Noah L. Schrick
1492657

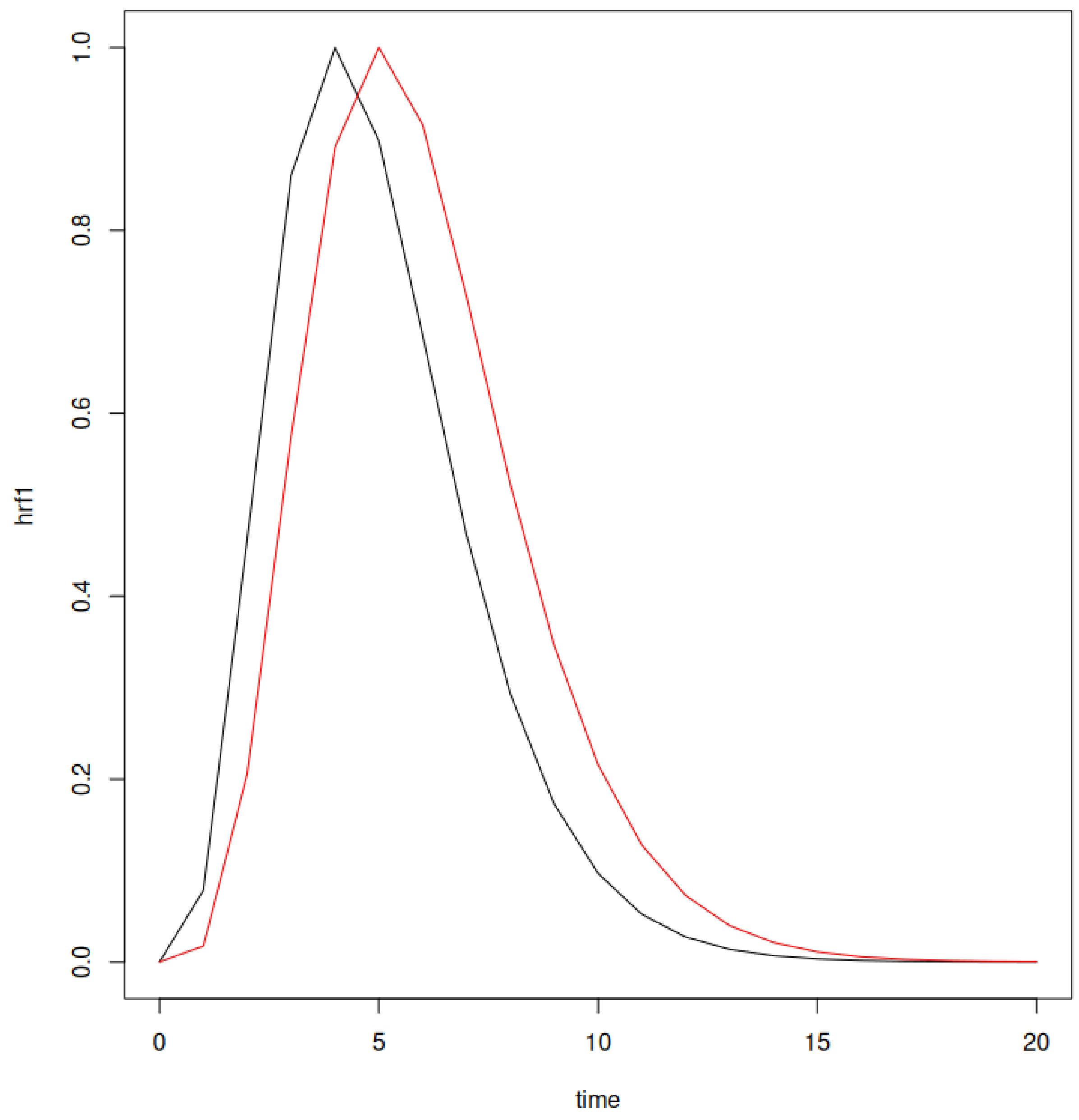
Introduction to fMRI analysis and ICA. A functional MRI (fMRI) scanner measures brain activity as a function of time. The quantity measured is called the blood oxygenation level dependent (BOLD) signal. BOLD is correlated with the delayed firing of a population of neurons in a brain region. The BOLD signals can be spatially mapped and temporally tracked. The two main types of fMRI experiments are task based and resting state. **A.** Task-based experiments look for brain regions whose BOLD time series are correlated with an external stimulus, often an image shown for blocks of time. **B.** Resting-state experiments look for brain regions whose BOLD time series are correlated with each other while the subject is resting (not sleeping but not stimulated by any task). From these correlations, resting-state networks can be constructed and analyzed.

A. Haemodynamic response functions and block design.

Plot the following very basic haemodynamic response function (HRF) from 0 to 20 seconds with parameters q=4 and q=5. This HRF models the BOLD response after a firing of a population of neurons. **1.** Show the two curves overlaid. What is the effect of q?

```
hq <- function(t,q=4){
      # q=4 or 5, where 5 has more of a delay
      return (t^q * exp(-t)/(q^q * exp(-q)))
}

# use seq to create vector time and use hq to create hrf vectors
time <- seq(0,20)

# plot
plot(time,hrf1,type="l")
lines(time,hrf2,col="red")
```

1. “q” acts as a shift. A larger q value delays the function.

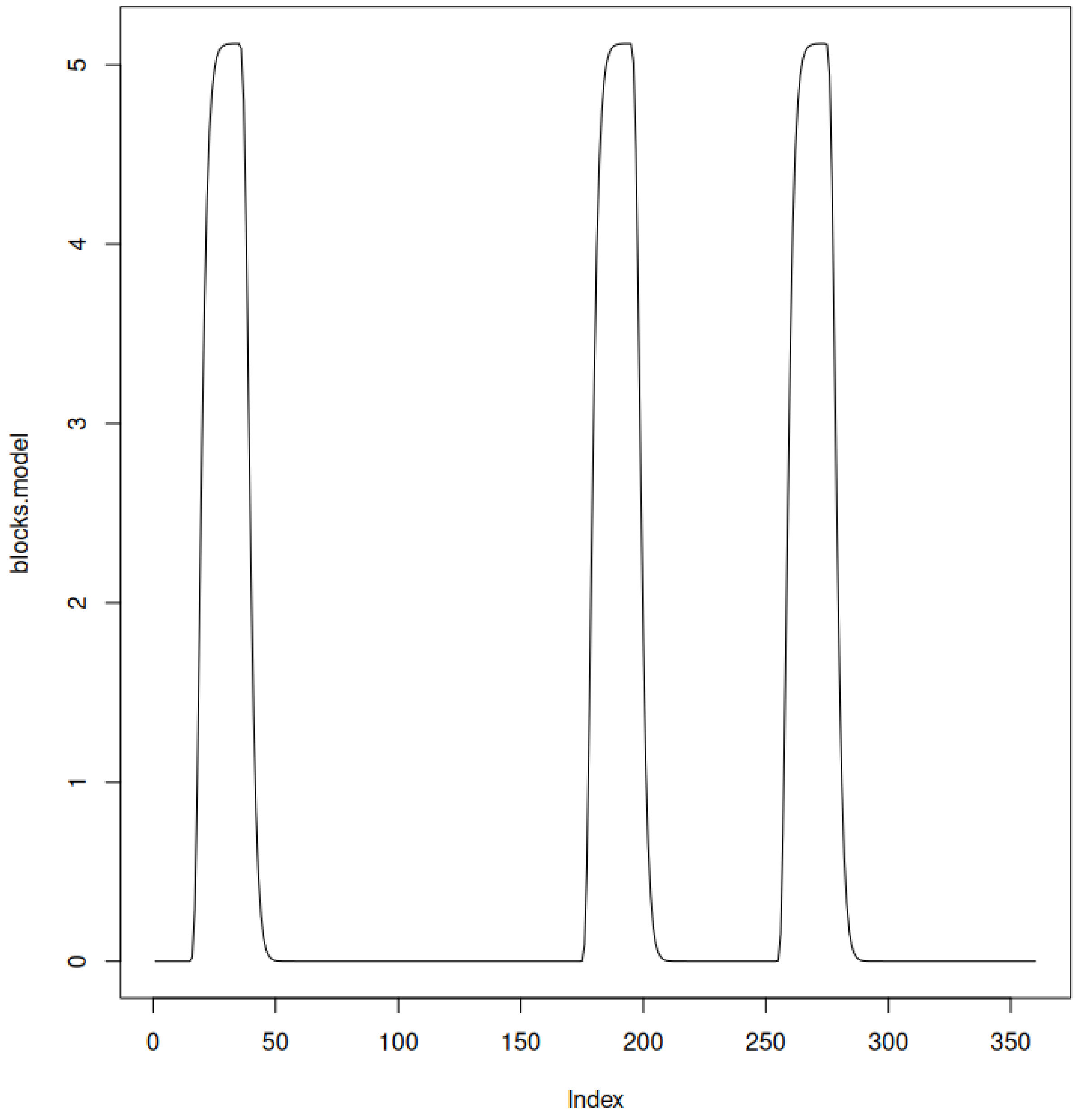


This more complex HRF comes from the `basis_block_hrf4` function in `3dDeconvolve.c` from the AFNI (analysis of functional neuroimage) software maintained by the NIH. Use the code below to "deconvolve" or mix the HRF with the known task onset times (`onsets`), which last for 20 seconds each block. **2.** Show the `blocks.model` plot. What does this time-series vector represent?

```
# grabbed from afni c code
# basis_block_hrf4 from 3dDeconvolve.c
```

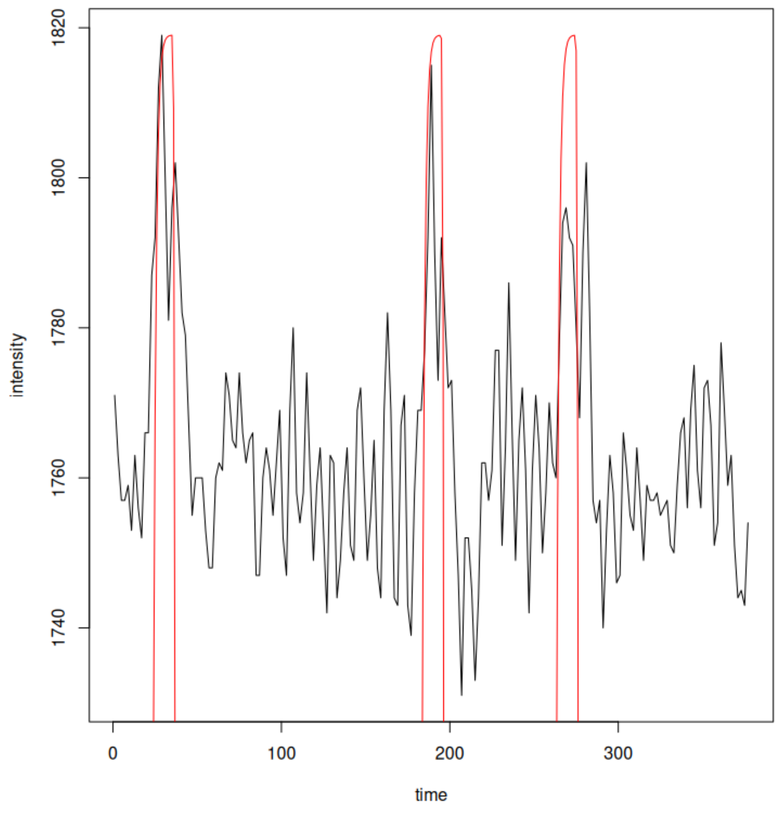
```
HRF <- function(t, d){
      if (t<0){
            y=0.0
      }else{
            y = 1/256*exp(4-t)*(-24-24*t-12*t^2-4*t^3-t^4 +
exp(min(d,t))*(24+24*(t-min(d,t)) + 12*(t-min(d,t))^2+4*(t-min(d,t))^3+(t-
min(d,t))^4))
      }
      return(y)
}

t=seq(0,360,len=360)
onsets=c(14,174,254)
blocks.model = double()
for (curr_t in t){
      summed_hrf=0.0
      for (start in onsets){
            summed_hrf=summed_hrf+HRF(curr_t-start,20)
      }
      blocks.model = c(blocks.model,summed_hrf)
}

plot(blocks.model,type="l")
```

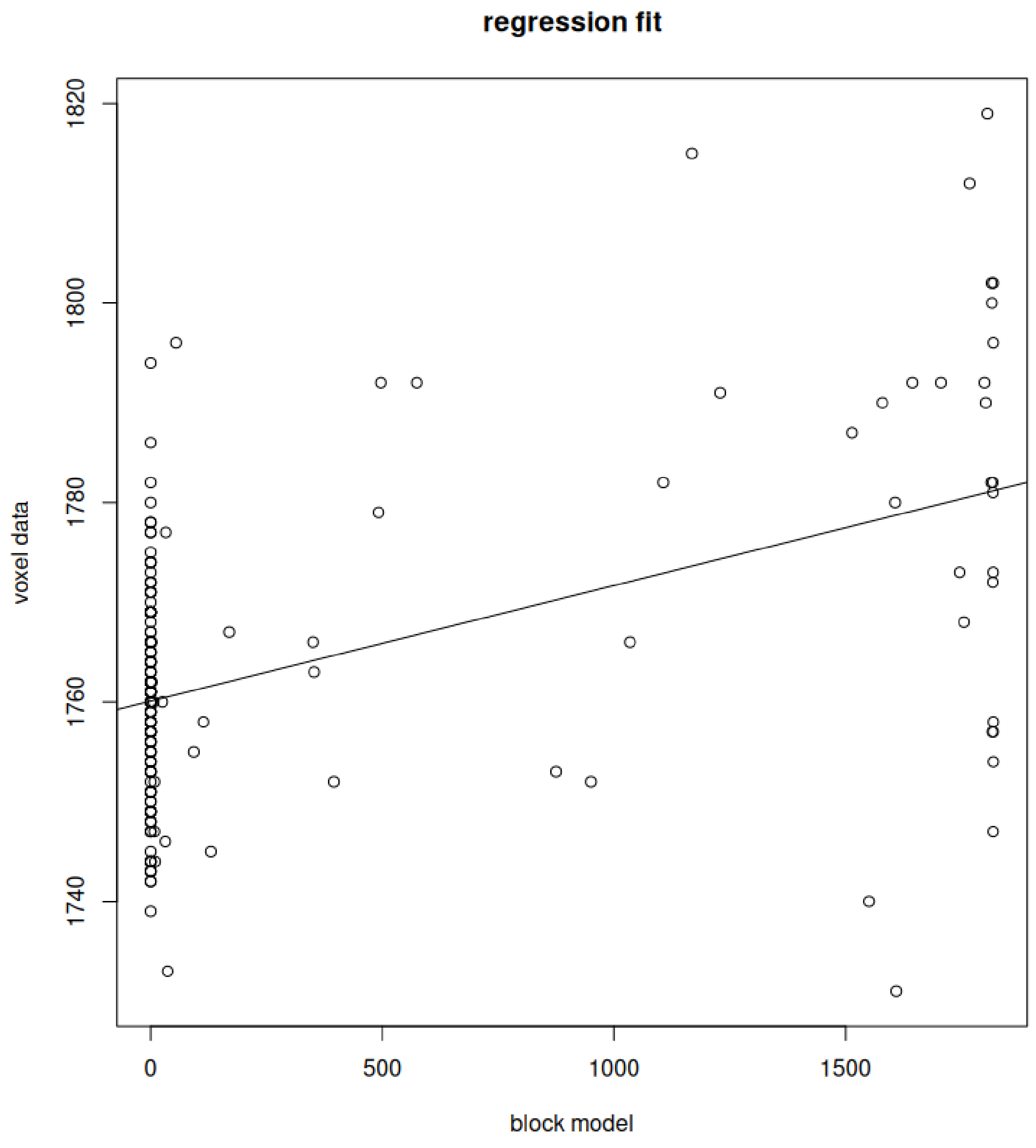
2. This represents the original signal of the times (in seconds) when the task onsets occurred.



Set the working directory to the location of the 1D file, which is a time-series profile for one voxel in one subject in the task-based experiment. **3.** Plot the voxel time series data and the block design curve. Use `lm` to determine how well the data fit the block model (`voxel.data.vec` will represent the outcome variable and `blocks.norm.subset` will represent the predictor variable). **4.** Show the statistical results and add a fit line to the voxel/model scatter plot using `abline`. The slope and intercept of the fit line come from the `lm` coefficients. **5.** In the last plot, why are there a lot of points bunched up near 0 on the horizontal axis?

```
voxel.data <- read.delim("059_069_025.1D")
plot(seq(1,2*dim(voxel.data)[1],by=2),t(voxel.data),
     type="l", xlab="time",ylab="intensity")
# normalize the height of the blocks model
blocks.normal <- max(voxel.data)*blocks.model/max(blocks.model)
lines(blocks.normal,type="l",col="red")

# regression
length(blocks.normal) # too long
dim(voxel.data)[1]
# grab elements from blocks.normal to make a vector same as data
blocks.norm.subset <-
blocks.normal[seq(1,length(blocks.normal),len=dim(voxel.data)[1])]
length(blocks.norm.subset)

voxel.data.vec <- matrix(unlist(voxel.data),ncol=1)

# use lm to create voxel.fit <- lm…
voxel.fit <- lm(voxel.data.vec ~ blocks.norm.subset)

plot(blocks.norm.subset,voxel.data.vec,
     xlab="block model",ylab="voxel data",main="regression fit")

# use abline(voxel.fit) to overlay a line with fit coefficients
abline(voxel.fit)
```





Call:

```
lm(formula = voxel.data.vec ~ blocks.norm.subset)
Coefficients:
      (Intercept)  blocks.norm.subset
        1.764e+03         -7.599e-03
```

5. The underlying blocks.model has many values less than 1:

```
> sum(blocks.model < 1)
[1] 290
> 100* sum(blocks.model < 1)/length(blocks.model)
```

[1] 80.56% **values less than 1**

B. Resting state fMRI visualization, multidimensional arrays and independent component analysis (ICA).

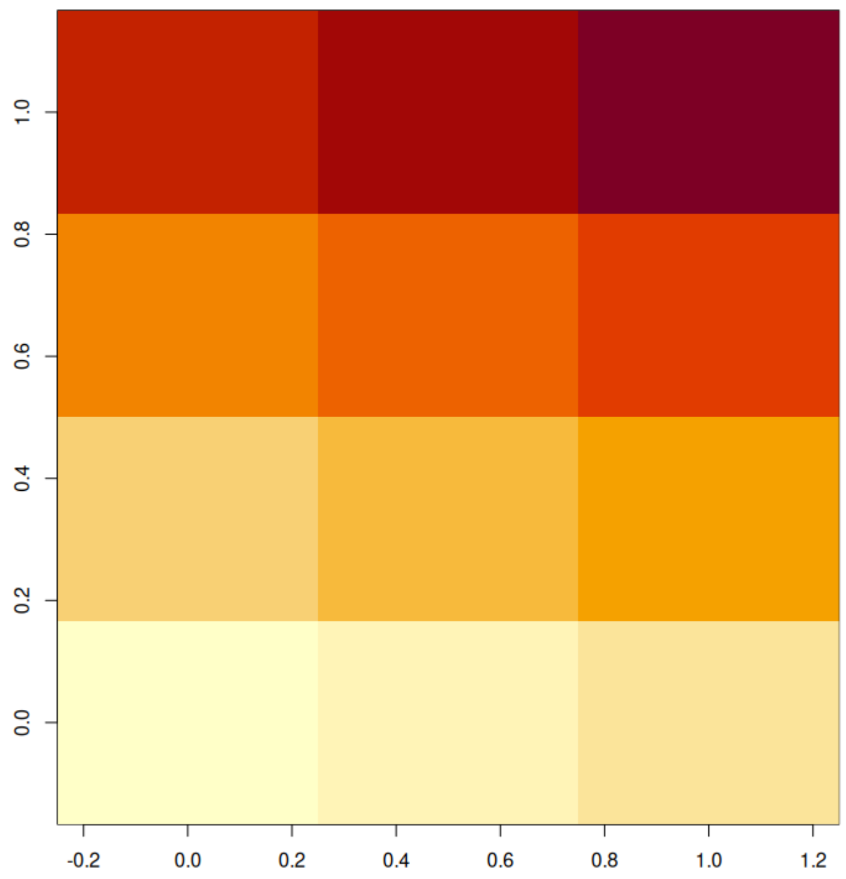
Multi-dimensional arrays. Typically we deal with at most 2d arrays (matrices), but now we are dealing with 4d arrays (3 space plus time). To illustrate how multi-dimensional arrays work in R, study the code below. The `array` command creates a multi-dimensional array called `multArray` from the numbers 1-24. **6.** How many dimensions does it have and what are the sizes of the dimensions? Show the numbers in the last two slices and use image to plot a heat map of the values. **7.** What data type are these slices in terms of arrays? When `multArray` was created from the numbers 1-24, what order of the dimensions was used to store the numbers?

```
## example: multi-dimensional array
#                        2 3x4 matrices
multArray <- array(1:24,dim=c(3,4,2))
dim(multArray)
multArray[,,1] # matrix 1
multArray[,,2] # matrix 2
image(multArray[,,1]) # plot slice 1
image(multArray[,,2]) # plot slice 2
```

6. 3 dimensions. “x” = 3 “y” = 4, “z” = 2

```
> multArray[,,1] # matrix 1
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> multArray[,,2] # matrix 2
     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

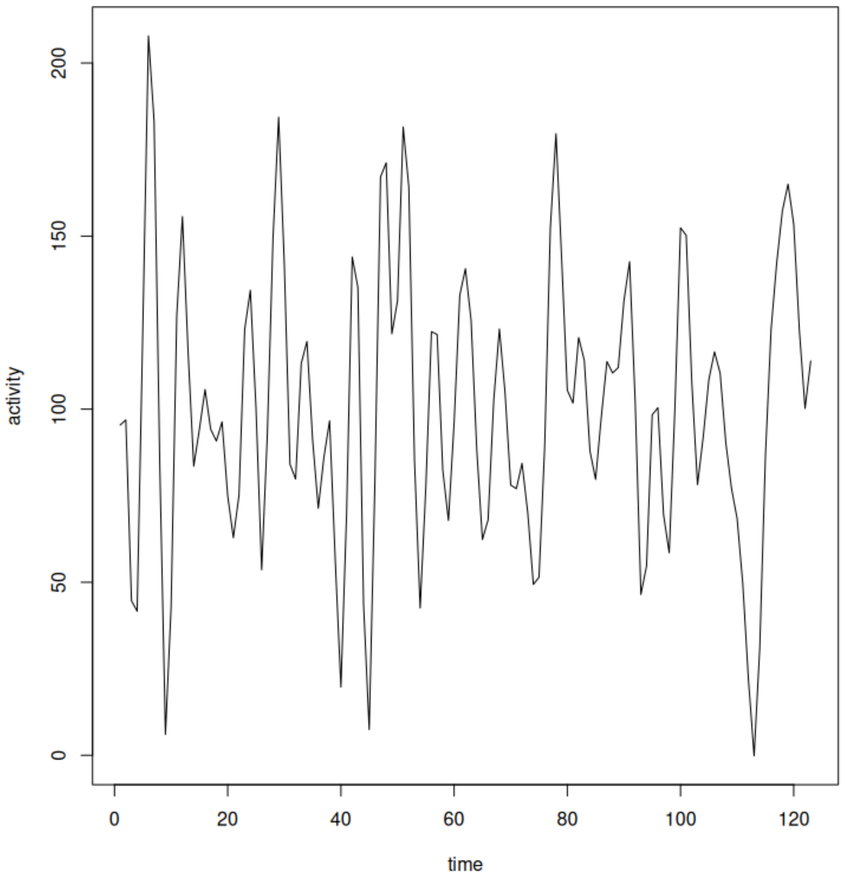
7. Started with the first slice and went down the first column, down the second column, down the third column, and then down the fourth column. This was then repeated for the second slice.



Install and load the following libraries. Download and load the nii file from Harvey. A nii file is a nifti (**N**euroimaging **I**n**F**ormatics **T**echnology **I**nitiative) file format. The attached nii file contains multiple cubes of data for a single subject, where each cube is a snapshot of the brain's BOLD activity. Some of the data in the file is outside the brain (skull, etc). The `mask` variable allows us to extract only the brain voxels. **8.** What is the size of the `ttt` data variable? What do the dimensions represent? How many voxels are there? Show the time-series for the voxel with xyz indices [30,30,30].

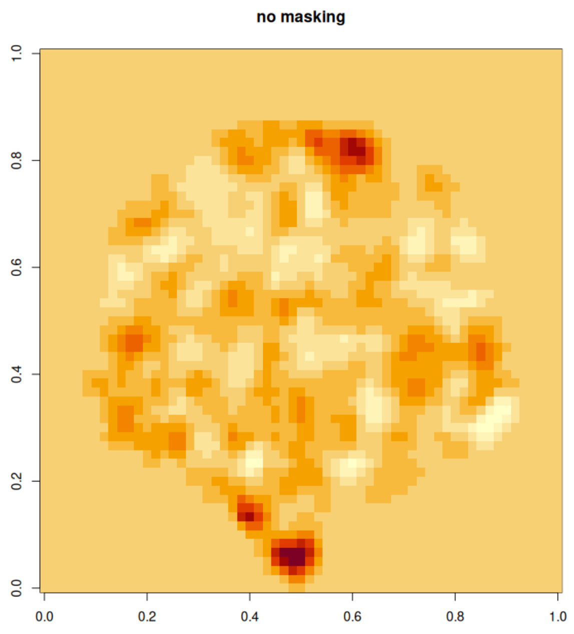
```
install.packages("fastICA")
#install.packages("AnalyzeFMRI")
#  "fmri" install requires Rtools for Windows (install outside of RStudio)
#   install.packages will prompt you to install gsl, say yes
install.packages("fmri")
#    If this fails, try library(devtools) (install.packages("devtools"))
#    and then install_github(https://github.com/cran/fmri.git)
library(fmri)
#    If fmri still doesn't work, ask me about dataMat.RData
library(AnalyzeFMRI)
library(fastICA)

# read in 4d nifti
# uses fmri library, takes about 3min to load
img <- read.NIFTI("rest_res2standard.nii")

mask <- img$mask  # Boolean mask for brain voxels
dim(mask)
ttt <- extractData(img) # extract 4d data cube
numScans <- dim(ttt)[4]

# plot a voxel's time series
plot(ttt[30,30,30,],type="l",xlab="time",ylab="activity")
```

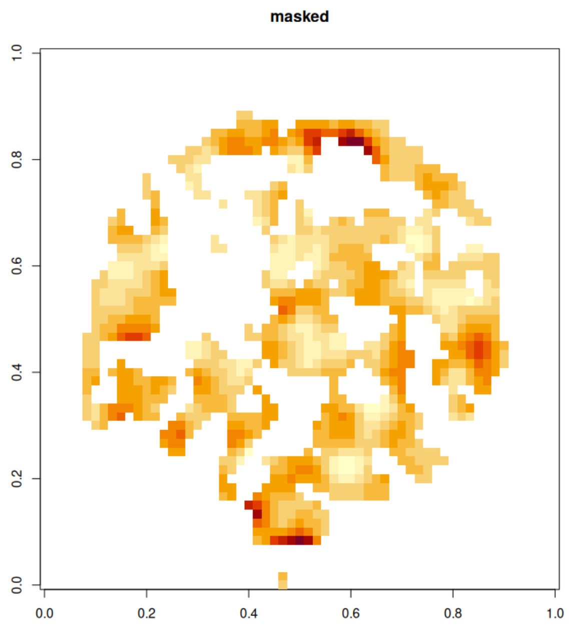
8. > dim(ttt)
[1] 61 73 61 123
> length(ttt)
[1] 33410859
Dimensions: x,y,z, and time
123 voxels



Plot a 2d slice. Next we plot a 2d slice (the plane in x-z when y=35) of the brain activity at a time point (t=50). Show the two images. What was used in the second one to make it look different?

```
yslice <- 35
scan2dslice <- ttt[,yslice,,50] # grab 2d slice at t=50
image(scan2dslice,main="no masking") # no mask
# mask it off
slice.mask <- mask[,yslice,]
scan2dslice[slice.mask] <- NA # NA's become white
image(scan2dslice,main="masked")
```





The boolean mask for brain voxels for this specific slice (y=35) was obtained, and the overlap of it and the scan2dslice variable was filled with NAs to result in white areas on the image.

Independent Component Analysis. ICA has recently become a popular way to extract interesting features from resting-state fMRI data. It is similar to principle component analysis (PCA) or multidimensional scaling (MDS). One of the goals of ICA for fMRI is to find sets of voxels that are correlated with each other in time. Each one of these sets is an ICA component and sometimes corresponds to a functional neural network, like the motor system, the visual system or the default mode network. We will use an implementation of ICA, called fastICA, that requires the data to be in a 2d (matrix) format. **9.** The following transforms the 4d data into 2d. Based on the size of `dataMat`, what does each column represent?

```
t1 <- Sys.time() # for timing purposes
dataMat <- NULL
for(t in seq(1,numScans)){
      scan <- ttt[,,,t]
      # stretched out the 61x73x61 3d matrix into one row
      # apply mask and stack
      dataMat <- rbind(dataMat,scan[mask])
}
t2 <- Sys.time()
difftime(t2,t1) # 3 minutes
dim(dataMat)
```

9.

> dim(dataMat)

[1]   123 226920

rows: voxels, columns: time

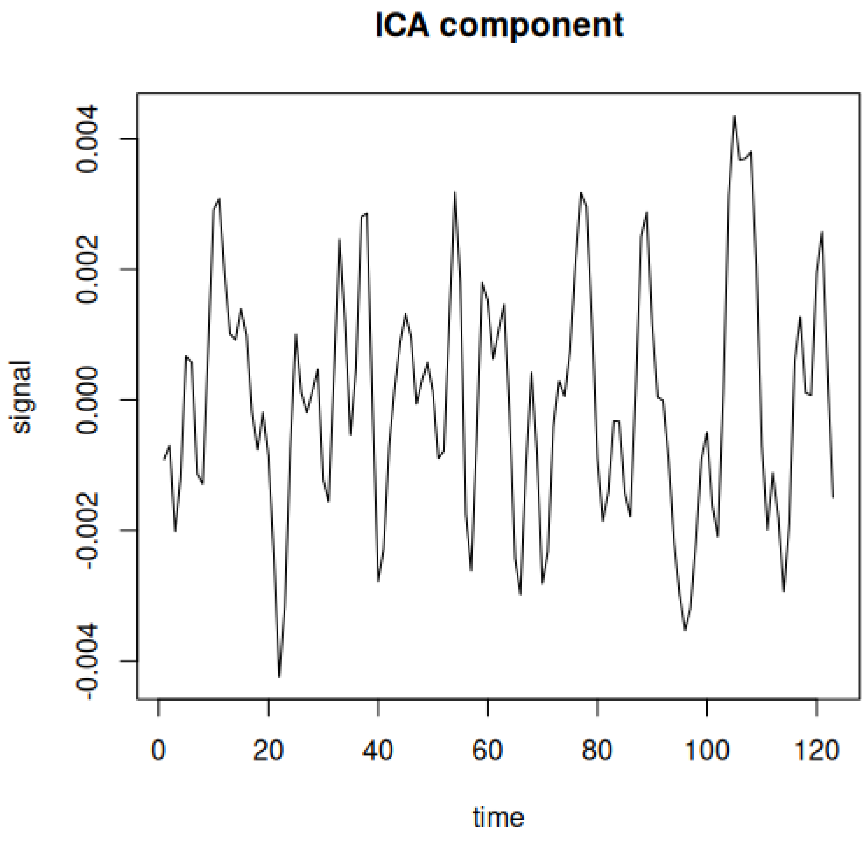
Given the data matrix X and a chosen number of components, m, the goal of ICA is to find a matrix, S, whose m columns are independent in a linear algebra sense. If you new the independent sources, S, and how they were mixed together with matrix A, then you could predict X = SA. ICA determines S, K and W, which satisfy S = XKW. Run the ICA code below for 20 components. **10.** What are the dimensions of S, K and W?

```
# ICA analysis
# input X: rows observations (voxels) and cols variables (time)
X <- t(dataMat)
m <- 20 # specify number of ICA components
t1<-Sys.time()
f<-fastICA(X,n.comp=m,method="C")
t2<-Sys.time()
difftime(t2,t1) # 1.23min
# S=XKW,
# K is a pre-whitening PCA matrix (components by time)
# S is the matrix of m ICAs (columns of S are spatial signals)
# S has dimensions voxel x components
S<-f$S  # you can find K and W with $
```

10. S=226920   20,   K=123   20,   W=20   20

K contains the time representations of the m ICA components. **11.** Plot the 5th component time series with the code below. The voxel locations corresponding to this signal share this time series profile.

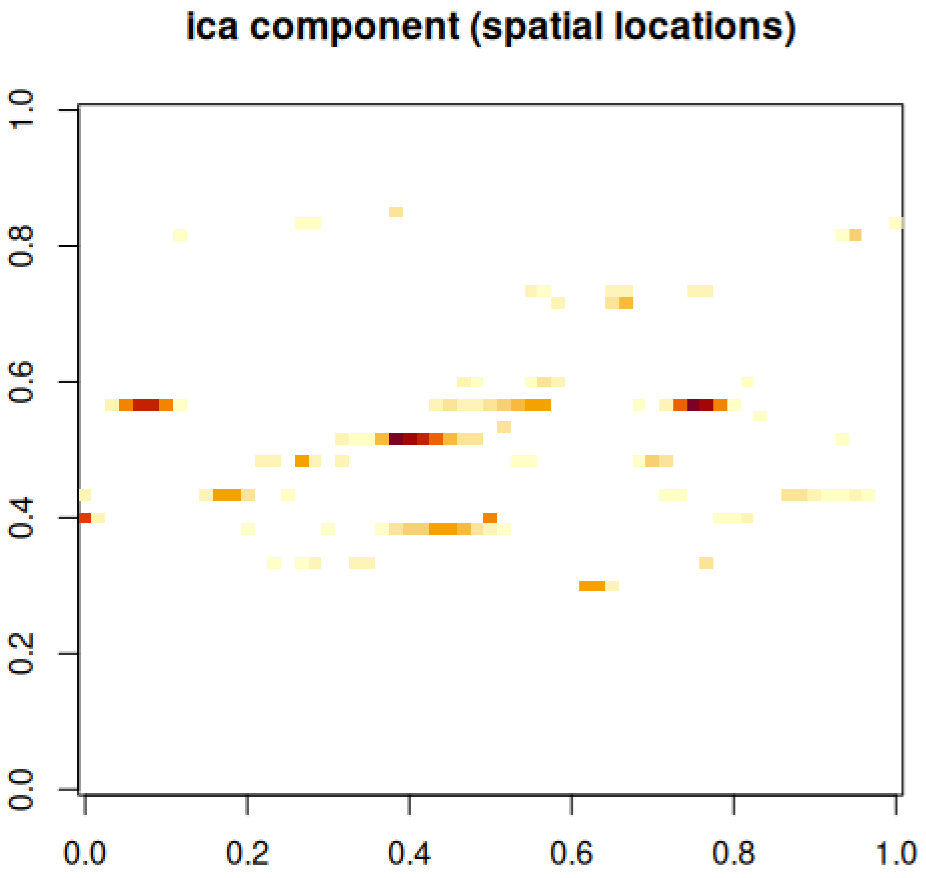
```
ica.comp <- 5 # look at 5th component
# plot the 5th time ICA
plot(f$K[,ica.comp],type="l",xlab="time",ylab="signal",main="ICA component")
```



S contains the weights for each ICA component in the voxel space. **12.** Use the code below to plot the weights of the 5th ICA component in one brain slice. More sophisticated plotting can be achieved (with structural brain templates) by exporting to nii or afni formats and opening in the AFNI or fsl software packages (linux). Since we are constrained to Windows in the lab, we will stop here with visualization.

```
# threshold S matrix
theta <- 2
S[S<=theta] <- NA

# turn S back into 4d multidimensional array
xdim<-dim(ttt)[1]
ydim<-dim(ttt)[2]
zdim<-dim(ttt)[3]
```

```
ica.4dArray <- array(matrix(S,ncol=1),dim=c(xdim,ydim,zdim,m))
dim(ica.4dArray) # 61x73x61x10
yslice <- 35
# grab 2d slice at y=yslice and ica 5
ica2dslice <- ica.4dArray[,yslice,,ica.comp]
image(ica2dslice,main="ica component (spatial locations)")
```


ica component (spatial locations)