Implementation of RCoT

Robin Fisher

March 14, 2018

```
print(Sys.time())
```

```
## [1] "2020-04-12 14:11:26 EDT"
```

The library glmnet has the ridge regression function.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-13
```

(Strobl, Zhang, and Visweswaran 2019) describe a nonparametric test of conditional indpendence, meant to be faster than other tests available. Using theory about reproducing kernel hilbert spaces, they argue that selecting random elements from the fourier transform and testing for correlation among those transformed variables will test independence of the original variable with high probability. I would like to use such a test, so I am implementing it. The elements of that implemtation are here, along with some tests.

Set the seed for reproducibility:

```
set.seed((108))
```

First I set up my functions. The function *xi* produces a random function.

```
xi<-function(d2){
  #output should be a function that takes a d1*d2 matrix
  w<-matrix(rnorm(d2,0,1.0),d2,1)
  b<-runif(1,0,2*pi)
  out<-function(z,dg=F){
    if(!dg){return(sqrt(2)*cos(z%*%w+b))
    }else{
      return(list(w,b))
    }
  }
  #print(w)
  #print(b)
  out
}
```

The next function produces a test data set. The first two (A) columns are multivariate normal with a random covariance matrix. The 5th and 6th columns (C) columns depend on the A columns, and columns 3:4 (the B columns) depend direcly on the C columns. It is true that $A \perp\!\!\!\perp B|C$ in makeTestDat. It is not true in makeTestDat2.

```
makeIIDNormalDat<-function(nsiz=100000){
  trydat<-data.frame(matrix(rnorm(nsiz*6),nsiz))
  names(trydat)<-c("a1","b1","c1","c2","c3","c4")
  return(trydat)
}
```

```r
makeTestDat<-function(nsiz=1000){

  c1<-rnorm(nsiz)
  c2<-rnorm(nsiz)
  c3<-rnorm(nsiz)
  c4<-rnorm(nsiz)
  cmat<-cbind(c1,c2,c3,c4)
  avec<-matrix(1/(2*sqrt(2)),4,1)
  a1<-cmat%*%avec+rnorm(nsiz)/sqrt(2)
  bvec<-matrix(1,4,1)
  b1<-cmat%*%bvec+rnorm(nsiz)
  trydat<-data.frame(cbind(a1,b1,cmat))
  names(trydat)<-c("a1", "b2", "c1", "c2", "c3", "c4")
  return(trydat)
}
```

The next function makes the transformed data set, using the random function *xi*. As I have it here, the dimension of each of the three transformed variables is 5.

```r
makeRBaseDat<-function(trydat,rbasis=xi,ncolv=list(1,2,3:6)){
  #  ncolv=list(acols,bcols,ccols)
  d<-c(5,5,5)
  acols<-ncolv[[1]]
  bcols<-ncolv[[2]]
  ccols<-ncolv[[3]]
  nsiz<-nrow(trydat)
  codtstdat<-matrix(0,nsiz,sum(d))
  codtstdat<-data.frame((codtstdat))
  names(codtstdat)<-list()
  for(i in 1:d[1]){
    rfun<-rbasis(length(acols))
    codtstdat[,i]<-rfun(as.matrix(trydat[,acols]))
    #codtstdat[,i]<-trydat[,acols]
    names(codtstdat)[i]<-paste("a",i,sep="")

  }
  for(i in 1:d[2]){
    rfun<-rbasis(length(ccols))
    codtstdat[,i+d[1]]<-rfun(as.matrix(trydat[,ccols]))
    #codtstdat[,i+d[1]]<-trydat[,2+min(i,4)]
    names(codtstdat)[i+d[1]]<-paste("c",i,sep="")
  }
  for(i in 1:d[3]){
    rfun<-rbasis(length(bcols))
    codtstdat[,i+d[1]+d[2]]<-rfun(as.matrix(trydat[,bcols]))
    #codtstdat[,i+d[1]+d[2]]<-trydat[,bcols]
    names(codtstdat)[i+d[1]+d[2]]<-paste("b",i,sep="")
  }
  marg<-apply(codtstdat,2,mean)
  #codtstdat<-sweep(codtstdat,2,marg,"-")

  margsd<-apply(codtstdat,2,sd)
  margsd<-ifelse(margsd!=0,margsd,1)
  #codtstdat<-sweep(codtstdat,2,margsd,"/")
  return(codtstdat)
}


makeRBaseDat0<-function(trydat,rbasis=xi,ncolv=list(1,2,3:6)){
  #  ncolv=list(acols,bcols,ccols)
  d<-c(5,5,5)
  acols<-ncolv[[1]]
  bcols<-ncolv[[2]]
  ccols<-ncolv[[3]]
  nsiz<-nrow(trydat)
  codtstdat<-matrix(0,nsiz,sum(d))
  codtstdat<-data.frame((codtstdat))
  names(codtstdat)<-list()
  for(i in 1:d[1]){
    rfun<-rbasis(length(acols))
    codtstdat[,i]<-rfun(as.matrix(trydat[,acols]))
    #codtstdat[,i]<-trydat[,acols]
    names(codtstdat)[i]<-paste("a",i,sep="")

  }
  for(i in 1:d[2]){
    rfun<-rbasis(length(ccols))
    codtstdat[,i+d[1]]<-rfun(as.matrix(trydat[,ccols]))
    #codtstdat[,i+d[1]]<-trydat[,2+min(i,4)]
    names(codtstdat)[i+d[1]]<-paste("c",i,sep="")
  }
  for(i in 1:d[3]){
    rfun<-rbasis(length(bcols))
    codtstdat[,i+d[1]+d[2]]<-rfun(as.matrix(trydat[,bcols]))
    #codtstdat[,i+d[1]+d[2]]<-trydat[,bcols]
    names(codtstdat)[i+d[1]+d[2]]<-paste("b",i,sep="")
  }
  marg<-apply(codtstdat,2,mean)
  #codtstdat<-sweep(codtstdat,2,marg,"-")

  margsd<-apply(codtstdat,2,sd)
  margsd<-ifelse(margsd!=0,margsd,1)
  #codtstdat<-sweep(codtstdat,2,margsd,"/")

  return(codtstdat)
}
```

Transform the data set into the random basis. Note the columns are standardized with (mean, varaince)=(0,1). I don't want to keep track of the means and this way I know what the variances should be.

Estimate the covariance matrix of the transformed variables, get the conditional correlation matrix, and estimate the covariance of that. The conditional correlation matrix is calculated by actuall doing the regressions of the A and B variables on C, then forming the cocovariances fo the covariances on the residuals. I need to check to make sure I handled my indices right and everything. Strobl et al use ridge regressions. Why? Maybe they just habitually handle the high-collinearity situation. Maybe the chance the two draws from the xi function are close is higher than I would expect.

Another thing they do that I don't see the point of is this. They use the eigendecomposition of the Pi matrix, which is the covariance matrix of the conditional crrelation matrix between A and B | C, but then they use the super-satterthwaite approximation to the weighted sum of chi-squared distributions. If we actually do the eigen deomposition, we have the eigenvalues and we can make a test statistic which is a simple χ^2 statistic, I think. Am I missing something?

```r
makeOLSResDat<-function(codtstdat,nc=c(5,5,5),lam=0.001){
acols<-1:nc[1]
ccols<-(nc[1]+1):(nc[1]+nc[2])
bcols<-(nc[1]+nc[2]+1):(sum(nc))
nsiz<-nrow(codtstdat)
resDat<-data.frame(matrix(0,nsiz,length(acols)+length(bcols)))
names(resDat) <-names(codtstdat)[c(acols,bcols)]
cvars<-paste0("c",1:nc[2],collapse="+")
paste("a1 ~ ",cvars,paste())
for(i in 1:length(acols)){
  fmla<-paste(names(codtstdat)[acols[i]]," ~ ",cvars)
 # print(fmla)
  resDat[,i]<-lm(fmla,data=codtstdat)$res
}
for(i in 1:length(bcols)){
  fmla<-paste(names(codtstdat)[bcols[i]]," ~ ",cvars)
 # print(fmla)
  resDat[,i+length(acols)]<-lm(fmla,data=codtstdat)$res
}
return(resDat)
}


makeRidgeResDat<-function(codtstdat,nc=c(5,5,5),lam=0.001){
  library(glmnet)
  acols<-1:nc[1]
  ccols<-(nc[1]+1):(nc[1]+nc[2])
  bcols<-(nc[1]+nc[2]+1):(sum(nc))
  nsiz<-nrow(codtstdat)
  resDat<-data.frame(matrix(0,nsiz,length(acols)+length(bcols)))
  names(resDat) <-names(codtstdat)[c(acols,bcols)]
  for(i in 1:length(acols)){
    modHere<-glmnet(x=as.matrix(codtstdat[,ccols]),
                    y=as.matrix(codtstdat[,acols[i]]))
    # print(fmla)
    resDat[,i]<-codtstdat[,acols[i]]-
      predict(modHere,newx=as.matrix(codtstdat[,ccols]),s=1.7e-4)
  }
  for(i in 1:length(bcols)){
    modHere<-glmnet(x=as.matrix(codtstdat[,ccols]),
                    y=as.matrix(codtstdat[,bcols[i]]))
    # print(fmla)
    resDat[,i+length(acols)]<-codtstdat[,bcols[i]]-
      predict(modHere,newx=as.matrix(codtstdat[,ccols]),s=1.7e-4)
  }
  names(resDat)<-c("a1", "a2", "a3", "a4", "a5", "b1", "b2", "b3", "b4", "b5")
  return(resDat)
}
```

```r
#makeResDat<-makeOLSResDat
makeResDat<-makeOLSResDat
#resDat<-makeResDat(codtstdat)
```

Now we have the dataset of residuals of the regressions of tha A and B variables on C. In the limit and the sample gets large and the number of random functions we generate, we are only testing for correlatedness between the A columns and the B columns.

Here is the test function:

```r
#This function lets me map an index of the vectized dxd matrix into the coordinates of the matrix.
indMap<-function(k,d=5){
  out<-c(k%%d,floor(k/d))
  if(out[1]==0) {
    out[1]<-d
  }else{
    out[2]<-out[2]+1
  }

  out
}

# testCor0old<-function(codtstdat,resDat){
#   acols<-1:5
#   bcols<-6:10
#   pihat<-matrix(0,25,25)
#   ro<-0
#   co<-0
#
#   for(i in 1:length(acols)){
#     for(j in (length(acols)+1):(length(acols)+length(bcols))){
#       ro<-ro+1
#       co<-0
#       for(k in 1:length(acols)){
#         for(ell in (length(acols)+1):(length(acols)+length(bcols))){
#           co<-co+1
#           pihat[ro,co]<-sum(resDat[,i]*resDat[,j]*resDat[,k]*resDat[,ell])
#         }
#       }
#     }
#   }
#   pihat<-pihat/nsiz
#   pieig<-eigen(pihat)
#   vsighat<-cov(codtstdat)
#   corab<-c(solve(solve(vsighat)[c(acols,bcols),c(acols,bcols)])[1:5,6:10])
#
#   normcorab<-diag(1/sqrt(pieig$values))%*%t(pieig$vectors)%*%matrix(corab,ncol=1)
#   chis<-sum(normcorab^2)
#   pval<-1-pchisq(chis,16)
#   return(pval)
# }


testCor0<-function(codtstdat,resDat){
  #tarr<-array("",c(25,25))
```

```r
acols<-1:5
bcols<-11:15
pihat0<-matrix(0,25,25)
for(co in 1:25){
  k<-indMap(co)[1]
  ell<-indMap(co)[2]+5
  for(ro in 1:25){
    i<-indMap(ro)[1]
    j<-indMap(ro)[2]+5
    pihat0[ro,co]<-sum(resDat[,i]*resDat[,j]*resDat[,k]*resDat[,ell])
 #   tarr[ro,co]<-paste(i,j,k,ell)
  }
}
pihat0<-pihat0/nsiz #covaraince matrix of the contional covariances
pihat<-pihat0/nsiz #covariance matrix of the sample mean conditional covariances
#See Strobl, et al eqn 16 for the CLT invocation
pieig<-eigen(pihat) ##the eigenvalues.

if(F){
vsighat<-cov(codtstdat)+(1e-5)*diag(rep(1,15))
#corab<-c(solve(solve(vsighat)[c(acols,bcols),c(acols,bcols)])[acols,bcols])
regCo<-vsighat[c(1:5,11:15),6:10]%*%solve(vsighat[6:10,6:10])
corab<-vsighat[c(1:5,11:15),c(1:5,11:15)]-regCo%*%vsighat[6:10,c(1:5,11:15)]
}else{
  corab<-cov(resDat[,1:5],resDat[,6:10])
}
#regCoAB<-corab[1:5,6:10]%*%solve(corab[6:10,6:10])
#so sum(corab^2)~=\sum lambda_i \chi^_1

#I use a simple moment-matching method (due to Sattertwhaite?) to get an
#approximate gamma distrbution for the test. Strobl et al use a more complicated
#method.  I don't know how important it is.

m<-sum(pieig$values)
v<-2*sum((pieig$values)^2)

alph<-0.5*m^2/v
bet<-m/v

xisq<-sum(corab^2)

pval<-pgamma(xisq,alph,bet,lower.tail = FALSE)


# the commented section here is my attempt to be more precise
# by using the eigen decompostion to get a standardized normal
# (under the null) which would lead to a precise test.  Soemthin wasn't right, and   # I want to get
# unstable, or am I still consructing the Pi matrix wrong wrt to permutations of
# the rows? That would do it, I think.

# pieig$values<-ifelse(pieig$values>1e-16,pieig$values,1e-16)
#
# normcorab<-diag(1/sqrt(pieig$values))%*%t(pieig$vectors)%*%matrix(corab,ncol=1)
```

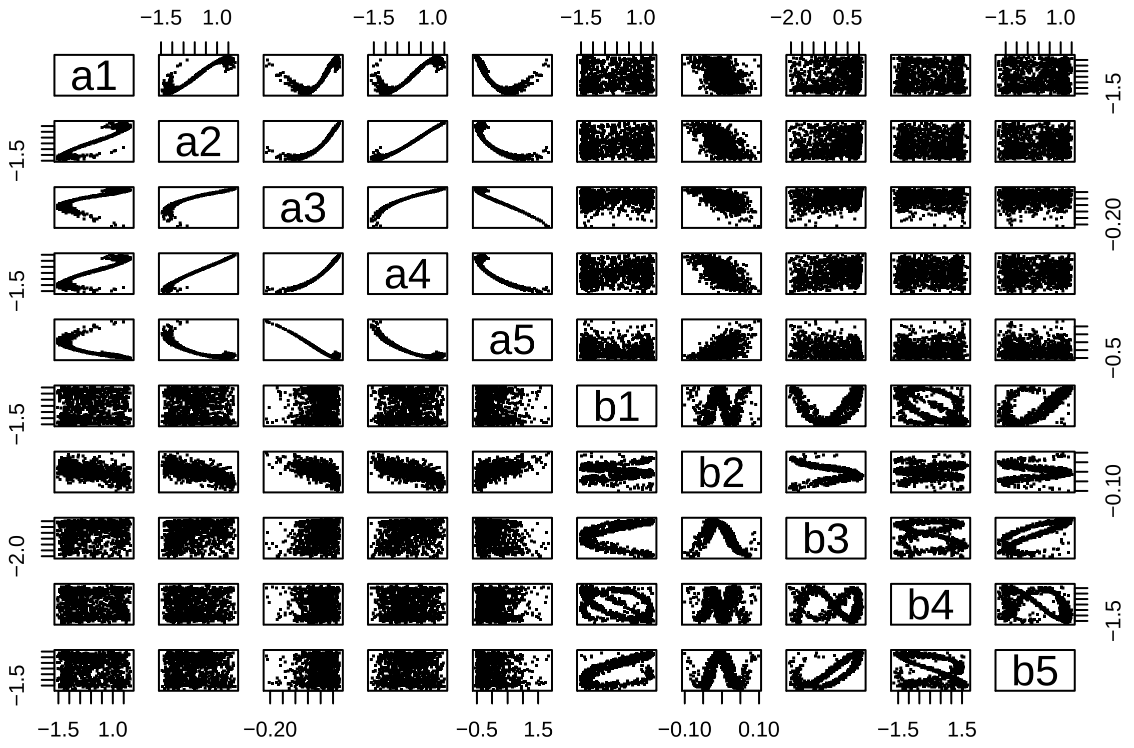
```r
  # chis<-sum(normcorab^2)
  # pval<-1-pchisq(chis,16)
  return(pval)
}

wisVar<-function(i,j,V,n){
  return((V[i,j]^2+V[i,i]*V[j,j])/n)
}

LR<-function(resDat,d=5){
  ns<-nrow(resDat)
  S<-cov(resDat)
  Sa<-cov(resDat[,1:5])
  Sb<-cov(resDat[,6:10])
  LRTestStat<-nsiz*log(det(S))-nsiz*(log(det(Sa))+log(det(Sb)))
}


testCor<-function(codtstdat){
  resDat<-makeOLSResDat(codtstdat,lam=0.1)
  return(testCor0(codtstdat,resDat))
}
```

test chunk

```r
set.seed(1024)
nsiz<-1000
trydat<-makeTestDat(nsiz)
#trydat<-makeIIDNormalDat(nsiz)
codtstdat<-makeRBaseDat(trydat)
resDat<-makeResDat(codtstdat)
testCor(codtstdat)
```

```
## [1] 8.855707e-08
```

```r
#pairs(trydat[1:1000,],pch=".")
pairs(resDat[1:1000,],pch=".")
```



```
resVar<-cov(resDat)

library(kpcalg)
sS<-list(data=trydat,method="hsic.gamma")
kernelCItest(x=1,y=2,S=c(3,4,5,6),suffStat=sS)
```

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

```
sS<-list(data=codtstdat,method="hsic.gamma")
kernelCItest(x=4,y=14,S=c(6:10),suffStat=sS)
```

```
## [1] 0.4950495
```

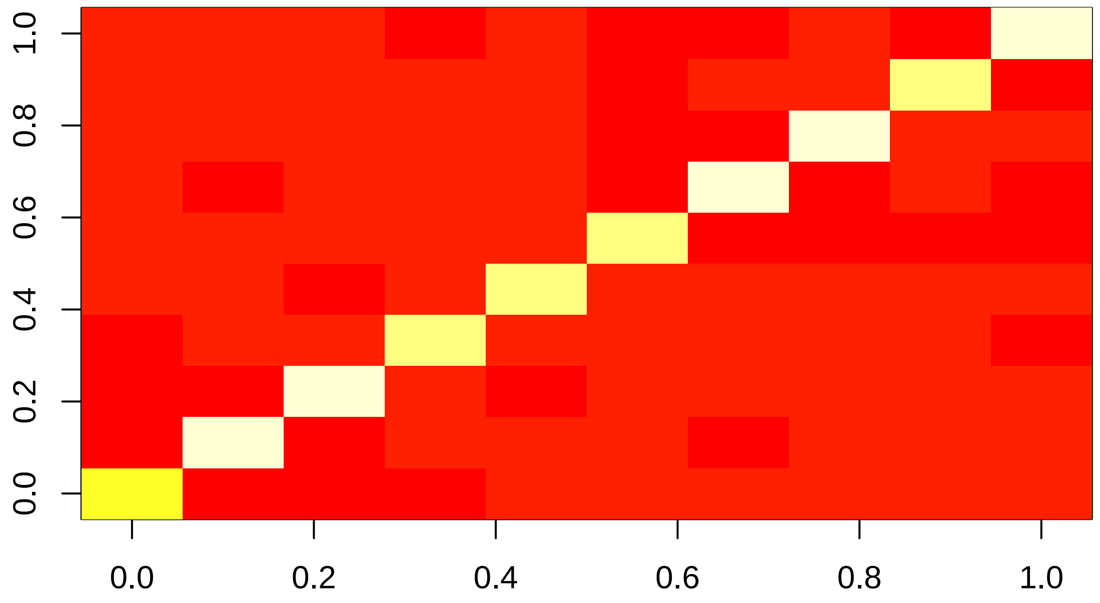
```
sS<-list(data=resDat,method="hsic.gamma")
kernelCItest(x=4,y=9,S=c(),suffStat=sS)
```

```
## [1] 0.4059406
```

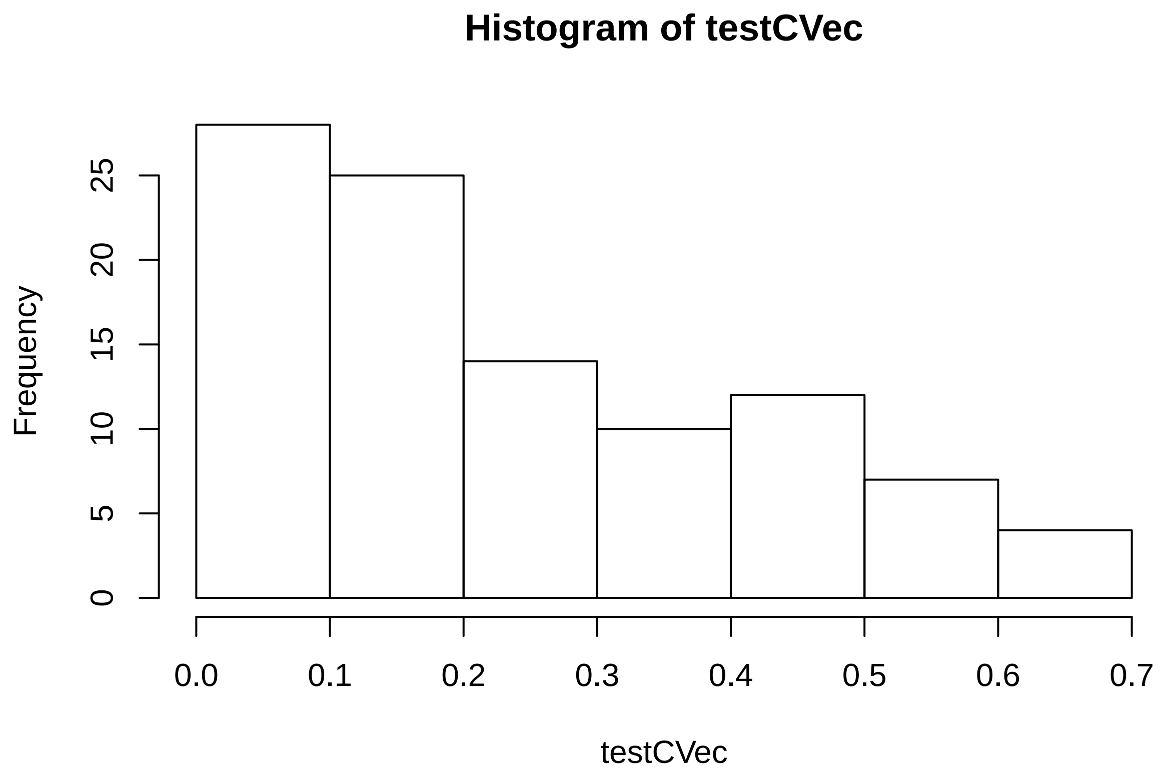
Here are the results of 100 replications of the estimation for a single randomly generated test tata set when $A \perp\!\!\!\perp B|C$:

```
nsiz<-1000

rescovave<-matrix(0,10,10)
testCVec<-matrix(0,B,1)
timeBefore<-Sys.time()
for(i in 1:B){
  #trydat<-makeTestDat(nsiz)
  trydat<-makeIIDNormalDat(nsiz)
  codtstdat<-makeRBaseDat(trydat,rbasis=xi)#,ncolv=list(acols,bcols,ccols))
  testCVec[i,1]<-testCor(codtstdat)
  rescovave<-rescovave+cov(makeResDat(codtstdat))
}
```

```
timeAfter<-Sys.time()
rescovave<-rescovave/B
image(rescovave)
```



```
hist(testCVec)
```

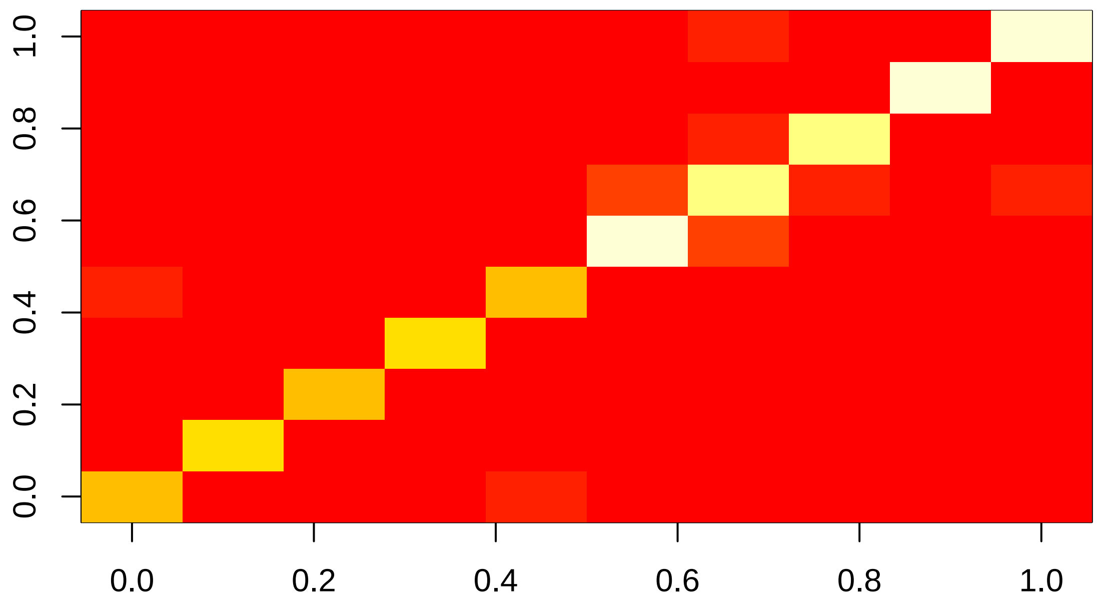


That took 13.70167 secs for 100 iterations. I think that's faster than the default test in the *rfci* algorithm. The residuals were calculated with the OLS regressions, rather than the ridge regression; I expect the OLS algorithm to be fster, but I don't actually know.

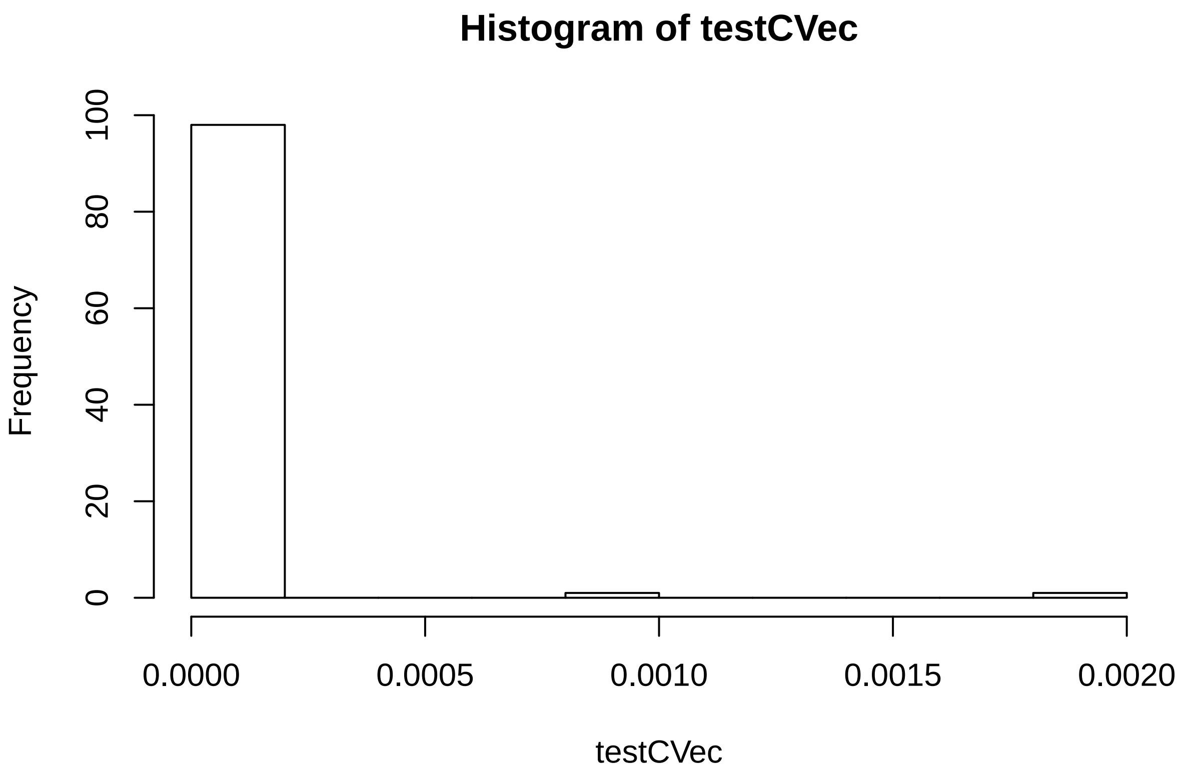
That histogram doesn't seem too bad; in general, of course, a hypothesis test under the null hypothesis in the classical method is $U(0,1)$ when the variance is well-calibrated. This was the easy of all possible tests, in a way, since the data columns are all independent normal. An interesing note is that I had to add that diagonal to the $cov(sighat)$ to make it so it was always invertible, even in this case. Sometimes, as it ran through hte replications, it wasn't; I think that may mean that the random function in makeRBase was almost linear. Maybe I should change the SD of the w vector. I'll think about that later.

Below I use makeTestDat, with is multivariate normal where the conditional independence $A \perp\!\!\!\perp B|C$ holds, but no global indpendence. I replicate the data ,generate the random basis, and perfom the test 100 times. Again, the residuals were caculated with OLS.

```r
nsiz<-1000

rescovave<-matrix(0,10,10)
testCVec<-matrix(0,B,1)
timeBefore<-Sys.time()
for(i in 1:B){
  trydat<-makeTestDat(nsiz)
  #trydat<-makeIIDNormalDat(nsiz)
  codtstdat<-makeRBaseDat(trydat,rbasis=xi)#,ncolv=list(acols,bcols,ccols))
  testCVec[i,1]<-testCor(codtstdat)
  rescovave<-rescovave+cov(makeResDat(codtstdat))
}
timeAfter<-Sys.time()
rescovave<-rescovave/B
image(rescovave)
```



```r
hist(testCVec)
```



That took 14.03231 secs for 100 iterations.

Next, $A \perp\!\!\!\perp B|C$, but the distribution is more complicated.

```
if(F){
nsiz<-1000

testCVec<-matrix(0,B,1)
for(i in 1:B){
  trydat<-makeTestDat3(nsiz)
  codtstdat<-makeRBaseDat(trydat,rbasis=xi)#,ncolv=list(acols,bcols,ccols))
  testCVec[i,1]<-testCor(codtstdat)
}
image(cov(makeResDat(codtstdat)))
hist(testCVec)
}
```

References

Strobl, Eric V, Kun Zhang, and Shyam Visweswaran. 2019. "Approximate Kernel-Based Conditional Independence Tests for Fast Non-Parametric Causal Discovery." *Journal of Causal Inference* 7 (1). De Gruyter.