# NRES 778: Bayesian Hierarchical Modeling in Natural Resources

Lab 01: Programming in R

## 1 Objective

The purpose of this lab is to exercise what you have learned (or need to learn) about some important topics in R programming, including:

1. writing functions,
2. creating data structures,
3. looping,
4. sub-setting matrices,
5. and plotting.

## 2 Exploring chaos with the discrete logistic equation

In 1976, Robert May authored a classic paper (May, 1976) revealing chaotic dynamics in discrete time models. He exercised the discrete logistic equation,

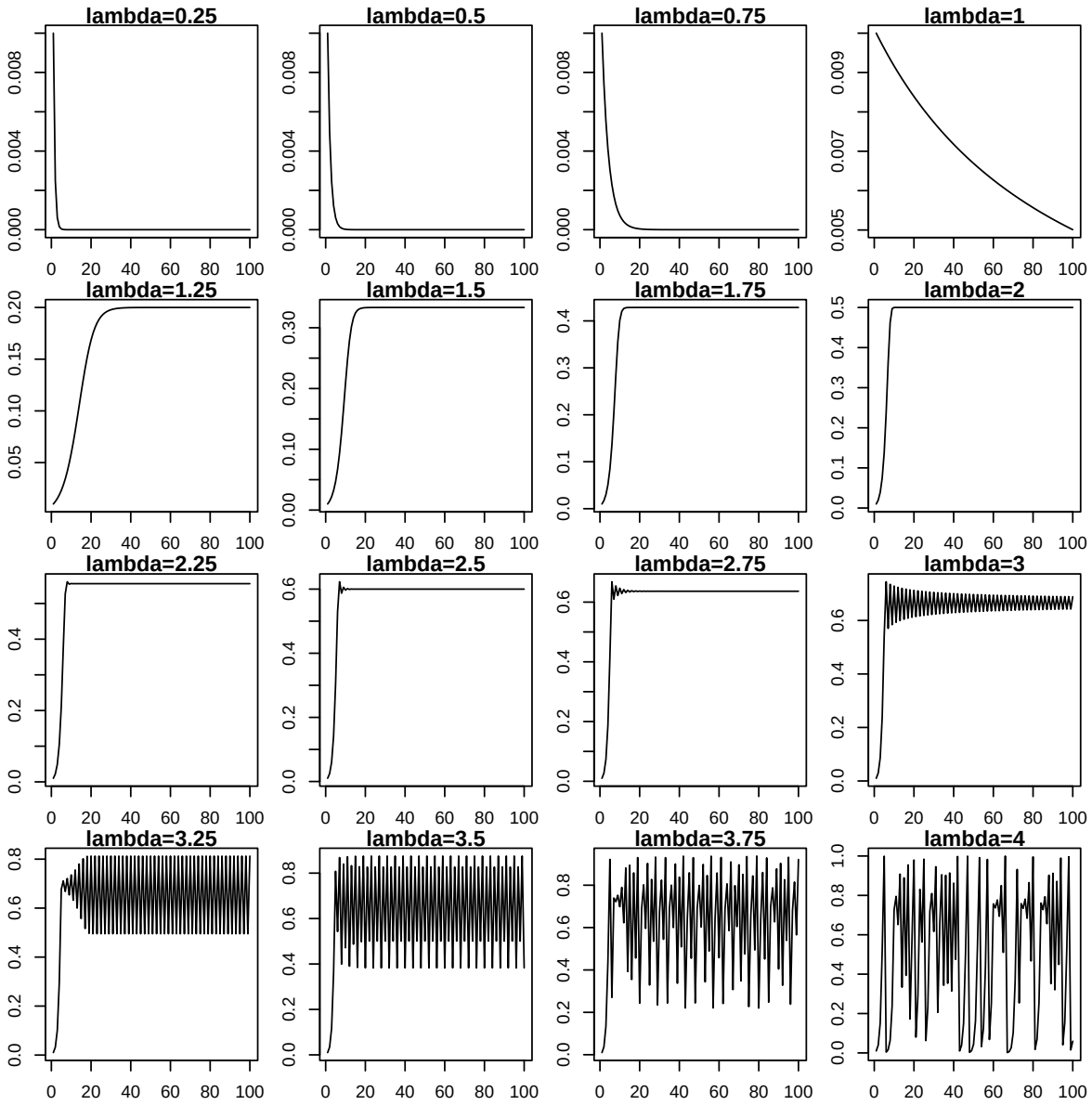
$$x_{t+1} = \lambda x_t(1 - x_t) \tag{1}$$

where $\lambda$ is the per capita rate of population growth and $x_t$ is the population size at time $t$. This form of the logistic equation may be unfamiliar to you because it lacks the parameter, $K$. May (1976) used a mathematical trick to rescale the equation to eliminate $K$, allowing us to focus our attention on the effect of $\lambda$ on the populations dynamics. In your first exercise, you will vary $\lambda$ and observe its influence on the trajectory of a simulated population.

1. Write an R function for equation 1. Use your function to simulate how population size changes over time in response to variation in the parameter $\lambda$. Set up a model experiment with an outer `for` loop controlling the value of $\lambda$ to range from .25 to 4.0 in steps of .25. Create an inner loop varying time from 2 to 30 in steps of 1. Assume that the initial condition, $x_1$ (i.e., the value of $x$ at time = 1), for the population size is 0.01. Create a plot of $x$ as a function of time for each value of $\lambda$. You should display your plots with $4 \times 4$ panels, one panel for each of your simulations. Give each plot a title showing the value of $\lambda$. (Hint, convert the numeric value of $\lambda$ to a character value using z = as.character(lambda) and use main = z as an option in the plot statement). What can you conclude about the effect of $\lambda$ on the dynamics? Use your panel plot to illustrate the points in your discussion. For an engaging time waster, figure out how to put a symbolic $\lambda$ in the title of your plots. So, you will need to combine a symbolic $\lambda$ with a numeric value that changes with each plots title. See `?mathplot` for hints.

```
calc.x2=function(x,lambda){
  output=lambda*x*(1-x)
  return(output)
}
```

```
lambda=seq(.25,4,.25)
time=seq(1,100,1)
T=length(time)
x=matrix(NA,T,length(lambda))
x[1,]=0.01

for(i in 1:length(lambda)){
  for(t in 2:T){
    x[t,i]=calc.x2(x[t-1,i],lambda[i])
  }
}

par(mfrow=c(4,4),mar=c(2,2,1,1))
for(i in 1:length(lambda)){
  plot(1:T,x[,i],type='l',
       main=paste("lambda=",lambda[i],sep="")
       )
}
```



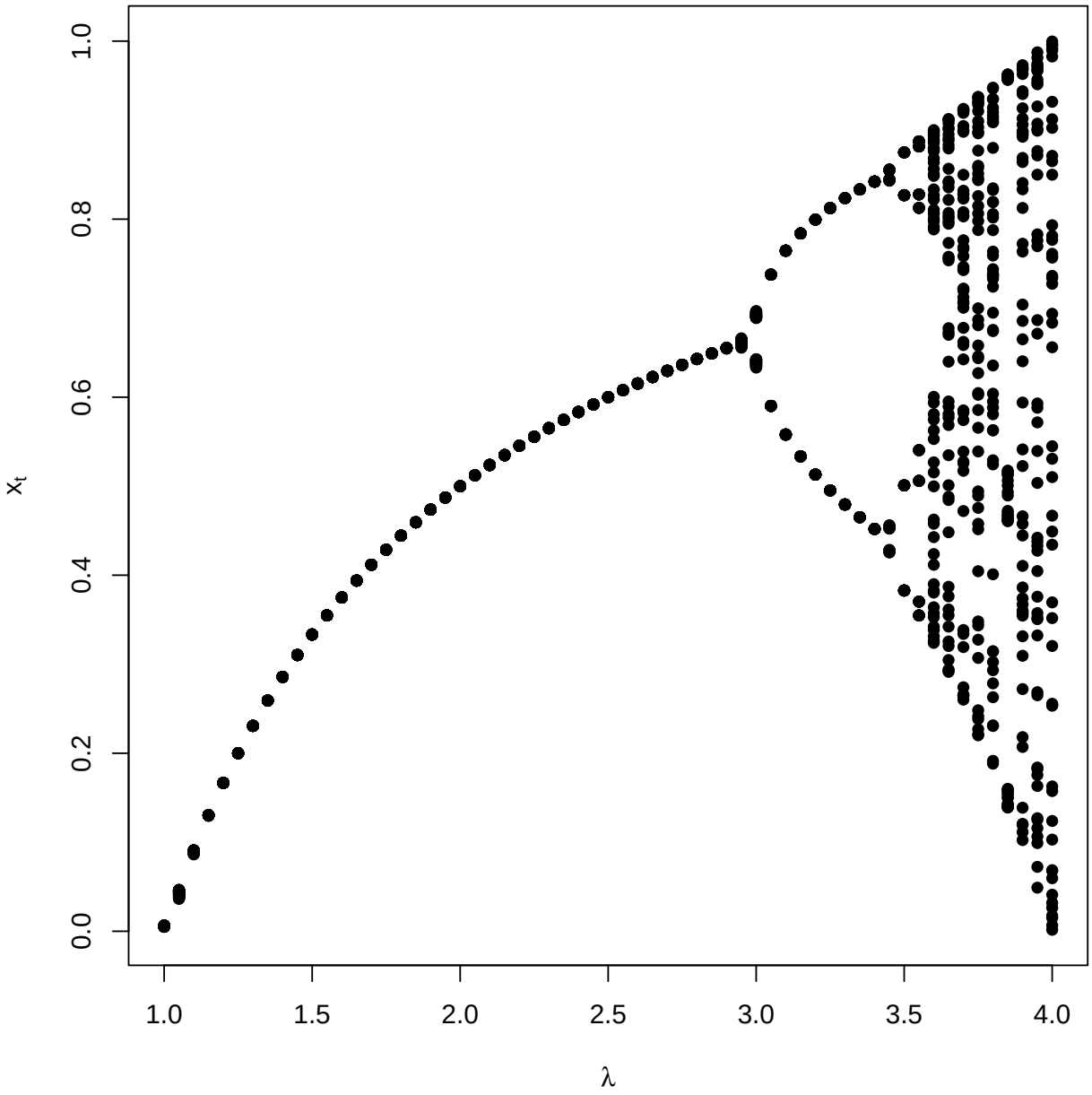

2. Now set up a model experiment where you vary $\lambda$ in steps of 0.05 from $\lambda = 1$ to $\lambda = 4$. Run the model for 100 time steps and save the values of $x_t$ for the last 50 time steps ($t = 51 - 100$) to a matrix where column one contains the value of $\lambda$ and column two contains the value of $x_t$. (Discard the values of $x + t$ for $t < 51$). Plot the values of $x$ at $t = 51$ to 100 as a function of $\lambda$. The trick to this problem is thinking about how to store your data relative to the iteration that is going on (which is what makes it a good problem). Think about it; you need a vector that keeps track of $x_t$ as $t$ goes from 1 to 100. You also need a matrix that consists of groups of rows of data, where each group consists of 50 rows of each value of $\lambda$ in one column and the values of $x_t$ at time = 51-100 in the other column. The full stack of these rows is the matrix. As a hint, you could implement this by creating a counter within your loops (say, $j = j + 1$) that you use to index the rows of the array used to store the values of $\lambda$ and $x_t$. At the end of your model experiment, that is, after all of the looping is complete, the counter $j$ will equal the number of rows in the data matrix. Alternatively (and I think better) you might use the `rbind(...)` function without needing a counter. For example, consider the following code illustrating how to build a matrix of data by accumulating rows one at a time:

```
M=NULL
for(j in 1:4){
  v=c(1,1,j)
  M=rbind(M,v)
}
M
```

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

Your plot should look something like this:

```
lambda=seq(1,4,.05)
n=length(lambda)
time=seq(1,100,1)
T=length(time)
x.save=matrix(NA,1,2)

for(i in 1:n){
  for(t in 1:T){
    if(t==1){
      x=0.01
    }
    if(t>1){
      x=calc.x2(x,lambda[i])

      if(t>50){
        output=c(lambda[i],x)
        x.save=rbind(x.save,output)
      }
    }
  }
}
par(mfrow=c(1,1),mar=c(4,4,1,1))
plot(x.save[,1],x.save[,2],pch=16,
     xlab=expression(lambda),
     ylab=expression(x[t]))
```




*Hint: before the first split in $x$, there are 50 dots stacked on top of one another.*

The plot you produce, done correctly, is called a bifurcation diagram. Interpret what you see in this diagram. It will be helpful to think about this diagram by comparing it to plots of $x_t$ as a function of $t$ for different values of $\lambda$ that you accomplished in the first exercise. It might be helpful to re-run that code allowing time to go to 100 rather than 30. Explain the relationship between the bifurcation diagram and the 16 plots that you produced in exercise 1.

# 3 Fitting a logistic equation to data

This exercise anticipates work that we will do during the next few weeks. One of the ways that we can estimate parameters in non-linear models is to minimize the sum of squared differences between the model predictions and the observations (squared-error loss function). Presume we have a deterministic model, $f$, that is a function of model parameters, $\boldsymbol{\theta}$, and makes predictions $\hat{y}_i = f(\boldsymbol{\theta})$ of observations $y_i$. We search for a set of parameter values $\boldsymbol{\theta}^* \in \boldsymbol{\Theta}$

such that

$$\theta^* = \min_\theta \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Find the best-fit parameter values (i.e., $\boldsymbol{\theta} \equiv (r, K, N_1)'$) for a logistic model of growth of the Rocky Mountain National Park elk population,

$$N_{t+1} = N_t + rN_t\left(1 - \frac{N_t}{K}\right), \tag{2}$$

by finding the parameter values and initial conditions producing the minimum of the sum of squared differences between the observed population size and the predicted population size. Use the data 'RMNP elk time series.csv.

Here are some hints. For this problem, I want you to use a brute force approach to parameter estimation; we will use more elegant methods later. To make this problem a bit easier, you can reasonably guess that the value of $r$ is somewhere between 0.1 and 0.3. Moreover, you can look at the data and be sure that $K$ is somewhere between 800 and 1200 and that the initial conditions, $N_1$ are somewhere between and 200 and 500. So, to estimate the parameters, you will calculate the sum of squared error for all possible combinations of parameter values and find the combination that gives the minimum SSE.

Here are some more hints. Set up a matrix with 4 columns, one each for $N_1$, $r$, $K$, and SSE. Set up 3 nested loops to vary each parameter at fairly coarse intervals to start with (say 10 for $N_1$ and $K$, and .05 for $r$). Within the innermost parameter loop, you will have a loop for time, calculating the squared error for each model prediction and observation. After you exit this loop, you can calculate sum of the squared error over all of the years in your simulation ($T = 41$). You will need a counter embedded in this loop to index the rows of the SSE array. Once you have created the SSE array, find the row of your matrix containing the minimum value of SSE. See the section on arrays in the R manual if you need refreshing on this. The which() function will prove useful. This row will contain your parameter estimates.

After you have your program working, you can make the intervals in you for loop finer to give you more accurate estimates of the parameters (say 5 for $N_1$ and $K$, and .01 for $r$).

Subset the data to create 3 data sets so that you can the plot the sum of squared error as a function of each model parameter and the initial condition ($r, K, N_1$) with the other parameter and initial conditions set at their best fit values. Your plots should look something like those shown below.

```
elk=read.csv(paste0("~/Dropbox/Teaching/NRES779/",
                    "2021Spring/Labs/Lab_01/RMNP elk time series.csv"))

time=1965:1999
N=numeric(length(time))

###
### parameters?
###

r=0.2
K=1200
N[1]=500

calc.lg=function(N1,r,K,T){
  N=numeric(T)
  N[1]=N1
  for(t in 2:T){
    N[t]=N[t-1]+r*N[t-1]*(1-N[t-1]/K)
  }
  return(N)
}
```

```r
y.hat=calc.lg(500,r=0.2,K=1200,T=35)
y=elk$Population_size
score=sum(abs(y-y.hat))
score
```

```
## [1] 8857
```

```r
r.candidates=seq(0,1,0.01)
N1.candidates=seq(300,600,5)
K.candidates=seq(400,1600,5)


## [r,N1,K]
R=length(r.candidates)
eta=length(N1.candidates)
epsilon=length(K.candidates)
score.save=array(,c(R,eta,epsilon))
y=elk$Population_size

tok=Sys.time()
for(i in 1:R){
  for(j in 1:eta){
    for(l in 1:epsilon){
      y.hat=calc.lg(N1.candidates[j],r=r.candidates[i],
                    K=K.candidates[l],T=35)
      score.save[i,j,l]=sum((y-y.hat)^2)
    }
  }
}
tik=Sys.time()
tik-tok
```

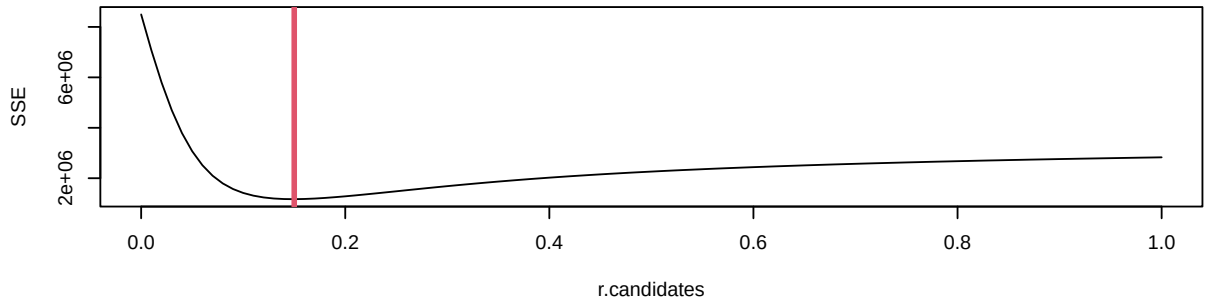
```
## Time difference of 18 secs
```

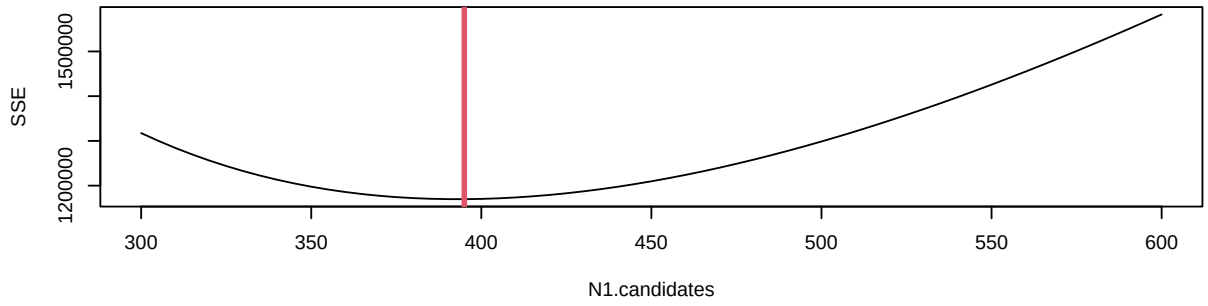
```r
## score.save
index=which(score.save==min(score.save),arr.ind=TRUE)
score.save[index[1],index[2],index[3]]
```

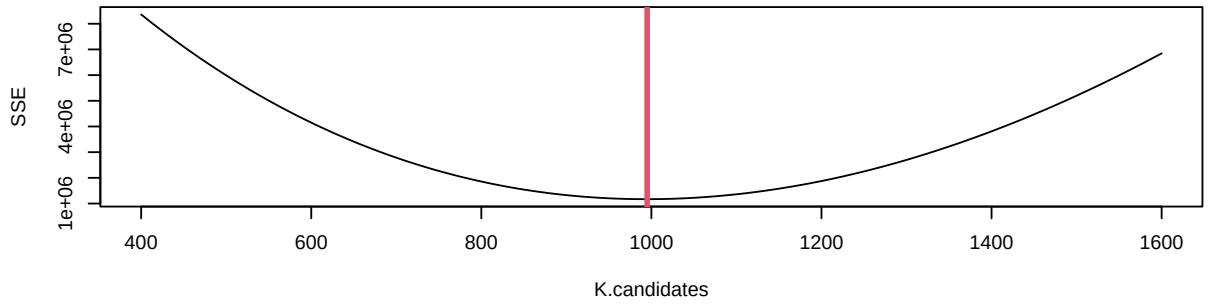
```
## [1] 1169743
```

```r
r.optimal=r.candidates[index[1]]
N1.optimal=N1.candidates[index[2]]
K.optimal=K.candidates[index[3]]

###
### r
###

par(mfrow=c(3,1))
plot(r.candidates,score.save[,index[2],index[3]],type='l',ylab="SSE")
abline(v=r.optimal,col=2,lwd=3)

###
### N1
###
```

```r
plot(N1.candidates,score.save[index[1],,index[3]],type='l',ylab="SSE")
abline(v=N1.optimal,col=2,lwd=3)

###
### K
###

plot(K.candidates,score.save[index[1],index[2],],type='l',ylab="SSE")
abline(v=K.optimal,col=2,lwd=3)
```

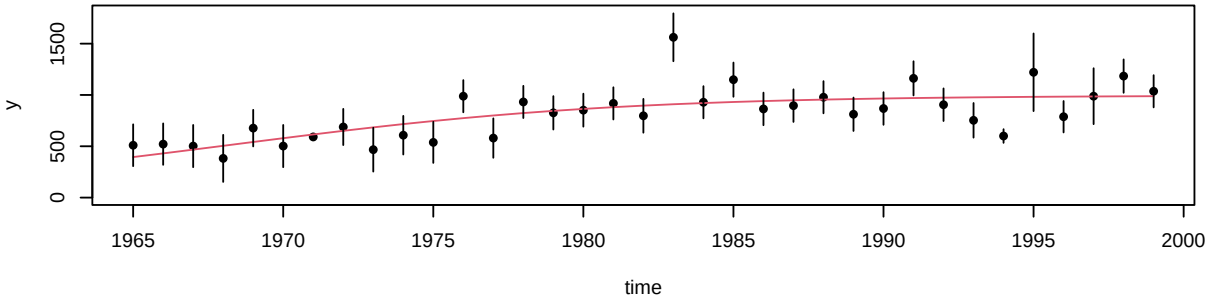



```r
###
### Optimal fit
###
```

```r
plot(time,y,ylim=c(0,1800),pch=16)
y.hat=calc.lg(N1.optimal,r=r.optimal,K=K.optimal,T=35)
lines(time,y.hat,col=2)

se=elk$SE
lb=y-1.96*se
ub=y+1.96*se

###
### Error bars
###

for(i in 1:length(ub)){
  segments(time[i],y[i],x1=time[i],y1=ub[i])
  segments(time[i],y[i],x1=time[i],y1=lb[i])
}
```




This exercise offers a great way for you to practice your knowledge of looping and manipulation of arrays. However, the "brute force" method is almost never used to estimate parameters from data. What are the problems with this approach?

For the motivated and diligent (i.e., all graduate students enrolled in this course), figure out how to write a function that includes vertical error bars for the standard errors. Use a web search or book to give you some ideas. Plot the model, the means, and the SE. This plot should resemble:

# References

May, R. M. 1976. Simple mathematical models with very complicated dynamics. Nature **261**:459–467.