Looking at predictability and chaos of the logistic map

Owen Petchey

30 Apr 2015

Contents

Calculating the Lyapunov exponent (LE) from time series

First simulate two time series

Set the value of r, the number of iterations, the initial abundance of population 1, and the difference between initial abundance of population 1 and population 2:

```
r <- 3.78
N0 <- runif(1, 0.001, 0.1)
N0.diff <- N0/100000
max_its <- 10000 ## length of time series to simulate
plot_nits <- 100 ## length of time series to plot
div_rate_nits <- plot_nits ## length of time series to use for divergence rate calculation of LE
tde_nits <- plot_nits ## length of time series to use for time delayed embedding calc. of LE
```

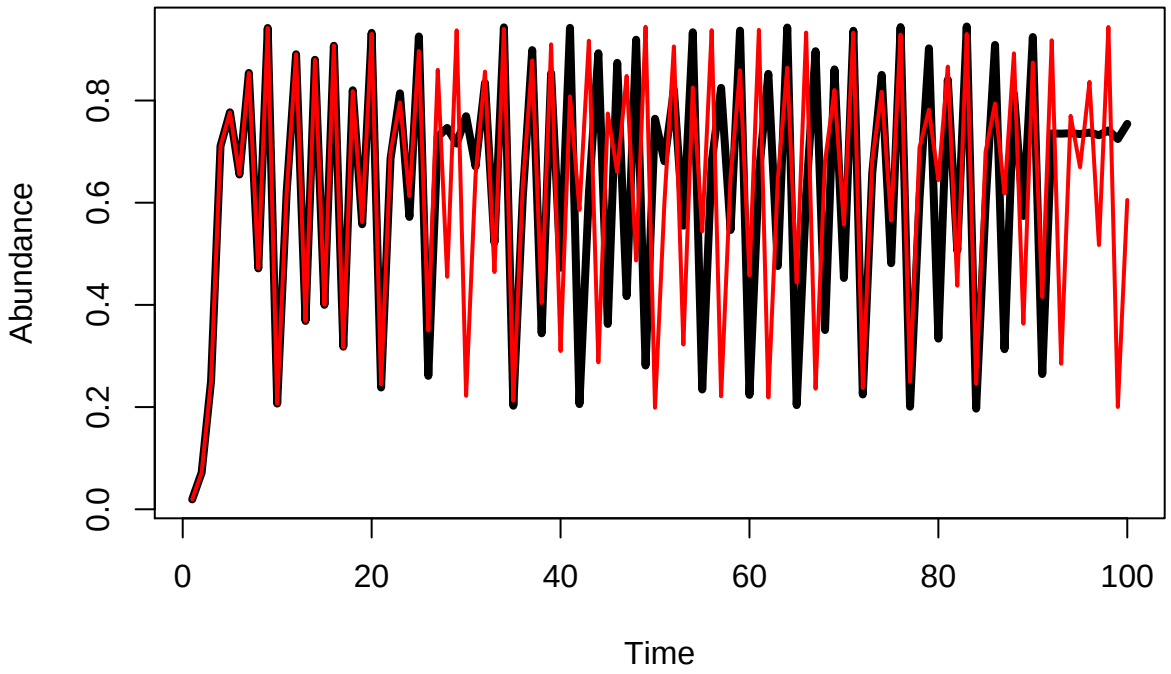
Simulate the logistic map:

```
N1 <- numeric(length=max_its)
N1[1] <- N0
N2 <- numeric(length=max_its)
N2[1] <- N0 + N0.diff
```

```
for(i in 2:max_its) {
  N1[i] <- r * N1[i-1] * (1 - N1[i-1])
  N2[i] <- r * N2[i-1] * (1 - N2[i-1])
}
```

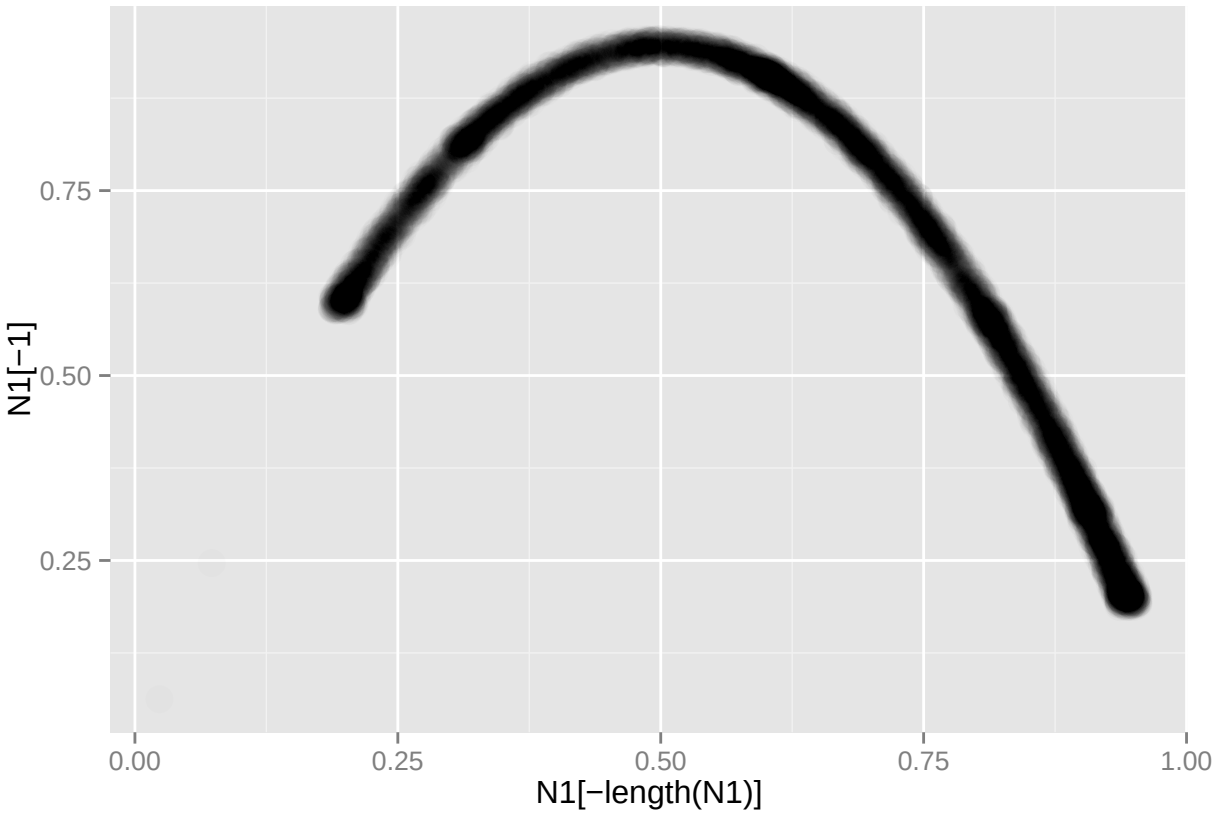
The two series of population dynamics:

```
plot_nits <- 100 ## number of iterations to plot
plot(1:plot_nits, N1[1:plot_nits], type="l",
     xlab="Time",
     ylab="Abundance",
     lwd=4)
lines(1:plot_nits, N2[1:plot_nits], type="l",
      lwd=2, col="red")
```



Look at the attractor (with some jitter):

```
qplot(x=N1[-length(N1)], alpha=I(0.01), size=I(5),
      y=N1[-1], position = position_jitter(w = 0.01, h = 0.01))
```



Estimate Lyapunov exponent from growth in difference

Distance between two time series

Get the absolute difference between the two time series:

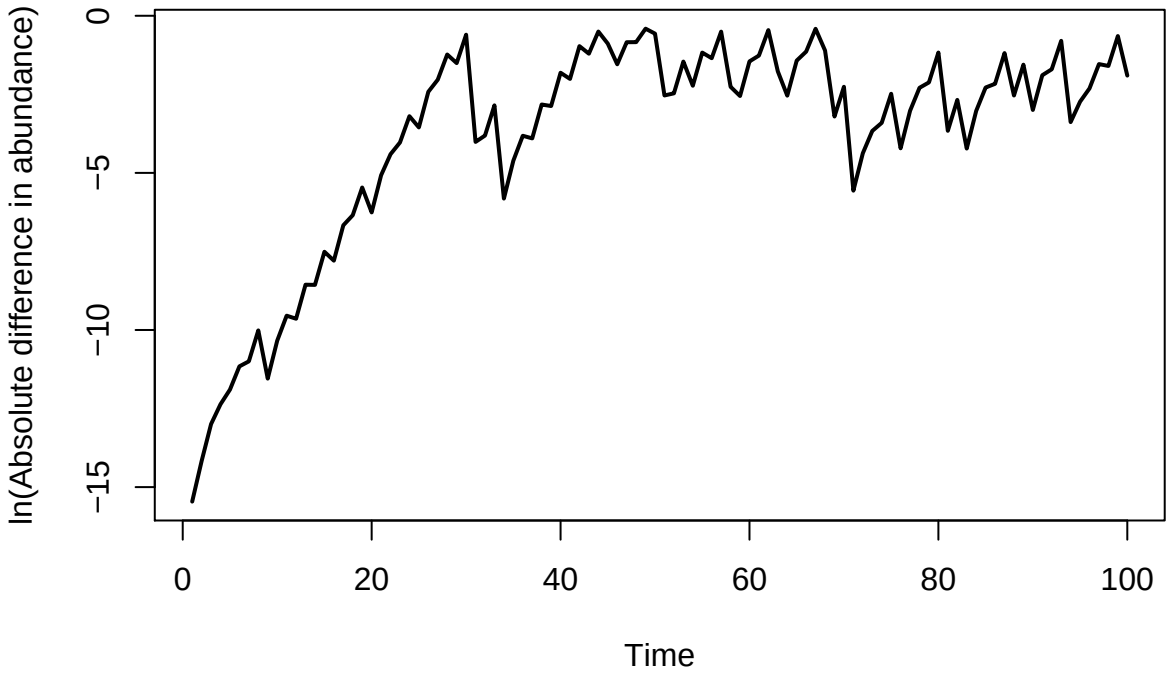
```
diffs <- abs(N1-N2)[1:div_rate_nits]
```

And remove any zeros (including their positions from a vector of times), since these will cause problems when we need to log the difference:

```
times <- 1:length(diffs)
times <- times[diffs!=0]
diffs <- diffs[diffs!=0]
```

Plot the difference between the two time series through time:

```
  plot(times, log(diffs), type="l",
       xlab="Time",
       ylab="ln(Absolute difference in abundance)",
       lwd=2)
```

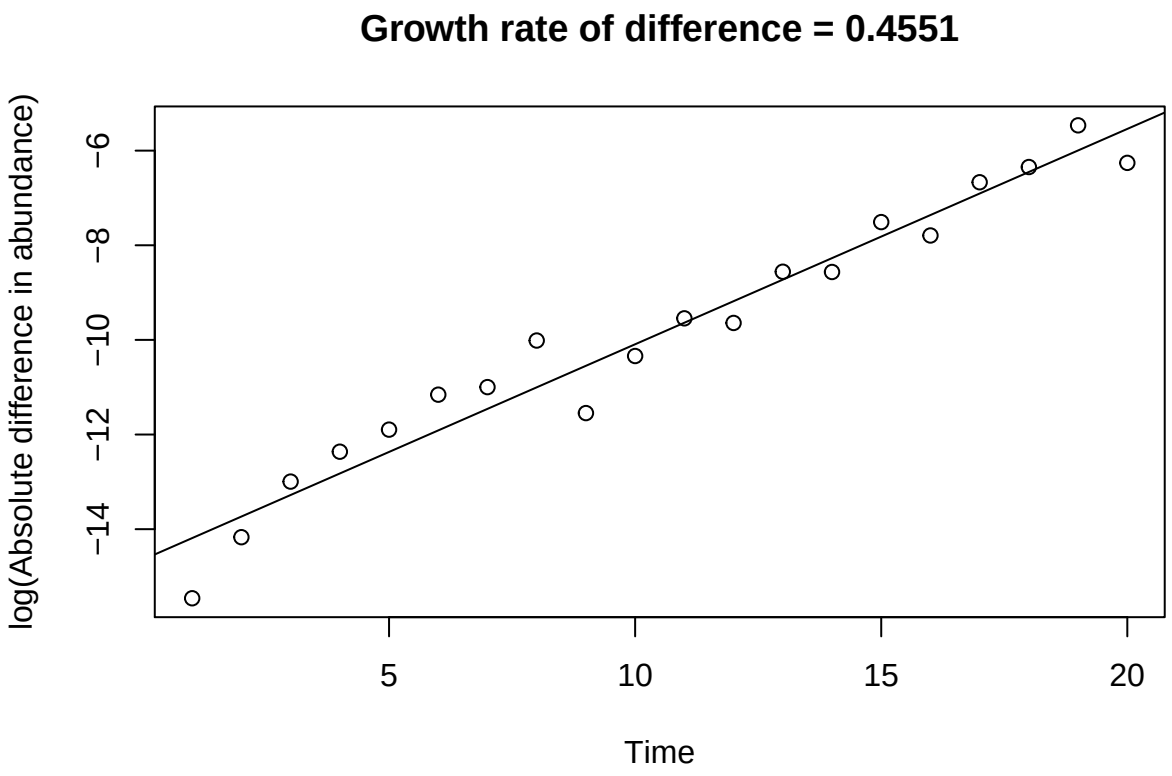


From this graph, select a range of times over which to calculate the growth rate of difference (be careful to choose a time range that contains data):

```
min.time <- 1 ## not less than 1 please
max.time <- 20
```

And plot this part of the data with the estimated Lyapunov exponent:

```
x <- min.time:max.time
y <- log(diffs)[min.time:max.time]
plot(x, y, type="p",
     xlab="Time",
     ylab="log(Absolute difference in abundance)",
     main=paste("Growth rate of difference =",  round(coef(lm(y~x))[2],4)))
abline(lm(y~x))
```

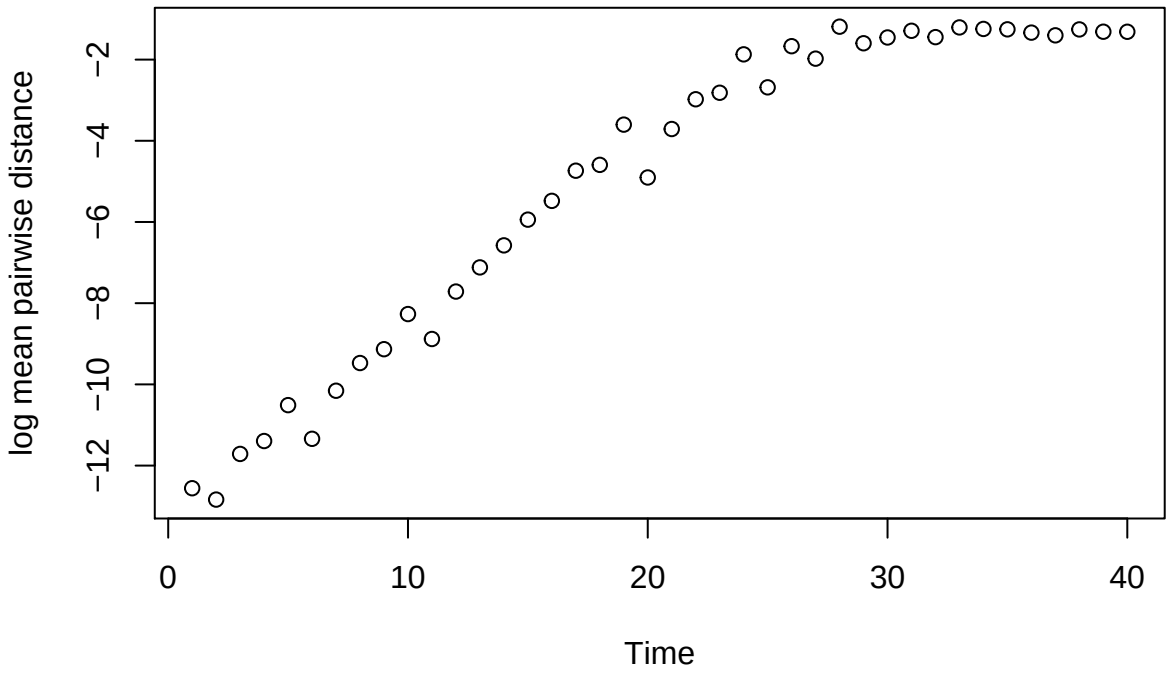



```
le0 <- coef(lm(y~x))[2]
```

Average distance between lots of time series

This would recreate the method used to produce figure 5 in Nychka et al (1992) Finding Chaos in Noisy Systems. (Not yet implemented.) Calculates the mean pairwise distance among a cohort of trajectories through time. (By the way, Nychka et al (1992) looks at calculating LE for noisy systems, and figure 5 shows growth of distance among trajectories is common to deterministic and stochastic chaotic systems.)

```
number_of_time_series <- 100
nits <- 40
Ns <- matrix(0, nits, number_of_time_series)
Ns[1,] <- runif(number_of_time_series, 0.4, 0.40001)
for(i in 1:number_of_time_series)
  for(j in 2:nits)
    Ns[j, i] <- r * Ns[j-1, i] * (1 - Ns[j-1, i])
log.diff <- log(apply(Ns, 1, function(x) mean(abs(dist(x)))))
plot(log.diff,
     xlab="Time", ylab="log mean pairwise distance")
```




```
time.to <- 20
ttimes <- 1:time.to
le0.1 <- coef(lm(log.diff[ttimes] ~ ttimes))[2]
```

Calculating the Lyapunov exponent by time-delayed embedding (a direct method)

This is the method uesd by Beninca et al. (Nature, 2008).

pdf of M. T. Rosenstein, J. J. Collins, C. J. De Luca, A practical method for calculating largest Lyapunov exponents from small data sets, Physica D 65, 117 (1993)

Do this for both of the time series made above. Owen guessed the parameter values below:

```
output1 <- lyap_k(N1[1:tde_nits], m=4, d=4, ref=4, t=4, s=40, eps=4)
```

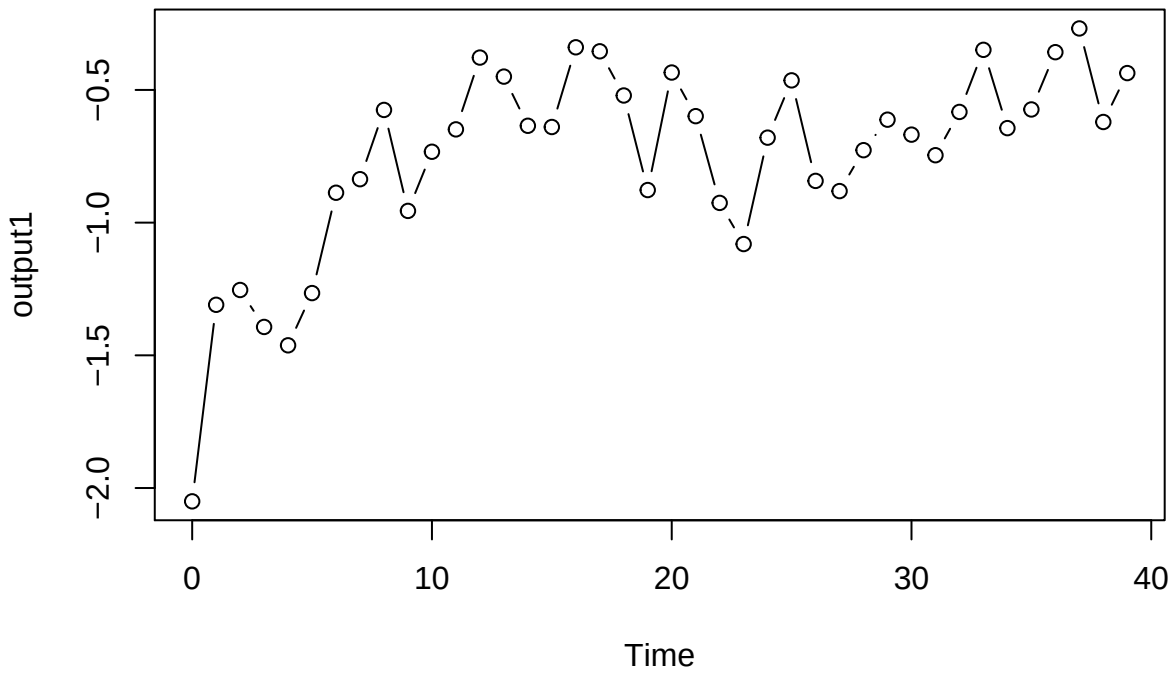
```
## Finding nearests
## Keeping  4  reference points
## Following points
```

```
output2 <- lyap_k(N2[1:tde_nits], m=4, d=4, ref=4, t=4, s=40, eps=4)
```

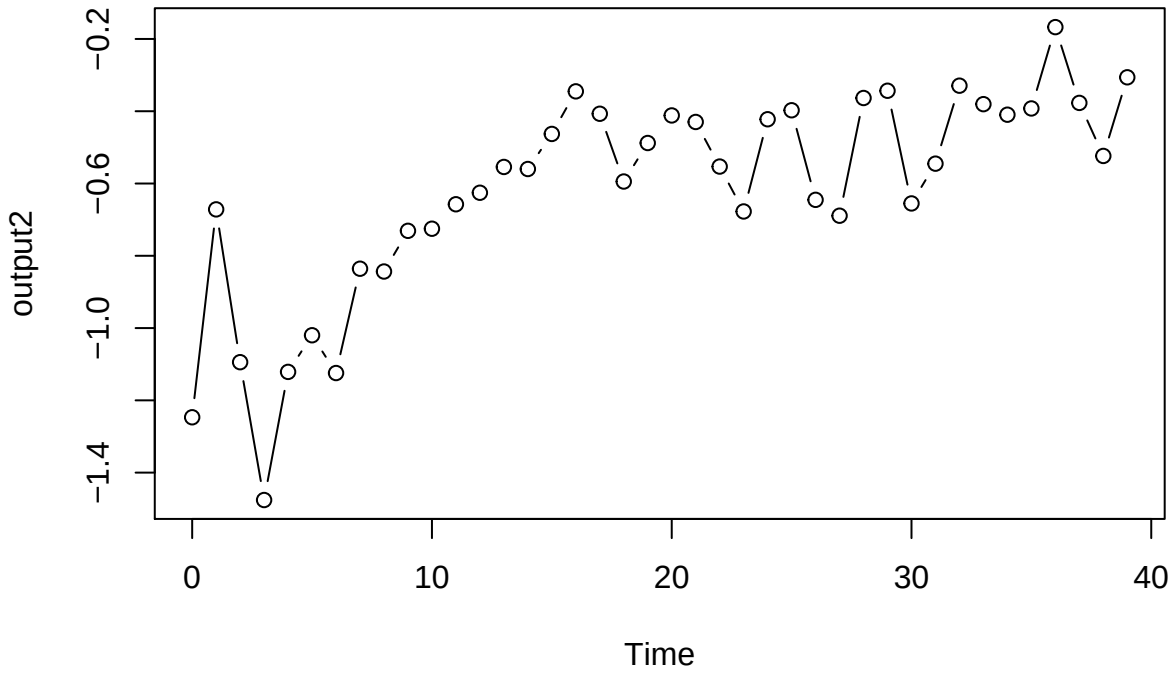
```
## Finding nearests
## Keeping  4  reference points
## Following points
```

Again need to carefully select the time range over which to calculate the Lyapunov exponent. Find it from these graphs

```
plot(output1, type="b")
```




```
plot(output2, type="b")
```



Select time range for each Lyapunov exponent (this is really fudgy at present!):

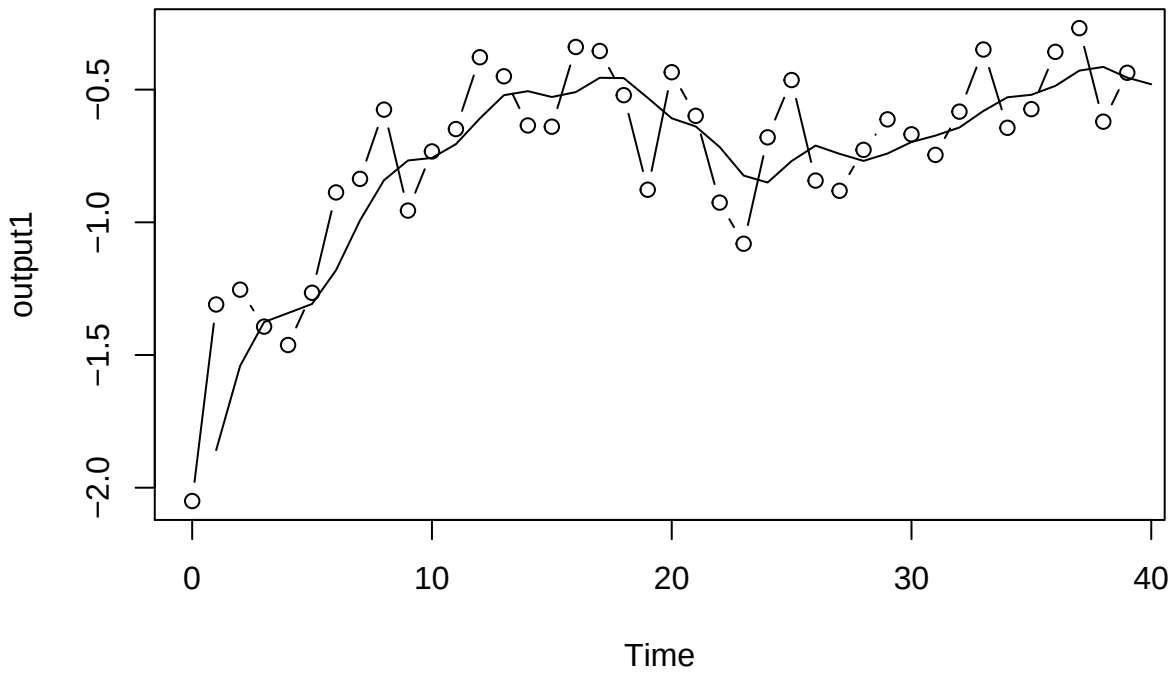
```
start <- 4
end <- 8
le1 <- lyap(output1, start, end)[2]
```

Get the two Lyapunov exponents:

```
start <- 4
end <- 8
le2 <- lyap(output2, start, end)[2]
```

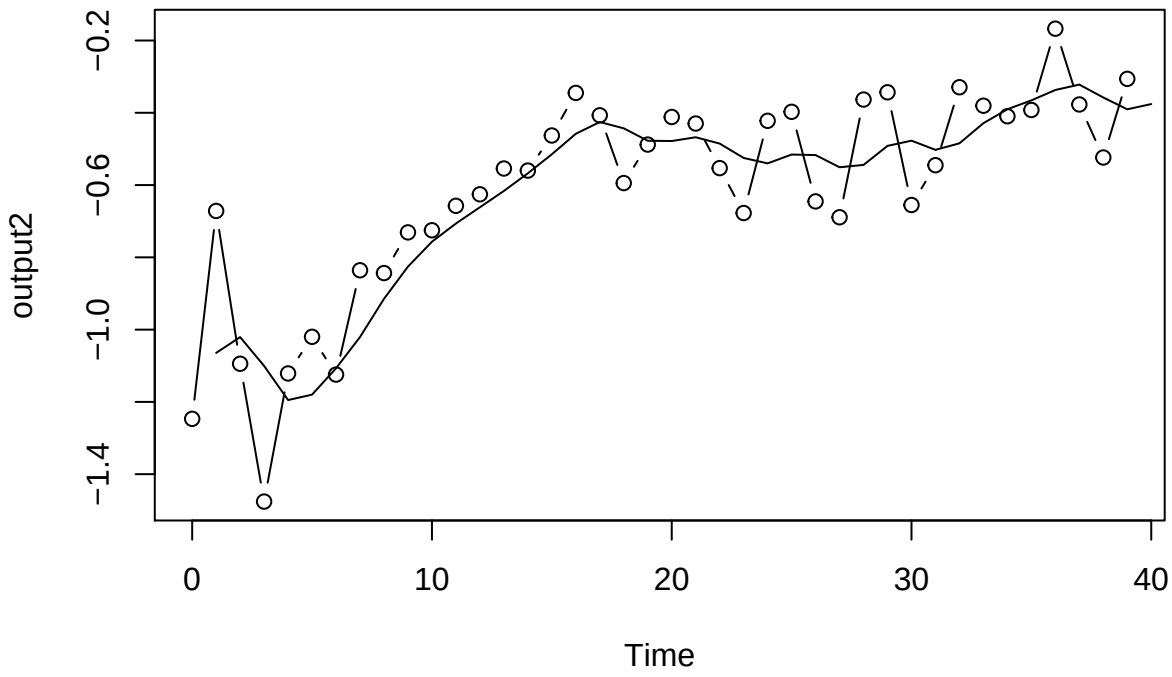
Another option would be to fit a spline to the data and estimate the max derivative of this.

```
plot(output1, type="b")
spline1 <- smooth.spline(1:length(output1), output1, df=15)
lines(spline1)
```




```
le1.1 <- max(predict(spline1, deriv=1)$y)
```

```
plot(output2, type="b")
spline2 <- smooth.spline(1:length(output1), output2, df=15)
lines(spline2)
```




```
le2.1 <- max(predict(spline2, deriv=1)$y)
```

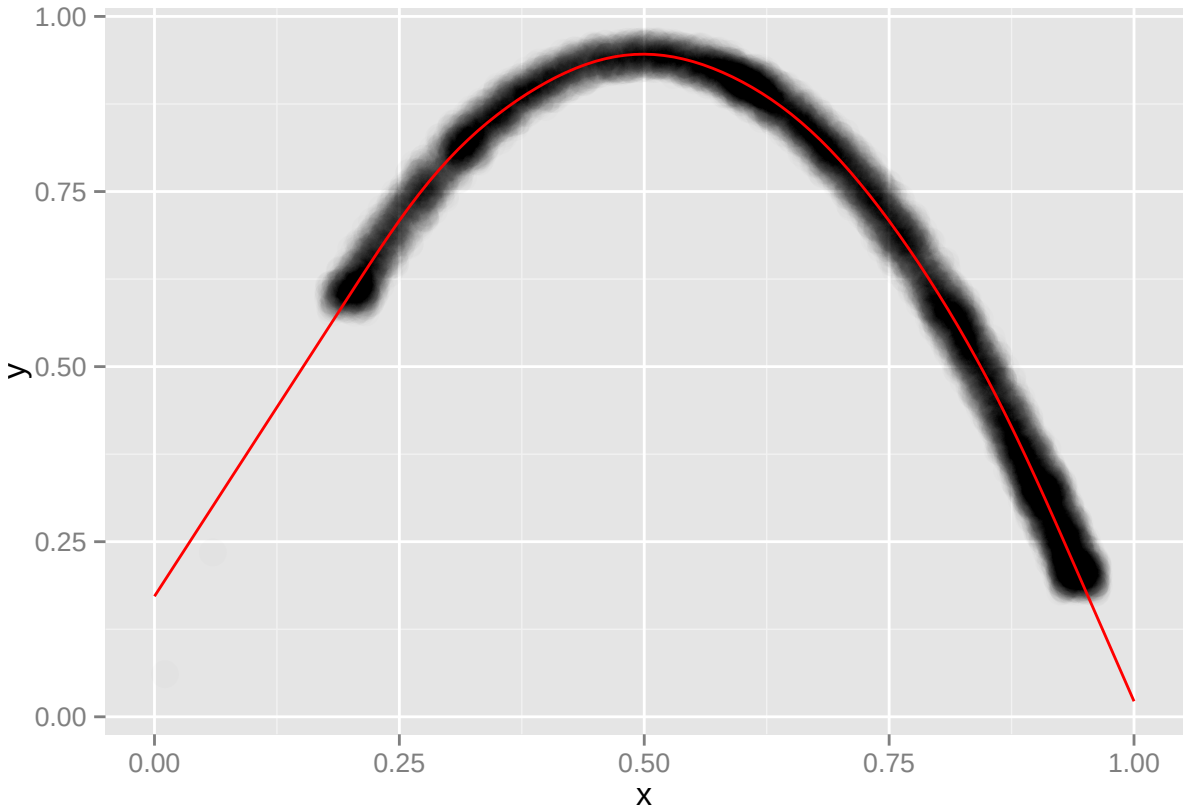
The Jacobian method

```
hh1 <- data.frame(x=N1[-length(N1)], y=N1[-1])

gamfit <- gam(y ~ s(x), data=hh1, gamma=1)
preds <- data.frame(x=seq(0, 1, 0.01))
preds <- cbind(preds, predict(gamfit, newdata=preds, se.fit=T))
str(preds)
```

```
## 'data.frame':    101 obs. of  3 variables:
##  $ x     : num  0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 ...
##  $ fit   : num [1:101(1d)] 0.172 0.194 0.215 0.237 0.258 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ : chr  "1" "2" "3" "4" ...
##  $ se.fit: num [1:101(1d)] 0.000511 0.000488 0.000465 0.000443 0.00042 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ : chr  "1" "2" "3" "4" ...
```

```
ggplot(hh1, aes(x=x, y=y)) +
  geom_point(alpha=0.01,  size=5, position = position_jitter(w = 0.02, h = 0.02)) +
  geom_line(aes(x=x, y=fit), data=preds, col="red") #+
```




```
#  geom_ribbon(aes(x=x, ymin=fit-se.fit, ymax=fit+se.fit), data=preds)
## not sure why I can't get the se.fit ribbon working



## the follow is inspired (=copied) from code provided by Stephen Ellner.
## Owen doesn't fully understand it.

#z <- hh1
#eps <- epsval
gamgrad=function(z, gamfit,eps) {
    gradmat=matrix(0,dim(z)[1],1);
    for(j in 1:1) {
        newzup=z; newzdown=z;
        newzup[,j]=z[,j]+eps; newzdown[,j]=z[,j]-eps;
        gradmat[,j]=predict(gamfit,newzup,type="response")-predict(gamfit,newzdown,type="response");
    }
    gradmat=gradmat/(2*eps)
    return(gradmat);
}


nt=length(hh1$x)
Jac=array(0,c(1,1,nt))
#Jac=array(0,c(12,12,nt))
epsval=0.01
Jac[1,1,1:nt]= t(gamgrad(hh1, gamfit,eps=epsval))
```

```
#############################################################################################
##  Compute GLE using the Jacobians
#############################################################################################

u=rep(1,1); u=u/max(abs(u)); LE=0;
for (j in 1:nt) {
    u=Jac[1,1,j]%*%u
    umax=max(abs(u))
    LE=LE+log(umax)
    u=u/umax;
}

LE=LE/nt
LE.jac <- LE
```

Analytical calculation of Lyapunov exponents

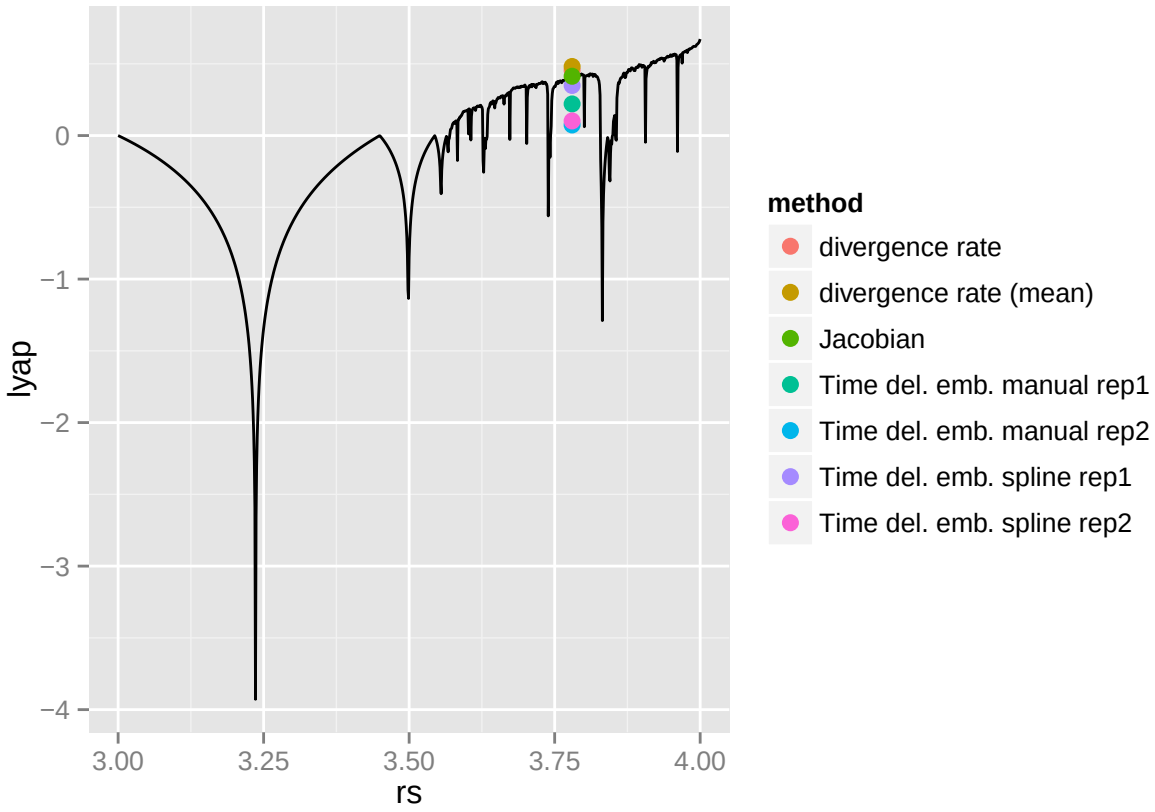
The following calculates the Lyapunov exponent from the value of r, assuming exponential growth (decline) in small differences between two time series, via some differentiation by the chain rule. The code is not evaluated to make this document; previous made data is loaded, for speed.

```
rs <- seq(3, 4, by=0.001)
lyap <- rep(NA, length(rs))
nits <- 10000
for(j in 1:length(rs)) {
  xn1 <- runif(1)
  lyp <- 0
  for(i in 1:nits) {
    xn <- xn1
    xn1 <- rs[j]*xn*(1-xn);
    if(i>300)
      ## The next line is the from some maths that Owen doesn't fully understand!
      lyp <- lyp + log(abs(rs[j]-2*rs[j]*xn1))
  }
  lyp <- lyp/nits
  lyap[j] <- lyp
}
#write.csv(cbind(rs, lyap), "~/Desktop/rs_lyaps.csv")
```

Load the previously created data from github (code might need adapting for windows):

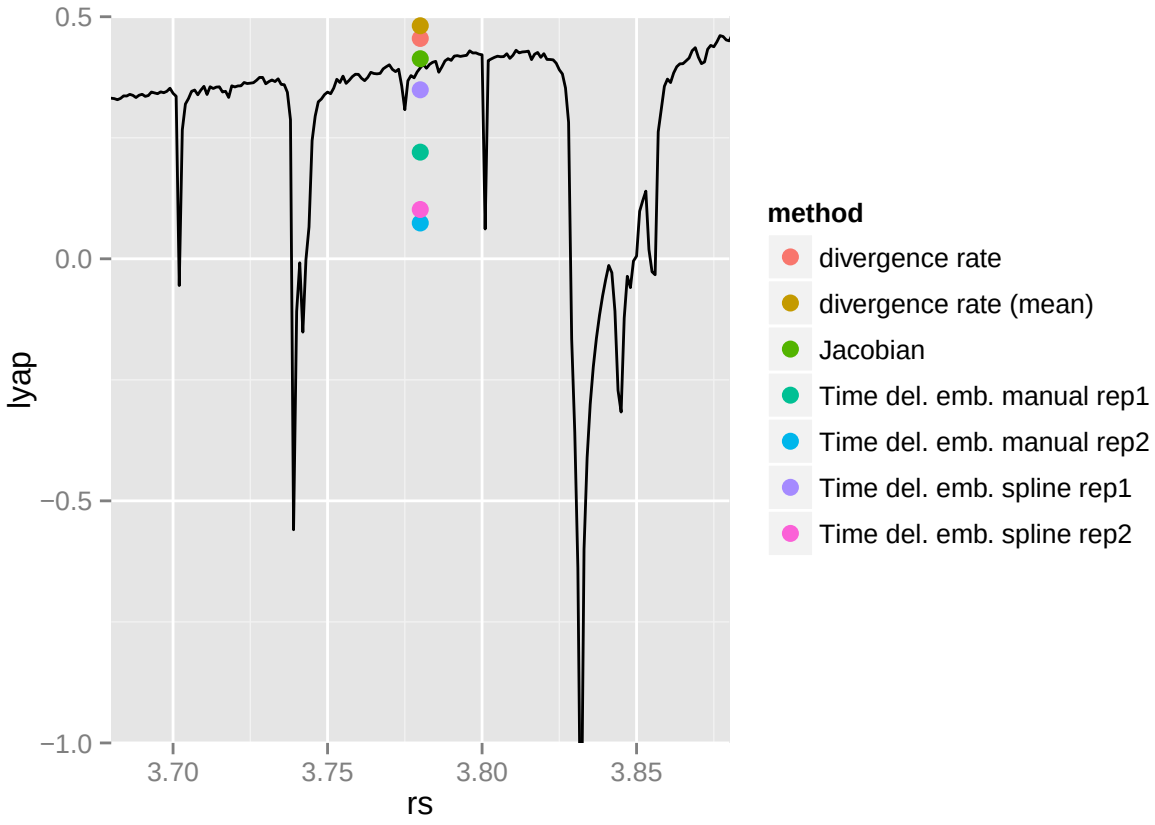
Comparison of the methods

```
qplot(x=rs, y=lyap, data=filter(dd.all, method=="analytic"), geom="line") +
  geom_point(aes(x=rs, y=lyap, col=method), size=3, data=filter(dd.all, method!="analytic"))
```



And zoomed in:

```
qplot(x=rs, y=lyap, data=filter(dd.all, method=="analytic"), geom="line") +
  geom_point(aes(x=rs, y=lyap, col=method),  size=3, data=filter(dd.all, method!="analytic")) +
   coord_cartesian(xlim=c(r-0.1, r+0.1), ylim=c(-1, 0.5))
```




```
dd.all[c(which(dd$rs==r), which(dd.all$method!="analytic")),]
```

```
##        rs       lyap                    method
## 781  3.78 0.39437794                  analytic
## 1002 3.78 0.45514675           divergence rate
## 1003 3.78 0.48136894    divergence rate (mean)
## 1004 3.78 0.22031208 Time del. emb. manual rep1
## 1005 3.78 0.34907857 Time del. emb. spline rep1
## 1006 3.78 0.07396539 Time del. emb. manual rep2
## 1007 3.78 0.10186980 Time del. emb. spline rep2
## 1008 3.78 0.41326404                 Jacobian
```

To do

- Check the importance of the parameters of time delayed embedding.
- Get multiple estimates from the same method (distribution of estimates).
- Trace the different methods across a range of r values (analytic result).
- What if we did an experiment in which the effect of a treatment was to influence r?