Exercise 4.2: Rossler attractor

In 1976 Otto Eberhard Rössler published the paper *An Equation for Continuous Chaos* (Physics Letters, 57A, 397-398). He modeled a chemical reaction mechanism with the system of three coupled differential equations:

\[
\begin{align*}
\frac{dx}{dt} &= -(y + z) \\
\frac{dy}{dt} &= x + ay \\
\frac{dz}{dt} &= b + z(x - c)
\end{align*}
\]

Note that the first two equations are linear, while the only nonlinear term is in the third equation, that is \( z \times x \).

In the original article the parameters have the following values: \( a = 0.20, b = 0.20, c = 5.70 \). In the literature we find also other values, for which the system is chaotic, for instance: \( a = 0.10, b = 0.10, c = 14 \). In the package tseriesChaos, the function rossler.syst uses: \( a = 0.15, b = 0.20, c = 10 \), perhaps these values are the most commonly used, as we do in the following code. The code is reported below. Its structure is the same as the code Lorenz attractor: Complete, but some graphical parameters are changed.

**Exercise:** Study the Rössler system with the code below. The reader is also encouraged to explore what happens by varying the values of \( a, b, c \).

**Solution:** Run the code Rossler attractor: Complete The reader will realise how the choice of the parameters to estimate the MCLE and \( D_2 \) is delicate.

```r
# Rossler attractor: Complete
library(tseriesChaos)
parms<- c(0.15,0.20,10)  # parameters: a, b, c
tinit<- 0
tfin<- 650
step<- 0.1
times<- seq(tinit,tfin,by=step)
funct<- function(t,integ,parms){
x<-integ[1]
y<-integ[2]
z<-integ[3]
a<- parms[1]
b<- parms[2]
c<- parms[3]
dx<- -(y+z)  # that is dx/dt = -(y+z)
dy<- x+a*y   # that is dy/dt = x+ay
dz<- b+z*(x-c) # that is dz/dt = b+z(x-c)
list(c(dx,dy,dz))
}  # end of funct
require(deSolve)
cinit<-c(0,0,0)
xyz<-lsoda(cinit,times,funct,parms)
#xyz # comment if you do not wish the xyz values printed
par(mfrow=c(3,1))
par(mar = c(6.3, 4.8, 1., 3))
par(cex.lab=2,cex.axis=1.6,lwd=1,lty=1)
plot(xyz[,1],xyz[,2],type="l",xlab="t",ylab="x(t)", # x(t) vs t

```

```r
xlim=c(tinit,tfin), ylim=c(-30,30))
plot(xyz[,1],xyz[,3],type="l",xlab="t",ylab="y(t)", # y(t) vs t
xlim=c(tinit,tfin), ylim=c(-30,30))
plot(xyz[,1],xyz[,4],type="l",xlab="t",ylab="z(t)", # z(t) vs t
xlim=c(tinit,tfin), ylim=c(0,50))
windows()
# phase space portrait
require(scatterplot3d)
scatterplot3d(xyz[,2],xyz[,3],xyz[,4],type="l",xlim=c(-20,20),
cex.lab=1.4,cex.axis=1.2)
windows()
par(mai=c(1.02,1.,0.82,0.42)+0.1)
plot(xyz[,2],xyz[,3],type="p",pch=20,cex=0.7,xlab="x(t)",ylab="y(t)",
cex.lab=1.5,cex.axis=1.2,lwd=3,lty=1,xlim=c(-1.5,1.5),ylim=c(-1,1))

trans<- 1000 # integration time step considered as transient
# discard initial transient:
x <- window(xyz[,2],trans)
y <- window(xyz[,3],trans)
z <- window(xyz[,4],trans)
t_start<- trans*step - step # new initial time
t_time<-seq(t_start,tfin,by=step) # new time interval
windows()
par(mfrow=c(3,1))
par(mar = c(6.3, 4.8, 1., 3))
par(cex.lab=2,cex.axis=1.6,lwd=1,lty=1)
# x(t), y(t), z(t) after the transient:
plot(t_time,x,type="l",xlab="t",ylab="x(t)",
xlim=c(t_start,tfin), ylim=c(-30,30))
plot(t_time,y,type="l",xlab="t",ylab="y(t)",
xlim=c(t_start,tfin), ylim=c(-30,30))
plot(t_time,z,type="l",xlab="t",ylab="z(t)",
xlim=c(t_start,tfin), ylim=c(0 ,50))
windows()
par(mai=c(1.02,1.,0.82,0.42)+0.1)
plot(x,y,type="p",pch=20,cex=0.7,xlab="x(t)",ylab="y(t)",
cex.lab=1.5,cex.axis=1.2,lwd=2,lty=1,xlim=c(-1.5,1.5),ylim=c(-1,1))

m_max<- 6 # embedding dimensions: from 1 to m_max
d<- 40 # tentative time delay (see below)
tw<- 100 # Theiler window
rt<- 10 # escape factor
eps<- sd(x)/10 # neighbourhood diameter
fn <- false.nearest(x,m.max,d,tw,rt,eps)
fn
windows()
plot(fn)

windows()
lm<- 60 # largest lag
mutual(x,lag.max = lm) # average mutual information to suggest d
```
# embedding Procedure

m<- 3  # choose embedding dimension
d<- 10 # choose time delay
xyz <- embedd(x,m,d) # embed the ‘observed’ series

windows()
scatterplot3d(xyz, type="l")

windows()
n<- 800
par(mai=c(1.02,1.,0.82,0.42)+0.1)
plot(0,0,type="n",xlab="i",ylab="j",xlim=c(1,n),ylim=c(1,n),cex.lab=1.6,cex.axis=1.2,
family="Times",font.lab=3)
distx<- matrix(,n,n)
eps<- 5
xx<- xyz[,1]
yy<- xyz[,2]
zz<- xyz[,3]
for(i in 1:n){ # construction of the thresholded recurrence plot
  for(j in 1:n){
    # Euclidean distance:
    distx[i,j]<- sqrt( (xx[i]-xx[j])^2 + (yy[i]-yy[j])^2 + (zz[i]-zz[j])^2 )
    if(distx[i,j]<eps) points(i,j,pch=19,cex=0.01)
  }
}

# unthresholded recurrence plot
time<- seq(1:n)
windows()
par(mai=c(1.02,1.,0.82,0.42)+0.1,cex.lab=1.6,cex.axis=1.2)
require(gplots)
distx<- distx/max(distx)
distx<- 1-distx
filled.contour(time,time,distx,col=gray.colors(10,start=1,end=0),nlevels=10,
xlab = "i", ylab = "j", main = "",xlim=c(0,n),ylim=c(0,n),las=0,
key.axes = axis(4, las=1) )

# MCLE: maximum characteristic Lyapunov exponent (Kantz)
S_nu <- lyap_k(x,m=3,d=10,t=20,k=6,ref=5000,s=600,eps=5)
windows()
par(mai=c(1.02,1.,0.82,0.42)+0.1)
plot(S_nu,xlab = expression(paste(nu)),ylab=expression(paste("S",(nu)))),
lwd=2,lty=1,cex=1.5)
lmin<- 80
lmax<- 280
# to add the vertical lines delimiting the linear region:
abline(v=lmin,lty=4,lwd=2,col="black")
abline(v=lmax,lty=4,lwd=2,col="black")
lr<- S_nu[lmin:lmax]    # output of lyap_k(.) in [lmin, lmax]
l_<- length(lr)-1
lt<- seq(lmin*step,lmax*step,by=step)
lm(lr=lt)    # regression analysis in R
abline(lyap(S_nu,lmin,lmax),lty=2,lwd=2,col="black")
# estimate of D2
C.m <- d2(x,m=6,d=10,t=100,eps.min=0.01,neps=100)  # correlation integral, m = 1,...,
C.m <- data.frame(unclass(C.m))  # class attribute removed
C.3 <- subset(C.m,eps > 0.8 & eps < 6, select=c(eps,m3))  # eps in [0.8, 6]
lm(log(m3) ~ log(eps), data = C.3)  # D2 estimate with m=3
windows()  # if other plots are made before
par(mai=c(1.02,1.,0.82,0.42)+0.2)
plot(C.m[,1],C.m[,4], type="l",log="xy",main="",
xlab=expression(paste(epsilon)),ylab=expression(paste(widehat(C),(epsilon))),
lwd=3,lty=1,cex.axis=1.3,cex.lab=2.0, xlim=c(0.01,100),ylim=c(0.00000002,1))
# stop here to plot only C(m=3)

# add the lines below to plot C(m=1,...,6)
n.m<- ncol(C.m)
for(i in (n.m-0):2) lines(C.m[,c(1,i)],lwd=2,lty=2)


The code yields the following figures:

2. Phase space portrait of the chaotic attractor.
3. Projection of the trajectory in the three-dimensional phase space onto the \((x,y)\) plane
4. As the first figure, but without the transient.
5. Projection of the trajectory in the three-dimensional phase space onto the \((x,y)\) plane without the transient.
6. Percentage of false nearest neighbours as a function of the embedding dimension, for the “observed” series \(x\).
7. The average mutual information as a function of the lag, for the “observed” series \(x\).
8. Plot in the reconstructed phase space of the Rössler strange attractor.
9. Thresholded recurrence plot.
10. Unthresholded recurrence plot.
11. Evolution of the logarithm of the mean distance \(S(\nu)\) as a function of the time step \(\nu\).
12. Estimated correlation integral with the embedding dimension from \(m = 1\) to \(m = 6\).
The estimates of the invariants result to be $\hat{\lambda} = 0.075$ and $\hat{D}_2 = 1.90$. With the original parameters $a = 0.20, b = 0.20, c = 5.70$, the invariants are slightly different: $\hat{\lambda} = 0.071$ and $\hat{D}_2 = 1.82$

You can experiment the effect of the noise by adding the same instructions as we did in the code `lorenz attractor`. Recall that for the normal distribution the line is $x \leftarrow x + rnorm(length(x), 0, sd(x)/w)$ (see p. 74), where $sd(x)$ is the standard deviation of the series. You can simulate different noise model, for instance, based on the uniform distribution.