of committing a type-II error for the same values of w when $n = 1000$?

```
# (b)
type2 <- function(n,w){
# Calculate the 95-precentile of the central chi-square distribution:
cutoff <- qchisq(0.95,df=n-2)
# Calculate the probability of exceeding this value for different values of w:
for (i in 2:length(w)){
prob <- pchisq(cutoff,df=n-1,ncp=n*w[i]^2)
message('p(type-2 error | w=',w[i],') = ',signif(prob,2))
}
```

```r
}
type2(n,w)
```

```
## p(type-2 error | w=0.1) = 0.92
## p(type-2 error | w=0.3) = 0.88
## p(type-2 error | w=0.5) = 0.79
```

```r
# (c)
type2(1000,w)
```

```
## p(type-2 error | w=0.1) = 0.92
## p(type-2 error | w=0.3) = 0.37
## p(type-2 error | w=0.5) = 0.00038
```

14 Regression

Consider the following data table:

x	$s[x]$	y	$s[y]$	$r[x,y]$
3	1	7	1	0.9
7	1	9	1	0.9
9	1	13	1	0.9
12	1	14	1	0.9
14	1	19	1	-0.9

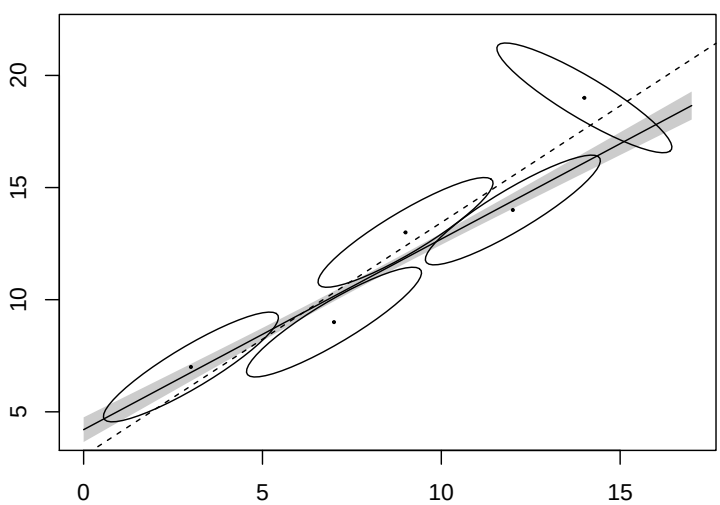
1. Fit a straight line through the x and y values, ignoring the uncertainties ($s[x]$, $s[y]$) and error correlations ($r[x,y]$). Predict a 95% confidence interval for y at $x = 20$.

```
x <- c(3,7,9,12,14)
y <- c(7,9,13,14,19)
lmfit <- lm(y ~ x)
predict(lmfit,newdata=data.frame(x=20),interval='confidence')
```

```
##        fit      lwr      upr
## 1 23.84595 17.22376 30.46813
```

2. Repeat the linear regression taking into account the analytical uncertainties, using the `geostats` package's `york()` function. Predict the y-value at $x = 20$ and estimate its standard error. Calculate the corresponding 95% confidence interval for y using a t-distribution with $n - 2 = 3$ degree of freedom. Note that `york()` does not use formula notation. See `?york` for details.

```
sx <- rep(1,5)
sy <- rep(1,5)
rxy <- c(rep(0.9,4),-0.9)
tab <- cbind(x,sx,y,sy,rxy)
yfit <- geostats::york(tab)
abline(lmfit$coefficients,lty=2)
```




```
xnew <- 20
ynew <- yfit$coef[1] + yfit$coef[2]*xnew
# error propagation formula 8.10 of the notes:
synew <- sqrt( yfit$cov[1,1] + yfit$cov[2,2]*xnew^2 + 2*xnew*yfit$cov[1,2])
df <- length(x)-2
```

```r
ci <- ynew + qt(c(0.025,0.975),df=df)
out <- signif(c(ynew,ci),5)
names(out) <- c('fit','lwr','upr')
out
```

```
##    fit    lwr    upr
## 21.207 18.025 24.389
```

Conclusion: 'York regression' produces more accurate and more precise predictions.

15 Fractals

1. What is the fractal dimension of the following pattern?

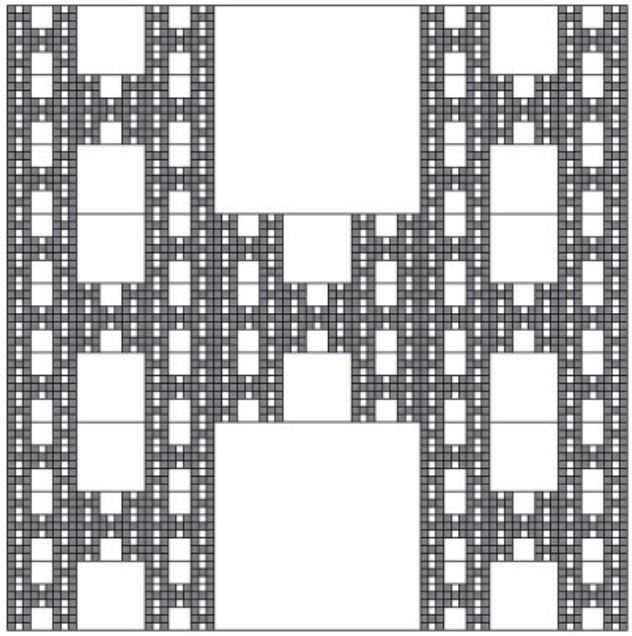


Figure 1: Rivera H-I fractal

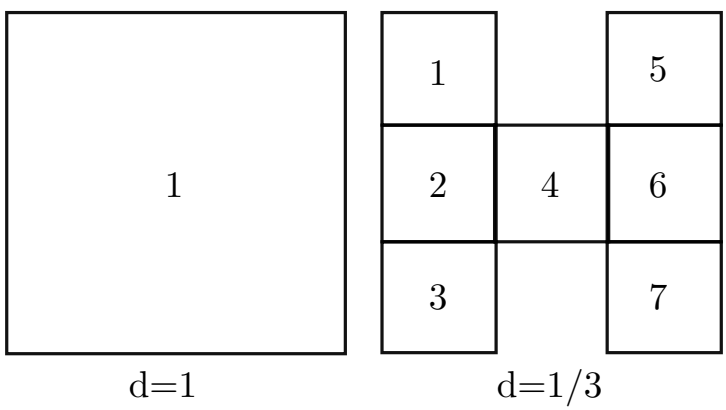
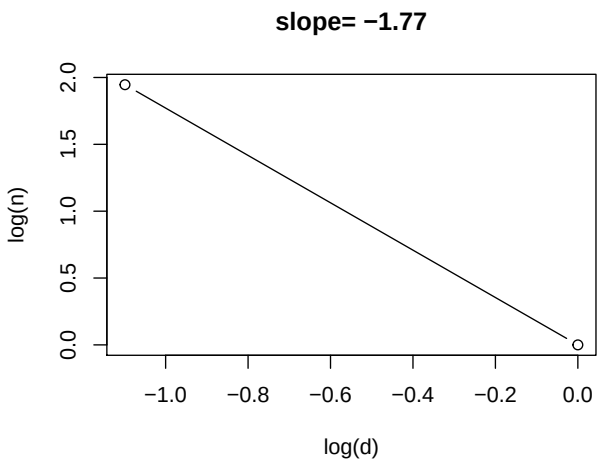


Figure 2: Box counting

1 square block with side $d = 1$ covers the entire pattern, and so do 7 square blocks of side $d = 1/3$. Hence the fractal dimension is

$$f = -\frac{\ln(7) - \ln(1)}{\ln(1/3) - \ln(1)} = \frac{\ln(7)}{\ln(3)} = 1.77$$

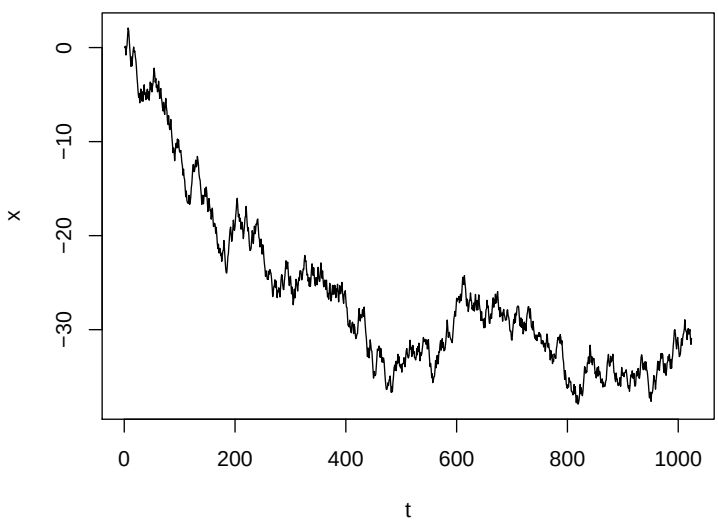
```
d <- c(1,1/3)
n <- c(1,7)
fdim <- (log(n[2])-log(n[1]))/(log(d[2])-log(d[1]))
plot(log(d),log(n),type='b',main=paste('slope=',signif(fdim,3)))
```



2. Using your code from exercise 1:

   (a) Create a random walk of 1024 steps and plot the x-position of a virtual particle against time (where time goes from 0 to 1024).

```
x <- walk(n=1024)[,1]
nx <- length(x)
t <- 1:nx
plot(t,x,type='l')
```

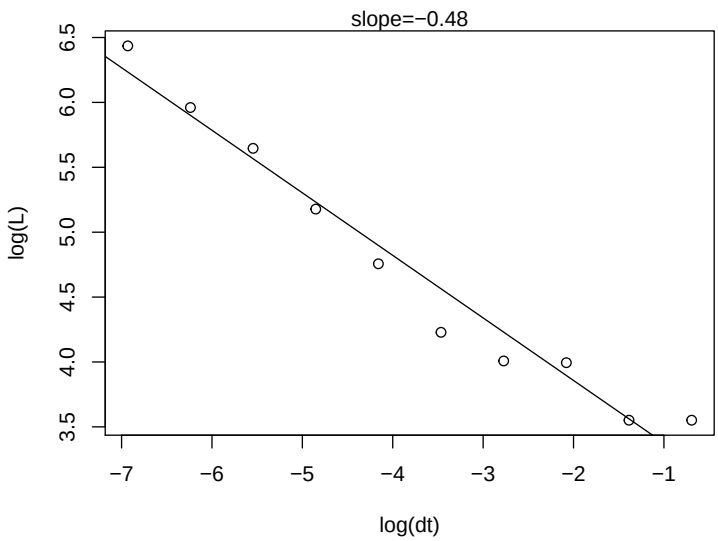


   (b) Subsample the vector of x-positions into N equally spaced segments (for $N = 2, 4, 8, 16, ..., 1024$) and add up their respective lengths.

   (c) Plot the lengths of the curve against the duration of the corresponding time steps, on a log-log plot. What is the slope?

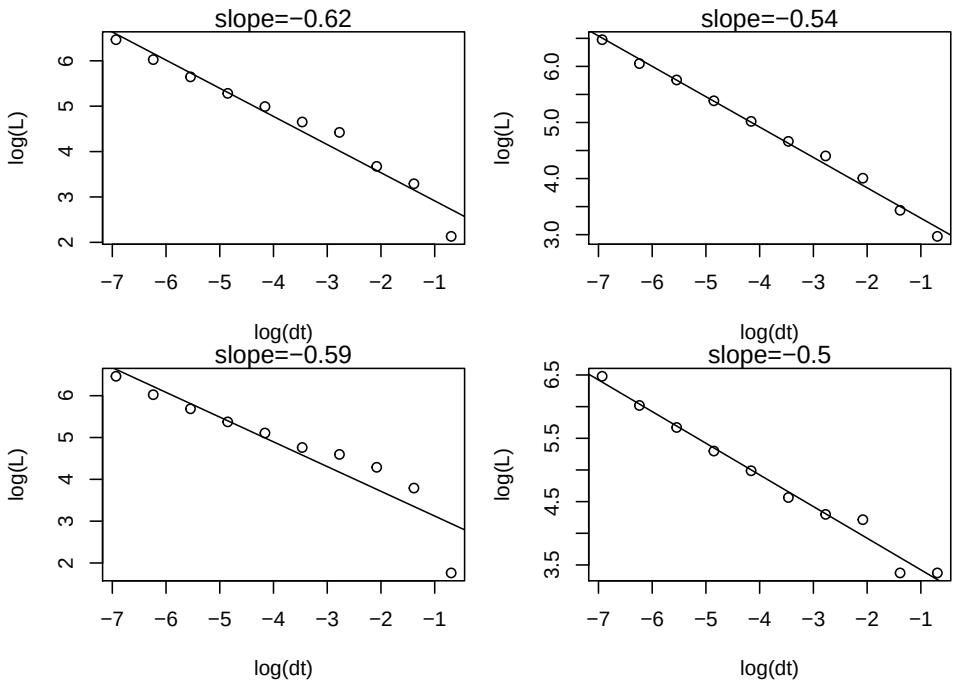
```
walklength <- function(x,N=1){
   nx <- length(x)
   i <- seq(from=1,to=nx,length.out=N+1)
   dx <- abs(diff(x[i]))
   sum(dx)
}
walkdim <- function(x){
   N <- 2^(1:10)
   nN <- length(N)
```

```
   L <- rep(0,nN)
   for (i in 1:nN){
      L[i] <- walklength(x,N=N[i])
   }
   dt <- 1/N
   plot(log(dt),log(L))
   fit <- lm(log(L) ~ log(dt))
   abline(fit)
   slope <- fit$coefficients[2]
   mtext(paste0('slope=',signif(slope,2)))
}
walkdim(x)
```



(d) Repeat steps (a)–(c) several times to assess the robustness of the result.

```
par(mfrow=c(2,2),mar=c(4,4,1,1))
for (i in 1:4){
   x <- walk(n=1024)[,1]
   walkdim(x)
}
```


slope=−0.62
log(L)
log(dt)
slope=−0.54
log(L)
log(dt)
slope=−0.59
log(L)
log(dt)
slope=−0.5
log(L)
log(dt)

16 Unsupervised learning

Consider three multivariate normal distributions with the following mean vectors and covariance matrices:

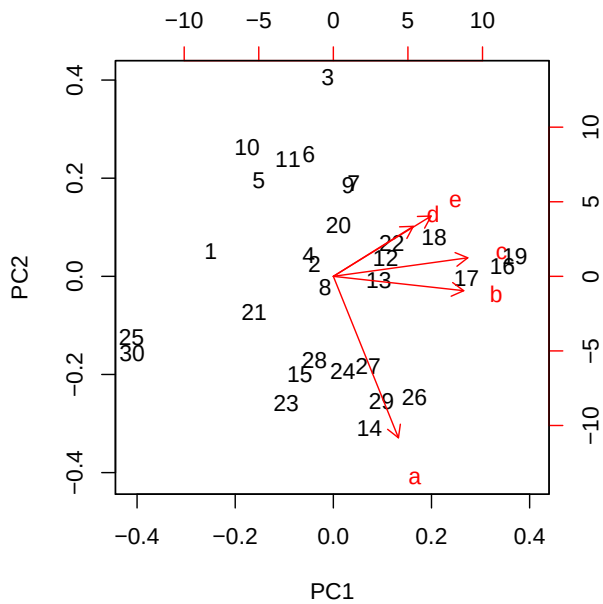
$$\mu_X = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \mu_Y = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \mu_Z = \begin{bmatrix} 4 \\ 0 \\ 0 \\ -2 \\ -2 \end{bmatrix}, \Sigma_X = \Sigma_Y = \Sigma_Z = \begin{bmatrix} 5 & 1 & 2 & 3 & 4 \\ 1 & 5 & 2 & 3 & 4 \\ 2 & 2 & 5 & 3 & 4 \\ 3 & 3 & 3 & 5 & 4 \\ 4 & 4 & 4 & 4 & 5 \end{bmatrix}$$

1. Draw 10 sets of random values from each of these 3 distributions and store them in a matrix. Name the variables a, b, c, d and e.

```
n <- 10
muX <- rep(0,5)
muY <- c(2,2,2,0,0)
muZ <- c(4,0,0,-2,-2)
S <- rbind(c(5,1,2,3,4),
           c(1,5,2,3,4),
           c(1,2,5,3,4),
           c(1,2,3,5,4),
           c(1,2,3,4,5))
library(MASS)
# defining a function because we will use this code again in a later exercise:
XYZ <- function(n,muX,muY,muZ,S){
   X <- mvrnorm(n=n,mu=muX,Sigma=S)
   Y <- mvrnorm(n=n,mu=muY,Sigma=S)
   Z <- mvrnorm(n=n,mu=muZ,Sigma=S)
   out <- rbind(X,Y,Z)
   colnames(out) <- c('a','b','c','d','e')
   return(out)
}
dat <- XYZ(n,muX,muY,muZ,S)
```

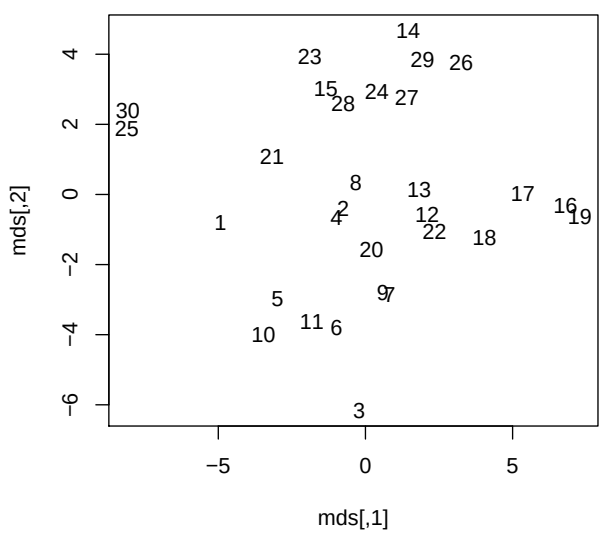
2. Analyse the dataset by principal component analysis and present the results as a biplot.

```
pc <- prcomp(dat)
biplot(pc)
```



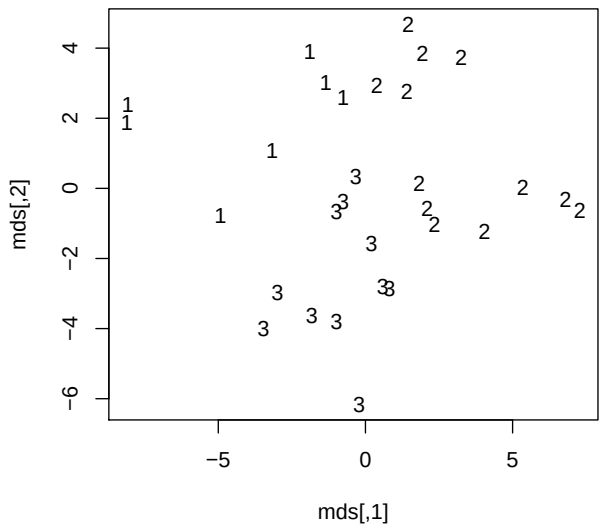
3. Analyse the dataset by classical multidimensional scaling. Plot the results as a scatter plot.

```
mds <- cmdscale(dist(dat))
plot(mds,type='n')
text(mds,labels=1:(3*n))
```



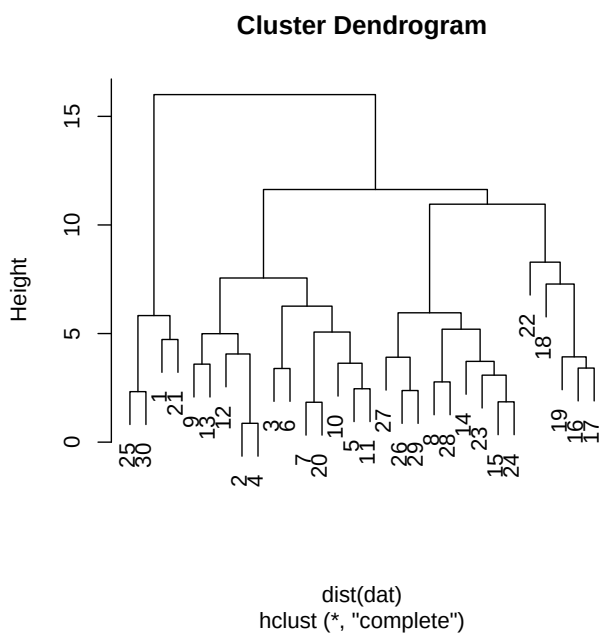
4. Classify the dataset by k-means clustering with $k = 3$. Use the results to add text labels to the MDS configuration. How successful was the clustering?

```
fit <- kmeans(dat,centers=3)
plot(mds,type='n')
text(mds,labels=fit$cluster)
```



5. Create a hierarchical dendrogram for the dataset. Would you have found the three groups?

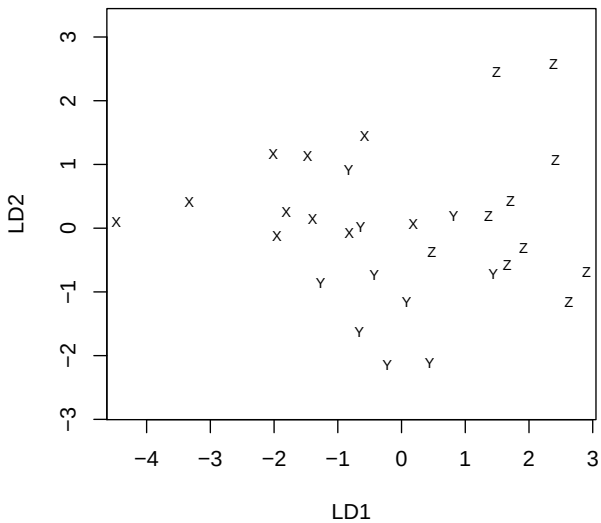
```
tree <- hclust(dist(dat))
plot(tree)
```



17 Supervised learning

1. Create a 30-element vector with the names of the known classes (X, Y, Z) of the 3×10 samples generated in the previous exercise. Build a linear discriminant model for the data given the group names. Plot the first two linear discriminants on a scatter plot.

```
library(MASS)
groups <- c(rep('X',n),rep('Y',n),rep('Z',n))
ldat <- data.frame(groups=groups,dat)
ld <- lda(groups ~ ., data=ldat)
plot(ld)
```



2. What is the misclassification rate of the LDA using the training data? Create a new dataset of 3×10 samples from the multivariate distributions of Exercise 16. Classify the new data using the linear discriminants generated in the previous step. What is the misclassification rate of the test data?

```
table(predict(ld)$class,groups)
```

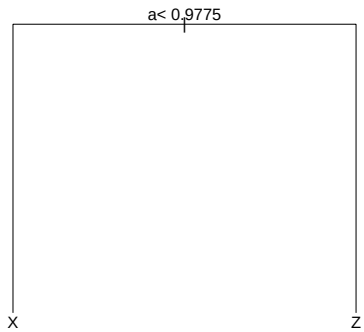
```
##    groups
##     X Y Z
##   X 8 1 0
##   Y 2 7 1
##   Z 0 2 9
```

```
ndat <- data.frame(groups=groups,XYZ(n,muX,muY,muZ,S))
pred <- predict(ld,newdata=ndat)
table(pred$class,groups)
```

```
##    groups
##     X Y Z
##   X 3 2 0
##   Y 7 4 3
##   Z 0 4 7
```

3. Build a decision tree using the same training data as before. How many branches does the default tree have?

```
library(rpart)
tree <- rpart(groups ~ ., data=ldat, method='class')
plot(tree)
text(tree,xpd=NA)
```

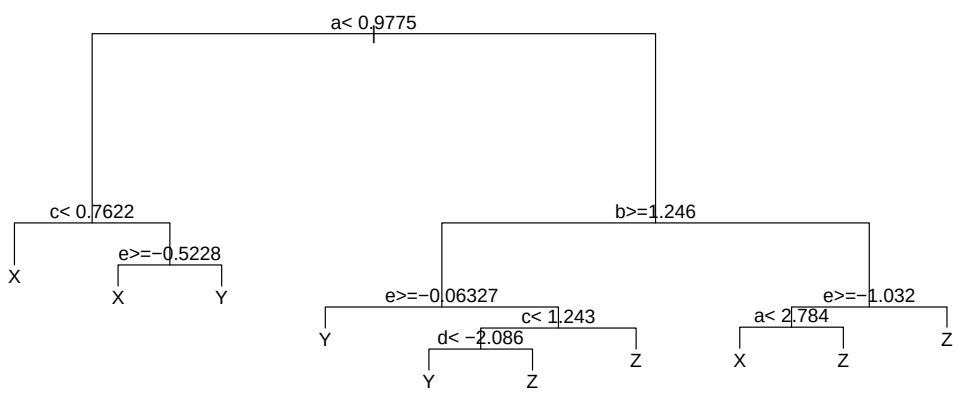


4. If your tree is too small then this is probably because, by default, `rpart` aims to have at least 20 items per branch. You can override this default setting with the optional `control` argument to the `rpart` function. Using the `iris` dataset to illustrate this:

```
my.control <- rpart.control(minsplit=1)
iristree <- rpart(Species ~ ., data=iris, method="class", control=my.control)
```

Apply the same procedure to your mixture of multivariate normal distributions.

```
my.control <- rpart.control(minsplit=1)
tree <- rpart(groups ~ ., data=ldat, method="class", control=my.control)
plot(tree)
text(tree,xpd=NA)
```



5. The `rpart.control` function can also be used to change other settings as well. For example, you can use it to build a maximum sized tree with 'pure' nodes:

```
my.control <- rpart.control(cp=0,minsplit=1)
iristree.unpruned <- rpart(Species ~ ., data=iris,
                           method="class", control=my.control)
```

You can then retrieve a nested set of subtrees by inspecting the cost-complexity table as follows (try it!):

```
printcp(iristree.unpruned)
plotcp(iristree.unpruned)
```

Finally, you can prune the tree using the, guess what, `prune` function. For example:

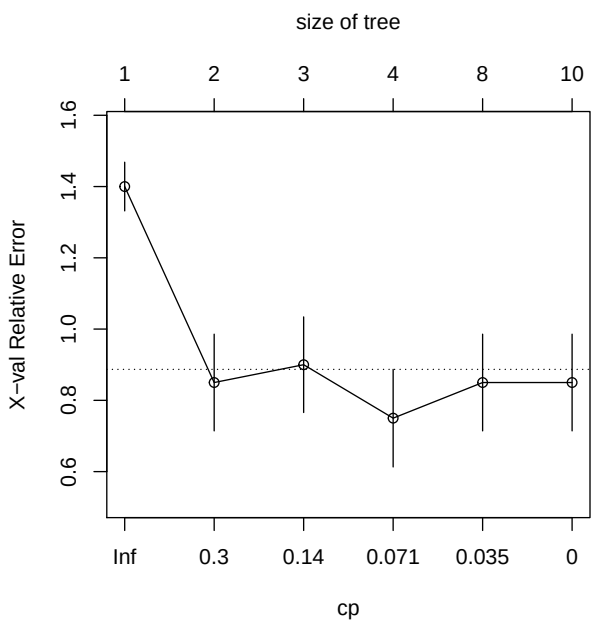
```
iristree.pruned <- prune(iristree, cp=.08)
```

Apply the above functions to your multivariate normal mixture.

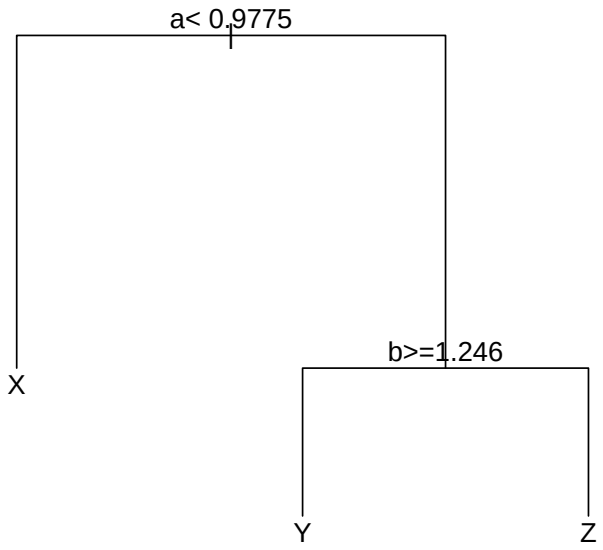
```r
my.control <- rpart.control(cp=0,minsplit=1)
tree.unpruned <- rpart(groups ~ ., data=ldat, method="class", control=my.control)
printcp(tree.unpruned)
```

```
##
## Classification tree:
## rpart(formula = groups ~ ., data = ldat, method = "class", control = my.control)
##
## Variables actually used in tree construction:
## [1] a b c d e
##
## Root node error: 20/30 = 0.66667
##
## n= 30
##
##      CP nsplit rel error xerror     xstd
## 1 0.450      0      1.00   1.40 0.068313
## 2 0.200      1      0.55   0.85 0.135708
## 3 0.100      2      0.35   0.90 0.134164
## 4 0.050      3      0.25   0.75 0.136931
## 5 0.025      7      0.05   0.85 0.135708
## 6 0.000      9      0.00   0.85 0.135708
```

```r
plotcp(tree.unpruned)
```




```r
tree.pruned <- prune(tree.unpruned, cp=.13)
plot(tree.pruned)
text(tree.pruned,xpd=NA)
```


a< 0.9775
X
b>=1.246
Y
Z

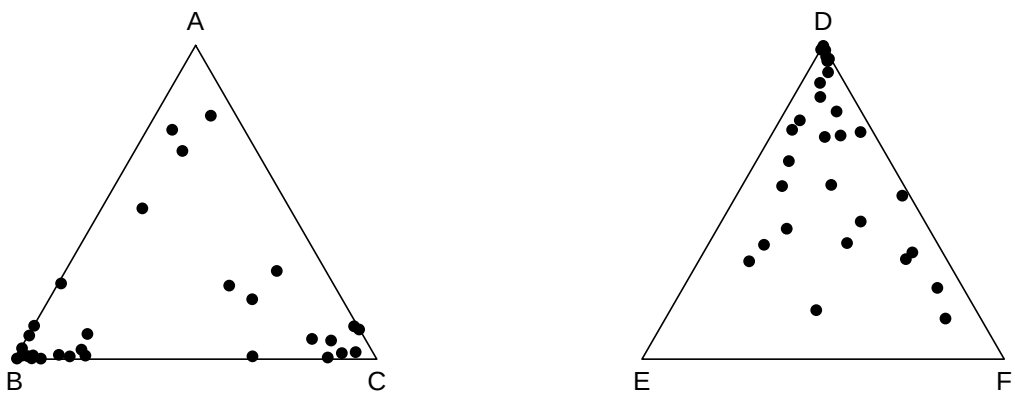
18 Compositional data

Generate a compositional dataset by applying an inverse logratio transformation to the multivariate normal dataset of exercise 16. Label the resulting 6 components A, B, C, D, E and F.

```
cdat <- geostats::alr(dat,inverse=TRUE)
colnames(cdat) <- c('A','B','C','D','E','F')
```

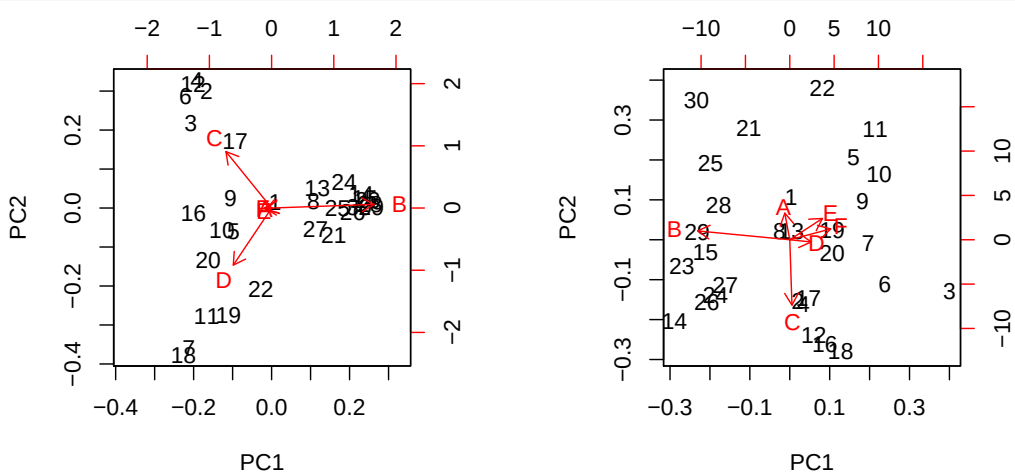
1. Plot the compositional data on two ternary diagrams, grouping components $\{A, B, C\}$ and $\{D, E, F\}$, respectively.

```
par(mfrow=c(1,2))
ternary(cdat[,1:3],pch=16)
ternary(cdat[,4:6],pch=16)
```



2. Carry out a 'naive' PCA analysis, ignoring the compositional nature of the transformed dataset. Then repeat the analysis using compositional PCA. Plot the results of both analyses as biplots. Do you note any differences?

```
par(mfrow=c(1,2))
biplot(prcomp(cdat))
biplot(prcomp(clr(cdat)))
```



3. Using the known groups (X, Y and Z) of the synthetic data, carry out a 'naive' LDA analysis, ignoring the compositional nature of the transformed dataset. Do you get any warnings? Then repeat the analysis using compositional LDA.

```
par(mfrow=c(1,2))
nldat <- data.frame(groups=groups,cdat)
```

```r
nld <- MASS::lda(groups ~ ., data=nldat)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```r
cldat <- data.frame(groups=groups,alr(cdat))
cld <- MASS::lda(groups ~ ., data=cldat)
plot(nld)
plot(cld)
```