Language in Time - 1 - Rudiments

To begin, let's get some data into R. We will start the manual with what is recognized as a sort of tradition in recurrence quantification of text. In the early 2000's, a demonstration of recurrence applied to text was conducted using Dr. Seuss' Green Eggs and Ham. Its small number of unique words (50) and characteristic weaving of rhyme and repetition make it a useful first demonstration. We uphold this tradition here.

```
setwd('~/Dropbox/new.projects/recurrence.recipes')
options(warn = -1)
library(crqa)
```

```
## Loading required package: Matrix
## Loading required package: tseriesChaos
## Loading required package: deSolve
## Loading required package: fields
## Loading required package: spam
## Loading required package: grid
## Spam version 1.0-1 (2014-09-09) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.
##
## Attaching package: 'spam'
##
## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve
##
## Loading required package: maps
## Loading required package: pracma
##
## Attaching package: 'pracma'
##
## The following object is masked from 'package:deSolve':
##
##     rk4
##
## The following objects are masked from 'package:Matrix':
##
##     expm, lu, tril, triu
```

```
library(tm)
```

```
## Loading required package: NLP
```

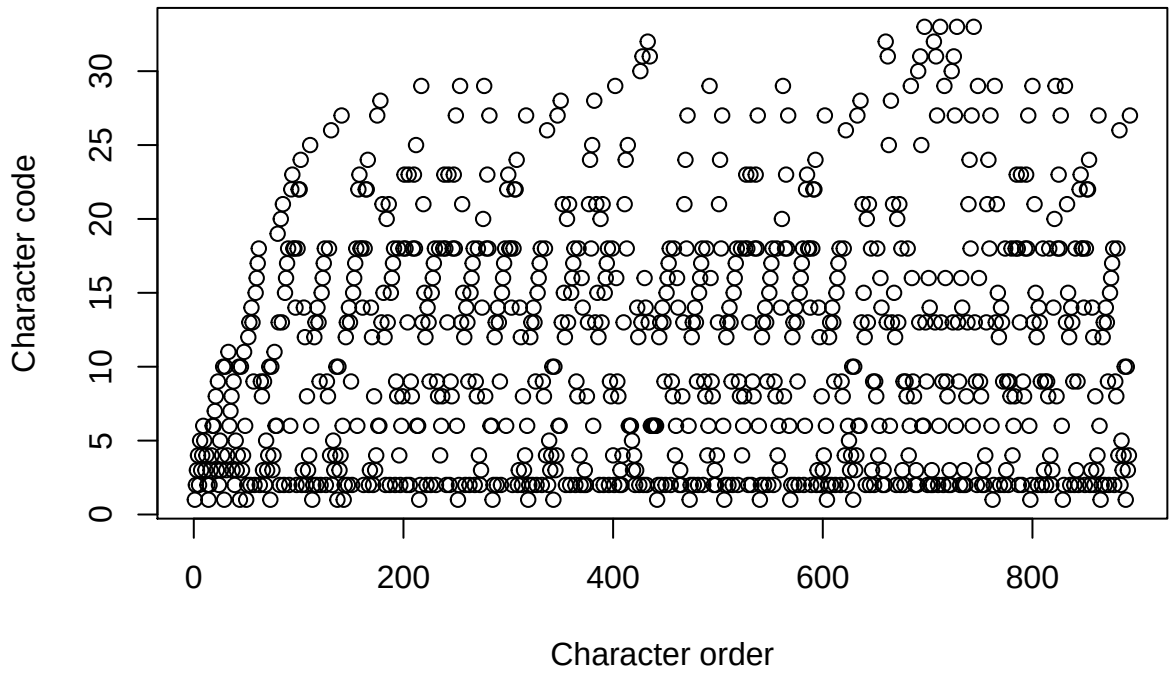
```
rawText = readChar('data/Sam I Am.txt',
  file.info('data/Sam I Am.txt')$size)
```

Once we have these data, let us convert them at the character level.

```
chars = unlist(strsplit(rawText, ""))
uniqChars = unique(chars)
charSeries = as.vector(sapply(chars,function(x) {
  which(x == uniqChars)
}))
```

Our sequence of characters, encoded with unique identifiers.

```
plot(charSeries,type='p',ylab='Character code',xlab='Character order')
```



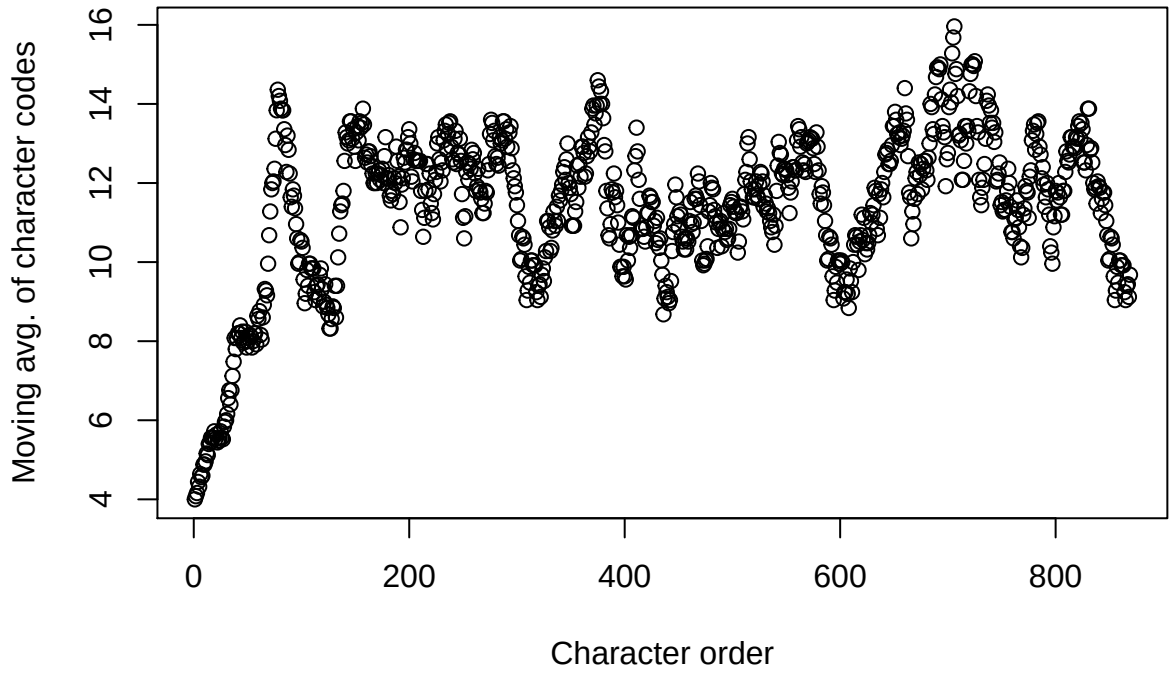
This plot isn't especially meaningful, but it does give us an opportunity to draw our first connection to NLP and corpus linguistics. A well-known regularity referred to as Heaps' Law relates the number of unique words in a document to the length of that document. The characteristic rise and gradual asymptote of the plot you see above reflects this law. Our code above uniquely numbers words in order. Since words are finite, the occurrence of a new identifier becomes less likely as the document is processed. The relationship between document length and the rise of numeric identifiers is $T(n) = an^b$, with a and b parameters determined by observation. Take a look at how the moving average of this function. You can see the average identifier top off pretty quickly, especially for so repetitive and short a text as Green Eggs and Ham!

```
library(zoo)
```

```
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
```

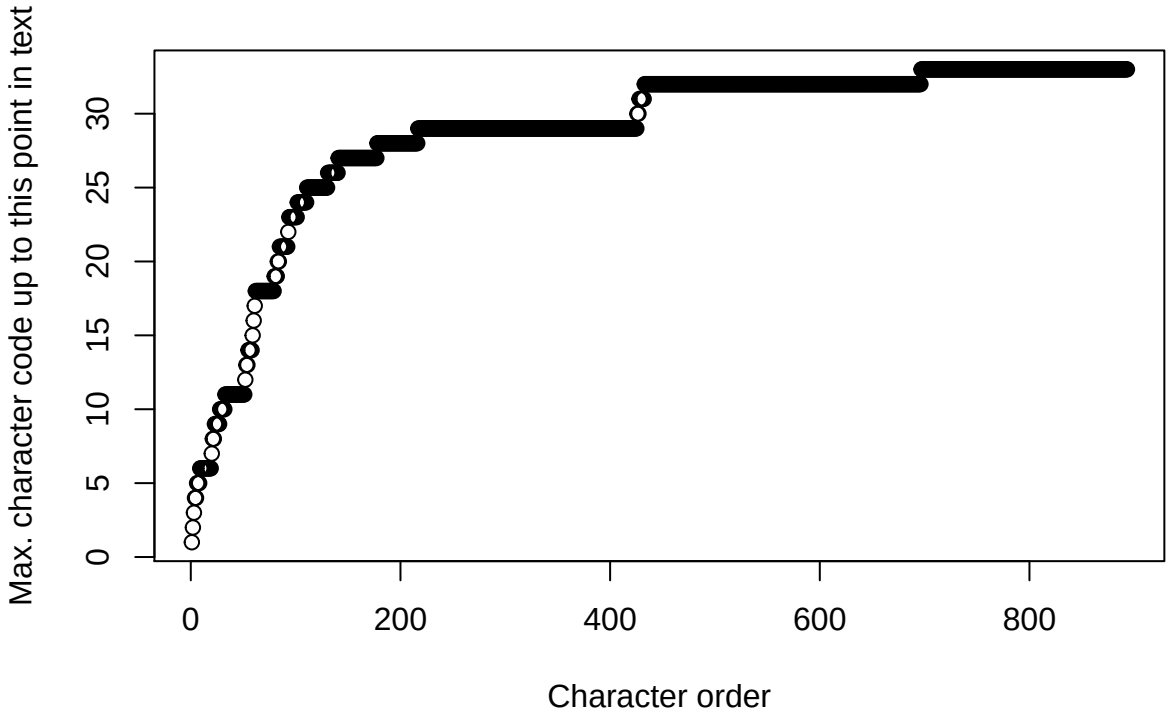
```
##
##     as.Date, as.Date.numeric
```

```
plot(rollmeanr(charSeries,25),type='p',ylab='Moving avg. of character codes',xlab='Character order')
```



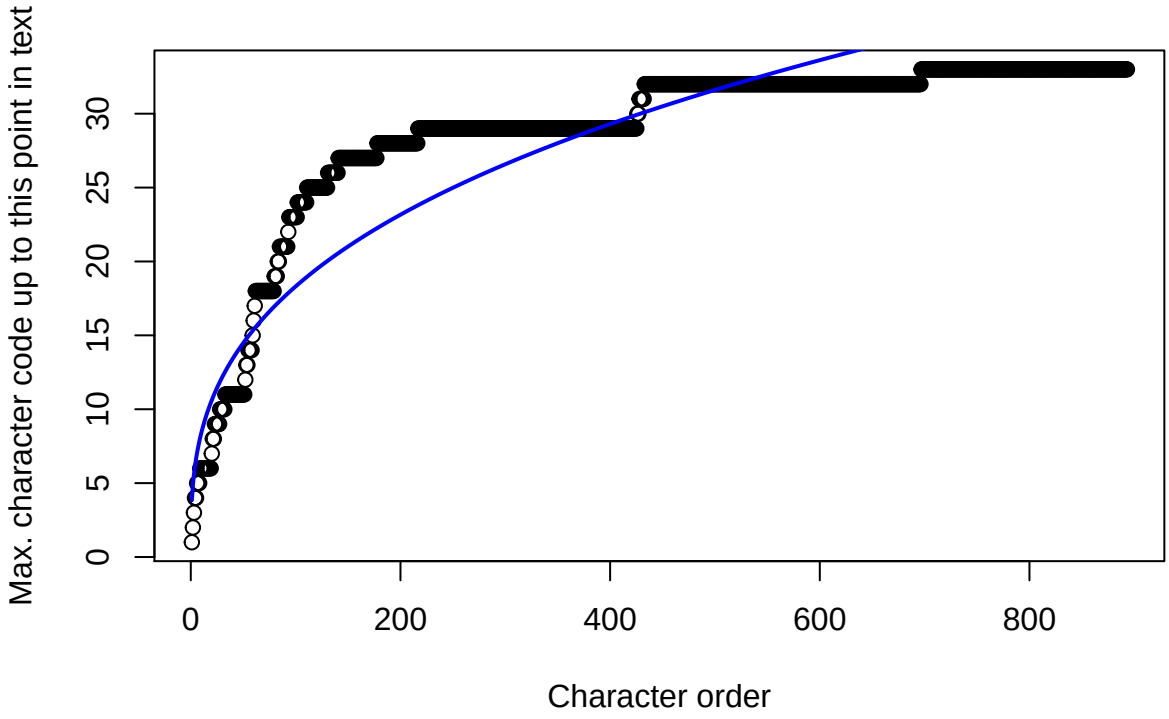
A typical way to display this is to show the maximum identifier value as a function of the length of our consideration. Like this:

```
maxByN = apply(data.frame(1:length(charSeries)),1,function(x) {
  return(max(charSeries[1:x]));
})
plot(maxByN,type='p',ylab='Max. character code up to this point in text',xlab='Character order')
```



We can create a fit to this result pretty easily in R in the following way. Taking the log of each side of the equation above, we get $log(T(n)) = log(a) + b * log(n)$, which is equivalent to a linear function fitting $log(T(n))$ to $log(n)$, with $log(a)$ and b determined by the fit.

```
x = 1:length(maxByN)
linFit = fitted(lm(log(maxByN)~log(x)))
plot(maxByN,type='p',ylab='Max. character code up to this point in text',xlab='Character order')
points(exp(linFit),type='l',col='blue',lwd=2)
```

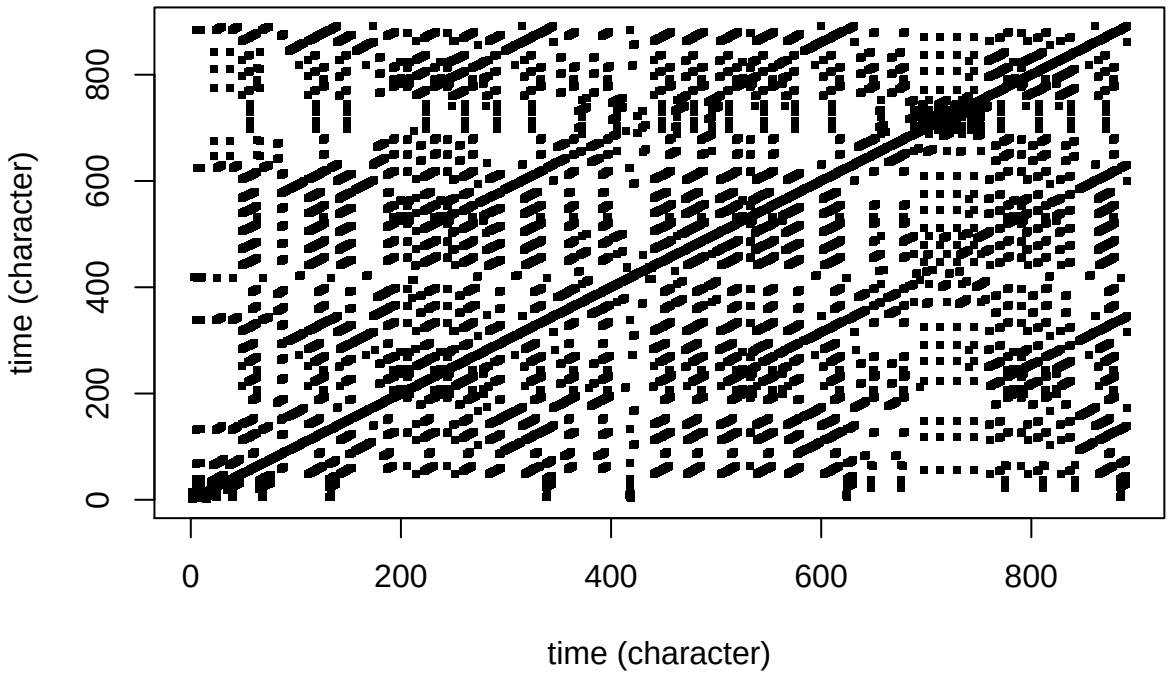


It appears that the text is so short, and lexically simple, that it may be underfit by the basic Heaps approach, though the coefficient is in the range expected by other English corpora of 0.5 or so (see the result of the lm function). This sort of thing has been discussed in various places about the Heaps relationship in different kinds of text (e.g., Kubo, 2010).

In any case, note that we convert characters to a series of nominal codes. These numeric codes represent the character types, and converting the characters to numeric codes allows us to utilize all the standard `crqa` library functions for reasons that will become quite clear below.

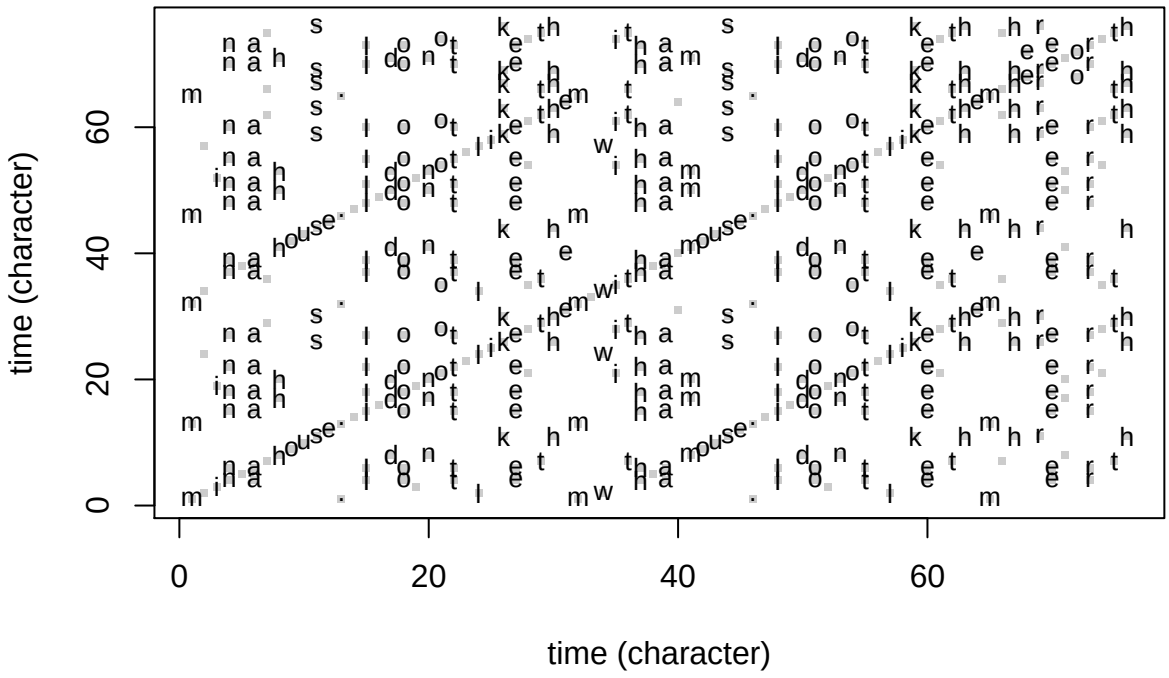
The core data structure that underlies all recurrence quantification is the recurrence plot (RP). Let's define it with respect to our character time series. A recurrence plot (RP) is the set of points (i,j) such that `charSeries[i]==charSeries[j]`. Building it with the `crqa` library is quite easy.

```
crqaResults  = crqa(charSeries,charSeries,1,3,1,.0001,F,2,2,0,F,F) # we'll explain parameters later
RP = as.matrix(crqaResults$RP) # convert into numeric non-sparse matrix
ij = which(RP==1,arr.ind=T) # get coordinates
plot(ij[,1],ij[,2],
     xlab='time (character)',ylab='time (character)',pch=15,cex=.5)
```



So the RP is just a collection of points representing the instances at which the "system" (here, Theodor Geisel, I guess) is revisiting particular states that we are interested in (in this case, characters). To get a sense of how this works, let's grab the especially repetitive sequencing taking place about midway into Green Eggs and Ham.

```
crqaResults  = crqa(charSeries[460:535],charSeries[460:535],1,1,1,.0001,F,2,2,0,F,F) # we'll explain pa
RP = as.matrix(crqaResults$RP) # convert into numeric non-sparse matrix
ij = which(RP==1,arr.ind=T) # get coordinates
plot(ij[,1],ij[,2],
     xlab='time (character)',ylab='time (character)',
     pch=15,cex=.5,col=rgb(.8,.8,.8))
text(ij[,1],ij[,2],chars[ij[,1]+458],cex=.8)
```

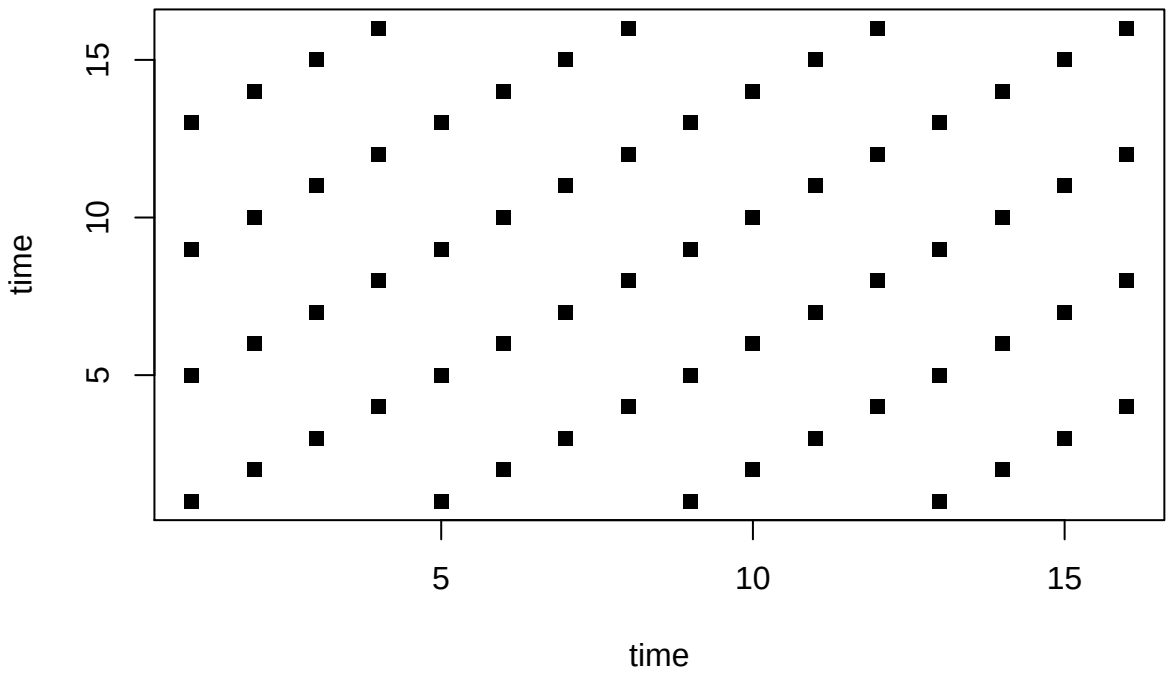


This shows you the specific character sequences that are being revisited by the story (note: grayed points represent spaces). The RP represents these patterns that are repeated as diagonal lines on the plot. Single points would represent single letters being repeated in an isolated fashion. Naturally, there will be a long line up the middle of this plot because by definition 'charSeries[i]==charSeries[i]', and so we have what is called the "line of incidence." This will become crucial letter, but in this initial demonstration of "auto-recurrence" – which means we're building an RP for just one time series compared to itself – it is trivially filled with points. The more interest points are the off-diagonal lines that show repetition of the system's character states. In the plot above, you can see a prominent "I do not like them" on either side of the plot. Of course, the story is filled with such repetition. A well-known application of this visualization was conducted by Orsucci et al. (2001), where this poem along with other from other languages are compared.

An RP is also symmetrical, for obvious reasons, because if `charSeries[i]==charSeries[j]` then `charSeries[j]==charSeries[i]`, and so we get these elegant looking Rorschach kind of symmetries. The symmetry will break down when we move beyond recurrence with a single time series (auto-recurrence) and plot the recurrence between two different time series (cross recurrence).

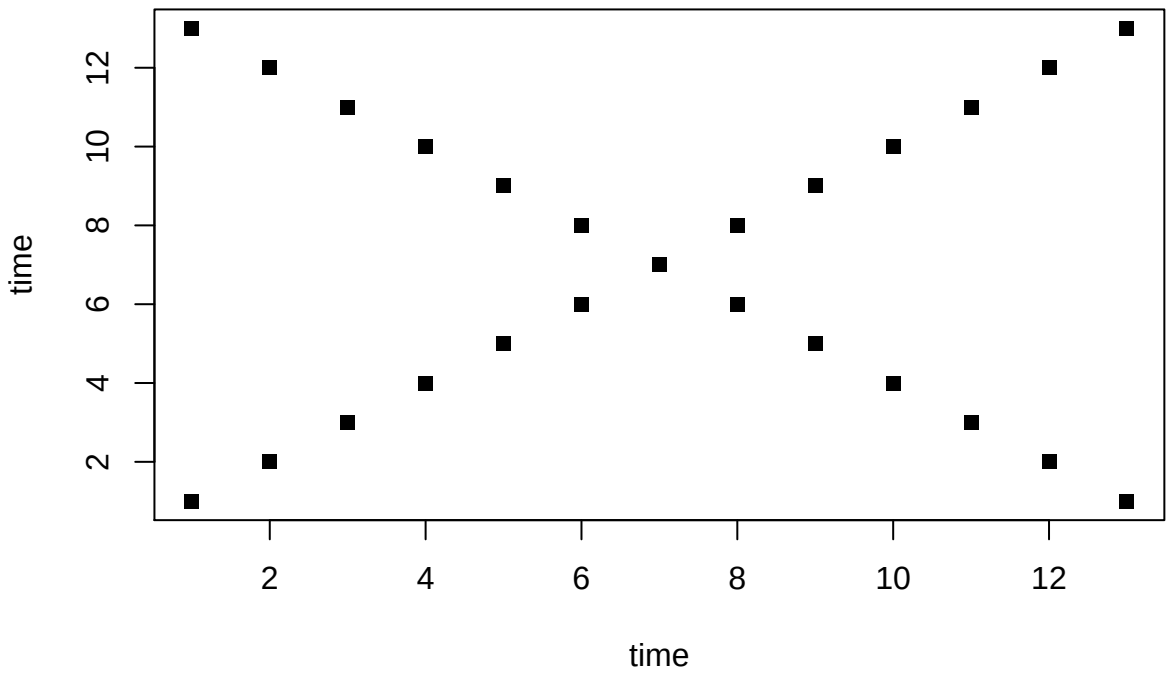
Let's take a quick look at the RP via much shorter time series to build your intuition about the "textures" that emerge on them. To start, let's build a plot that consists of a series of diagonal lines. These would reflect a system that is repeatedly revisiting a prior sequence at various times.

```
charSeries = c(1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4)
crqaResults  = crqa(charSeries,charSeries,1,1,1,.0001,F,2,2,0,F,F) # we'll explain parameters later
RP = as.matrix(crqaResults$RP) # convert into numeric non-sparse matrix
ij = which(RP==1,arr.ind=T) # get coordinates
plot(ij[,1],ij[,2],
     xlab='time',ylab='time',pch=15,cex=1)
```



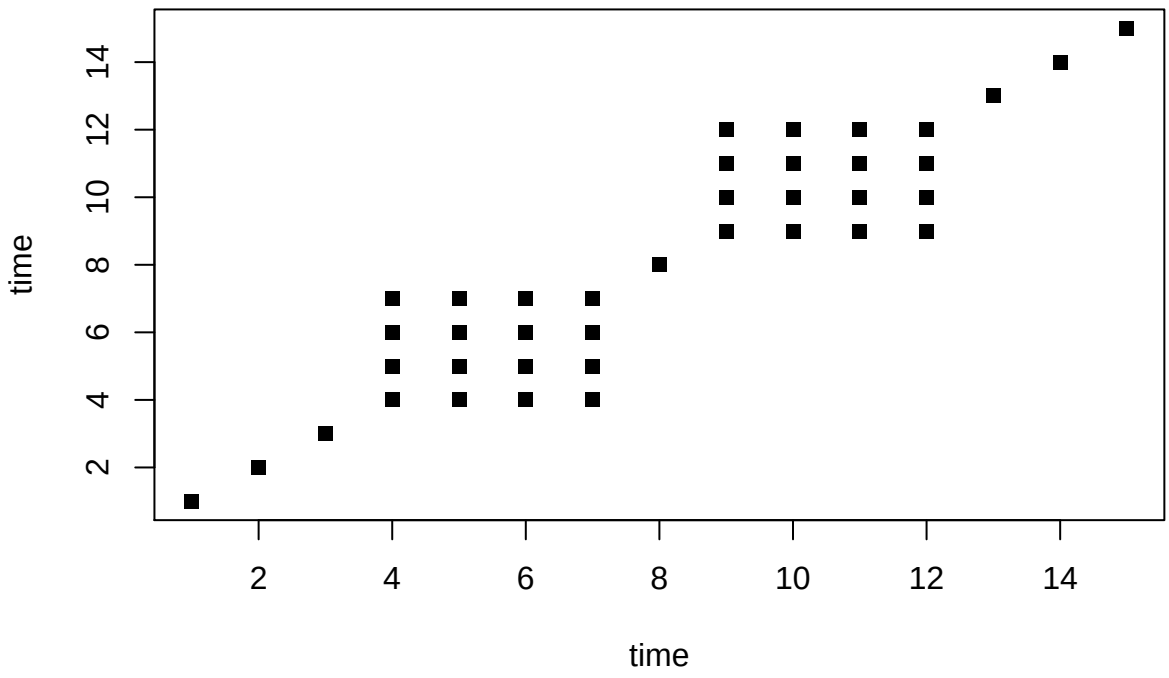
If we have "palindromic" patterns – a system that is reversing direction in its dynamics – this shows up as perpendicular lines in the plot. A palindrome itself looks like this:

```
charSeries = c(1,2,3,4,5,6,7,6,5,4,3,2,1)
crqaResults  = crqa(charSeries,charSeries,1,1,1,.0001,F,2,2,0,F,F) # we'll explain parameters later
RP = as.matrix(crqaResults$RP) # convert into numeric non-sparse matrix
ij = which(RP==1,arr.ind=T) # get coordinates
plot(ij[,1],ij[,2],
     xlab='time',ylab='time',pch=15,cex=1)
```



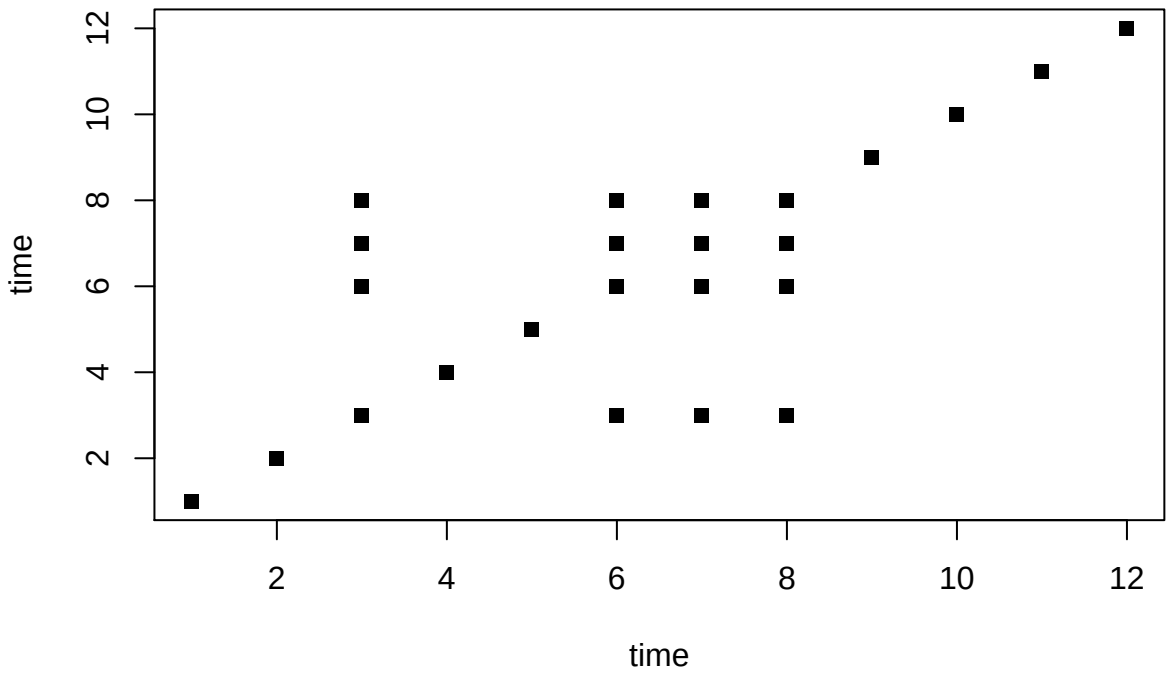
Plots that have "blocks" or wide swaths of occupied points reflect a system which is "trapped" in one state. This may reflect a stable state of the system that is attractor-like.

```
charSeries = c(1,2,3,4,4,4,4,5,6,6,6,6,7,8,9)
crqaResults  = crqa(charSeries,charSeries,1,1,1,.0001,F,2,2,0,F,F) # we'll explain parameters later
RP = as.matrix(crqaResults$RP) # convert into numeric non-sparse matrix
ij = which(RP==1,arr.ind=T) # get coordinates
plot(ij[,1],ij[,2],
     xlab='time',ylab='time',pch=15,cex=1)
```



Horizontal and vertical lines connote the same sort of thing: The system is, for a time, stable in some sequence of the same state. Lines occur because the system, relative to another point in time, has "moved on" from that particular state.

```
charSeries = c(1,2,3,4,5,3,3,3,6,7,8,9)
crqaResults  = crqa(charSeries,charSeries,1,1,1,.0001,F,2,2,0,F,F) # we'll explain parameters later
RP = as.matrix(crqaResults$RP) # convert into numeric non-sparse matrix
ij = which(RP==1,arr.ind=T) # get coordinates
plot(ij[,1],ij[,2],
     xlab='time',ylab='time',pch=15,cex=1)
```


time
2
4
6
8
10
12
time
2
4
6
8
10
12