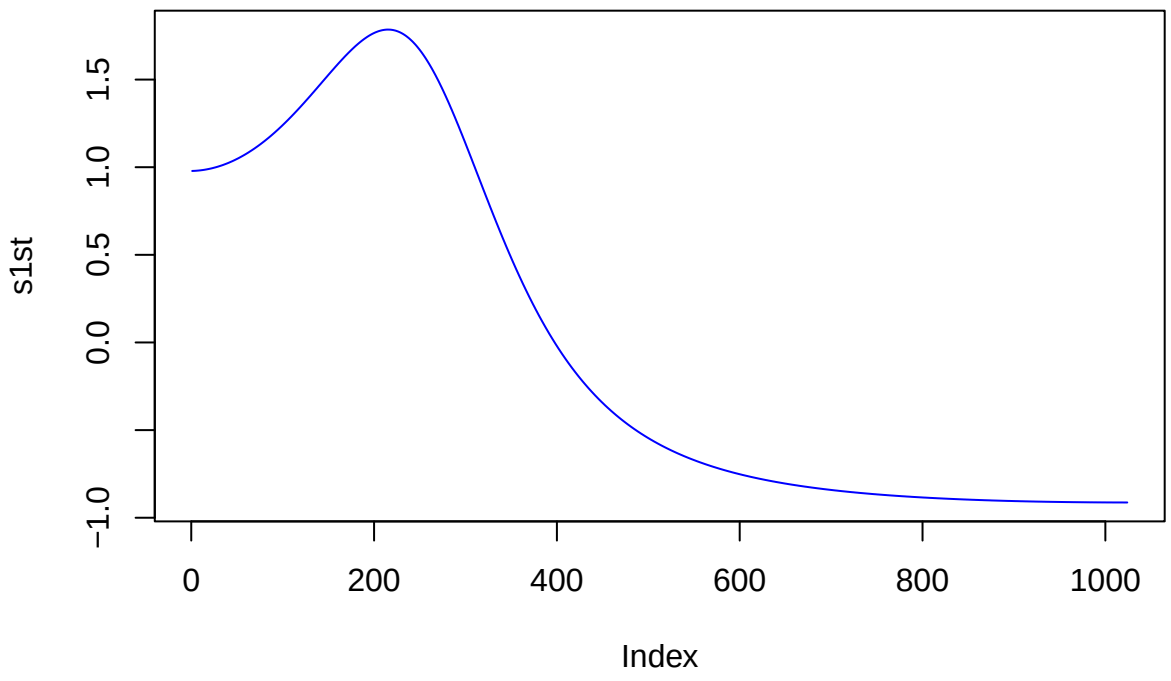
gump2 example

2025-06-19

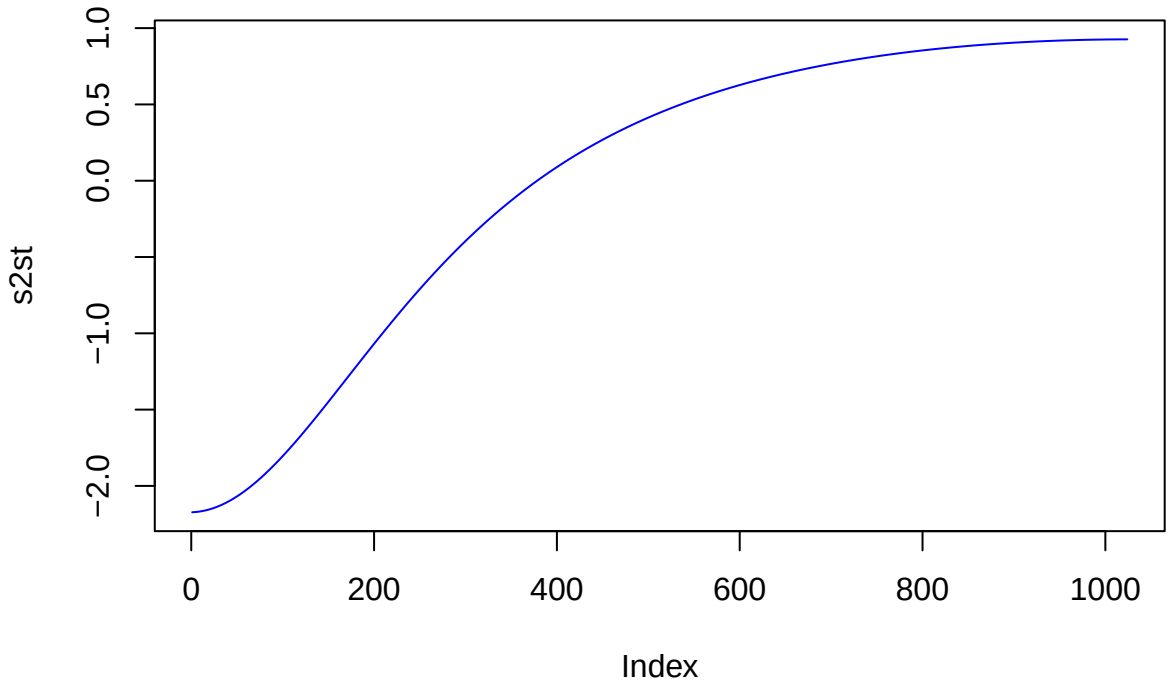
R Markdown

Start by defining our 2 modulating functions c1 and c2 and our 2 spectra s1 and s2. We use the package beyondWhittle to compute the spectral density of arma processes.

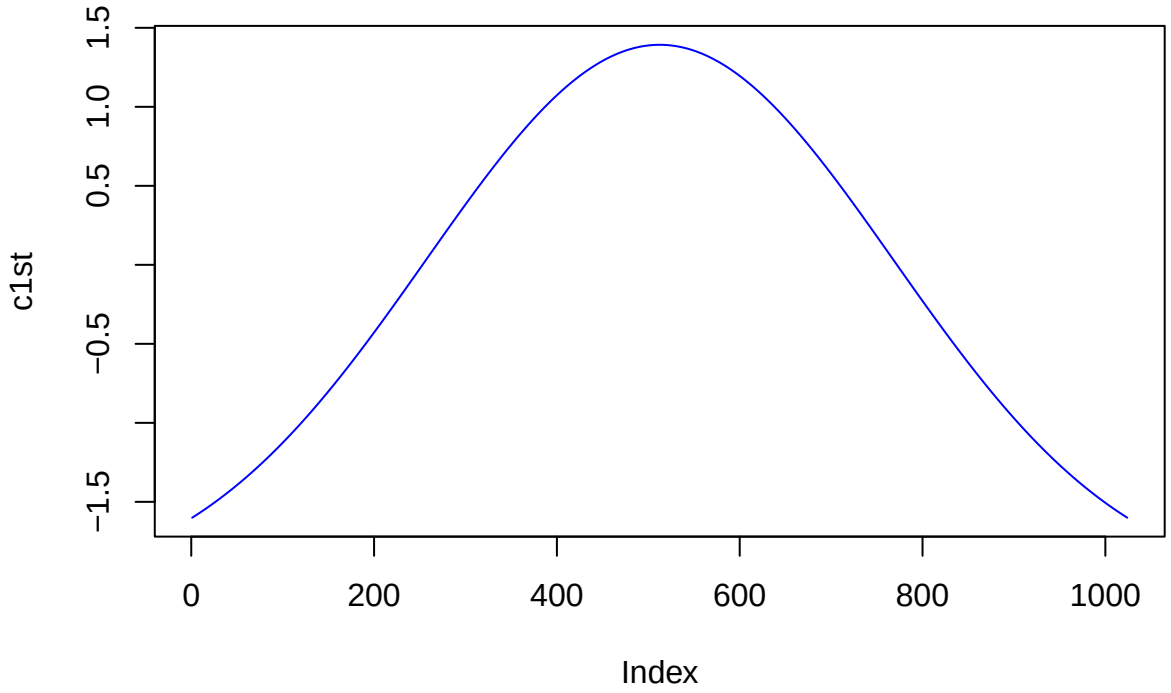
```
library(beyondWhittle) #has the psd_arma function
#first, define 2 spectra
freqs <- seq(.01,pi-.01,length.out=1024) #1024 frequencies between 0 and pi
s1 <- psd_arma(freqs,ar=c(.9,-.4),ma=numeric(0)) #AR(2) for s1
s2 <- psd_arma(freqs,ar=.4,ma=-.7) #ARMA(1,1) for s2
#function to standardize a vector (the independent components JADE returns are standardized)
stz <- function(x)
  + (x-mean(x))/sqrt(var(x))
s1st <- stz(s1)
s2st <- stz(s2)
#plot the standardized psds
plot(s1st,type="l",col="blue")
```



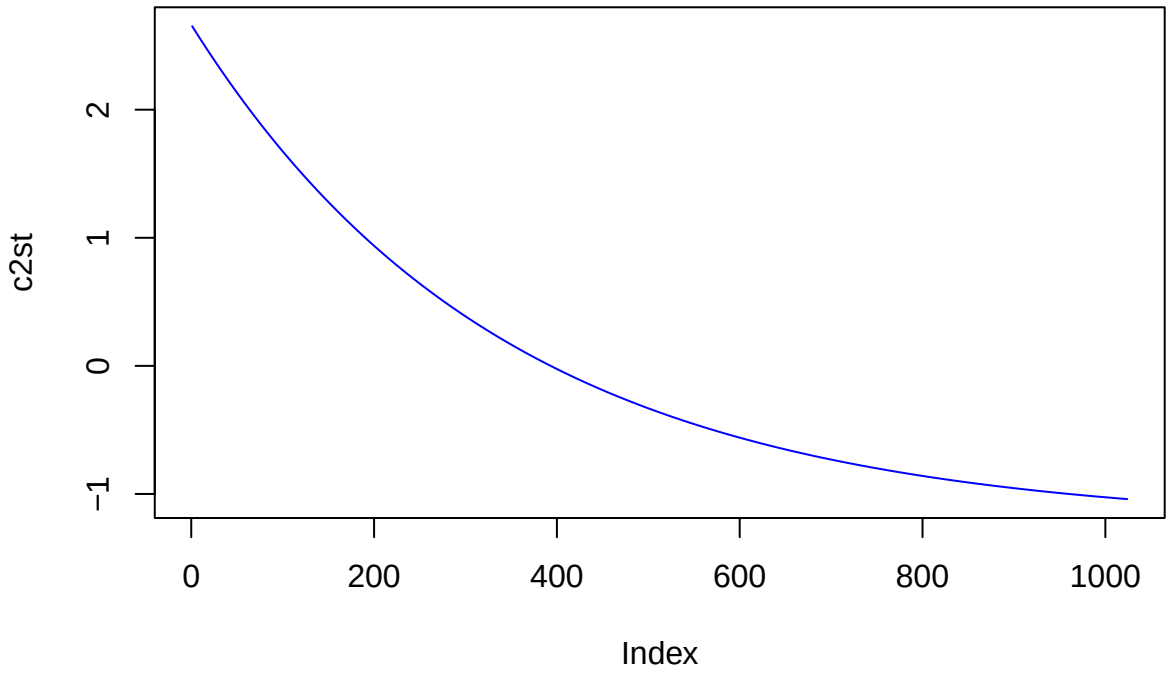

```
plot(s2st,type="l",col="blue")
```




```
#next, define the modulating functions
c1 <- dnorm(seq(-2,2,length.out=1024)) #normal density
c2 <- dexp(seq(.01,3,length.out=1024)) #exponential density
#standardize
c1st <- stz(c1)
c2st <- stz(c2)
#plot these
plot(c1st,type="l",col="blue")
```



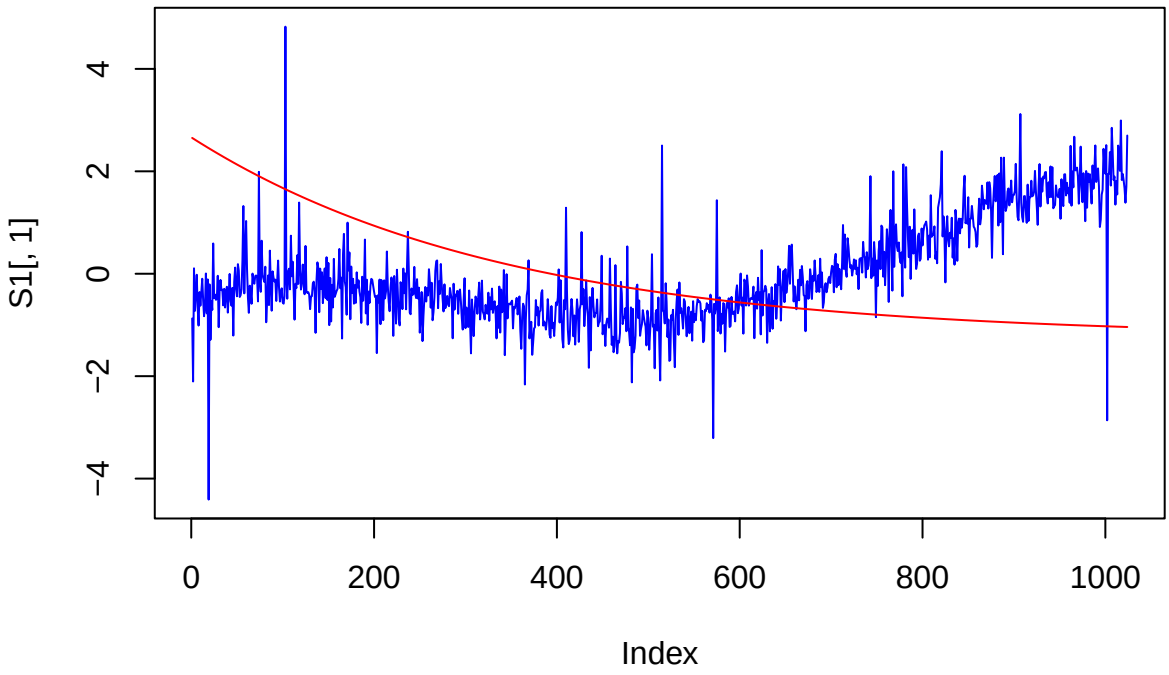
```
plot(c2st,type="l",col="blue")
```



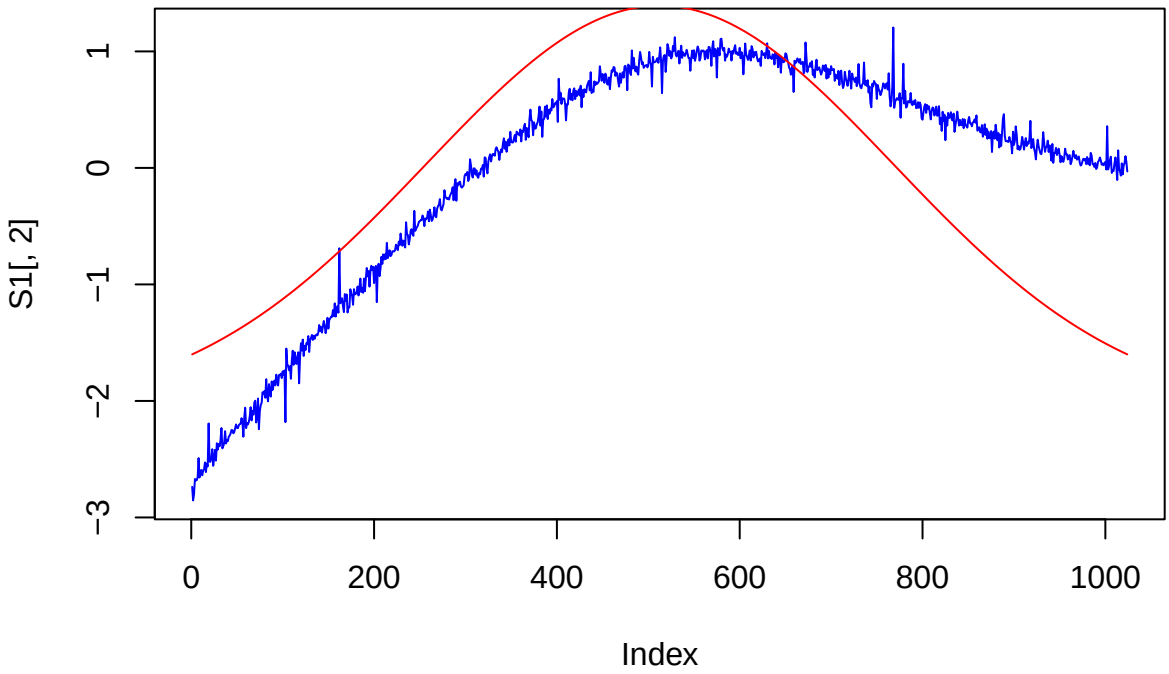

```
#noiseless time frequency matrix
gump2mean <- c1st%*%t(s1st)+c2st%*%t(s2st)
#scaled t distributed noise
gump2noise <- matrix(rt(1024*1024,3)/10,1024,1024)
#noisy time frequency matrix
gump2 <- gump2mean+gump2noise
```

The matrix gump2 is our noisy TFS matrix. It is 1024 by 1024. We will input pairs of columns of this matrix to JADE to try to recover (the standardize) c1 and c2. We will use the sample function to randomly choose 2 columns of gump2.

```
library(JADE)
i <- sample(1:1024,2)
S1 <- JADE(cbind(gump2[,min(i)],gump2[,max(i)]))$S
plot(S1[,1],type="l",col="blue")
lines(c2st,col="red")
```




```
plot(S1[,2],type="l",col="blue")
lines(c1st,col="red")
```



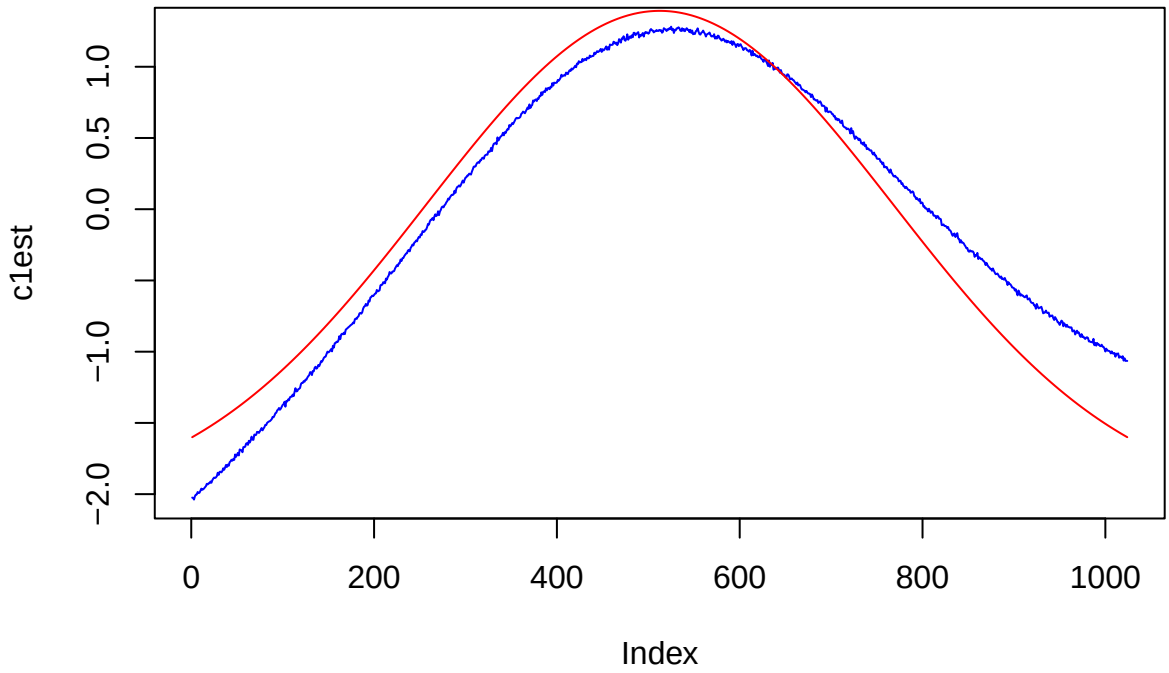
We see that c1 and c2 can be estimated via BSS up to a sign change. Also, the two columns of S1 (the 1024 by 2 matrix containing the estimates of the independent components) may be our estimates of c1 and c2, or of c2 and c1, but we don't know which order the columns are in. If the red lines in the plots got the order right then it was just by chance. Our approach to deal with this ambiguity in the independent component estimates will be as follows. Since this is a simulation proof of concept example we will cheat and assume we know the true c1 and c2 we are trying to estimate. We will call these our two reference functions r1 and r2. Also, in general we consider using k columns of the gump2 matrix to estimate the 2 independent components.

- set k and randomly choose k columns of the gump2 matrix

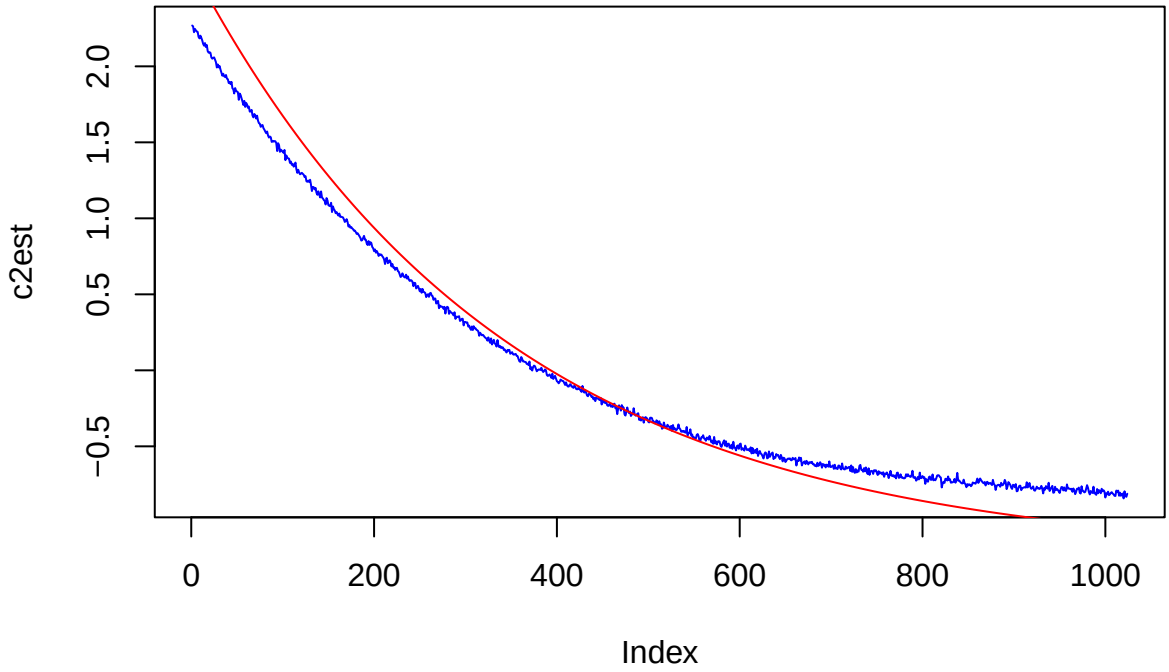
- call JADE to estimate the two independent components based on the k chosen columns of gump2. Put these into the 1024 by 2 matrix S1.
- replace S1 with cbind(S1,-S1), a 1024 by 4 matrix
- For each column of S1 compute the L2 distance between that column and r1. Save the column with the smallest distance.
- For each column of S1 compute the L2 distance between that column and r2. Save the column with the smallest distance.

We do the above steps for 1000 randomly chosen sets of k columns of gump2 to get 1000 estimates of c1 and 1000 estimates of c2. Our final estimates of c1 and c2 are the averages of these 1000 estimates. We can change the 1000 to any number we want, up to 1024 choose 2. The estimates are shown in blue and the true c1 and c2 in red in the plots below.

```
n <- 1000 #number of sets of k columns we will go through
k <- 4 #number of columns to use to estimate the two independent components
#define matrices to hold our n estimates of c1 and c2
c1estimates <- matrix(rep(0,n*1024),ncol=n)
c2estimates <- matrix(rep(0,n*1024),ncol=n)
for (j in 1:n) #our main loop
{
  i <- sample(1:1024,k) #sample k columns
  i <- sort(i) #sort the column indices from smallest to largest
  #define matrix X to hold the chosen k columns
  X <- gump2[,i[1]] #first column
  for (l in 2:k) #add remaining columns to X
    X <- cbind(X,gump2[,i[l]])
  #call JADE to estimate the 2 independent components based on X
  S1 <- JADE(X,n.comp=2)$S #IC estimates are in S
  S1 <- cbind(S1,-S1) #add negatives of the estimates
  #which columns of S1 are closest to c1 and c2 in L2 distance
  c1ind <- which.min(apply(sweep(S1,1,c1st)^2,2,sum))
  c2ind <- which.min(apply(sweep(S1,1,c2st)^2,2,sum))
  c1estimates[,j] <- S1[,c1ind]
  c2estimates[,j] <- S1[,c2ind]
}
#get our final estimates of c1 and c2 (average of the n estimates)
c1est <- apply(c1estimates,1,mean)
c2est <- apply(c2estimates,1,mean)
plot(c1est,type="l",col="blue")
lines(c1st,col="red")
```




```
plot(c2est,type="l",col="blue")
lines(c2st,col="red")
```



Concluding comments: There may be a better way to choose pairs of columns than just randomly - some pairs produce better estimates than others. It may be worth the time to try to think of a criterion for choosing columns. Also, in BSS one can have more observed series than independent components. So, for a 2 gump we could use 3 or more columns to estimate c1 and c2. This may be worth exploring. In practice we don't know the true c1 and c2 that we can use as reference functions to decide which of the estimated independent components, or their negatives, to use as estimates of c1 and c2. One possibility is to initially just use the two estimated independent components from the first pair of columns as the reference functions, and then keep a running average as the reference functions each time we try a new pair of columns. BSS can be used

to estimate s1 and s2 in exactly the same way except working with pairs of rows of the TFS matrix. Finally, I think we could use the BSS estimates of c1 and c2 to "bootstrap" an iterative estimation procedure where we use the BSS estimates to then estimate s1 and s2, then use these to re-estimate c1 and c2, and so on.