STA410 Homework 1

student number: 1003942326

Yulin WANG

2019-10-01

Question1

(a)

Plug $\hat{Z} = A_m Z A_n^T$ in:

$$\begin{aligned}
D_m^{-1} A_m^T \hat{Z} A_n D_n^{-1} &= D_m^{-1} A_m^T \{A_m Z A_n^T\} A_n D_n^{-1} \\
&= D_m^{-1} \{A_m^T A_m\} Z \{A_n^T A_n\} D_n^{-1} \\
&= \{D_m^{-1} D_m\} Z \{D_n D_n^{-1}\} \quad \text{since } A_m^T A_m = D_m \text{ and } A_n^T A_n = D_n \\
&= Z
\end{aligned}$$

(b)

hard-thresholding function

```
library('dtt')
#hard-thresholding function

hard_thres <- function(dctmat,lambda) {
  # if lambda is missing, set it to the 0.8 quantile of abs(dctmat)
  if(missing(lambda)) lambda <- quantile(abs(dct),0.8)
  a <- dctmat[1,1]
  dctmat1 <- ifelse(abs(dctmat)>lambda,dctmat,0)
  dctmat1[1,1] <- a
  # inverse DCT to obtain denoised image "clean"
  clean <- mvdct(dctmat1,inverted=T)
  clean <- ifelse(clean<0,0,clean)
  clean <- ifelse(clean>1,1,clean)
  clean
  }
```

soft-thresholding function

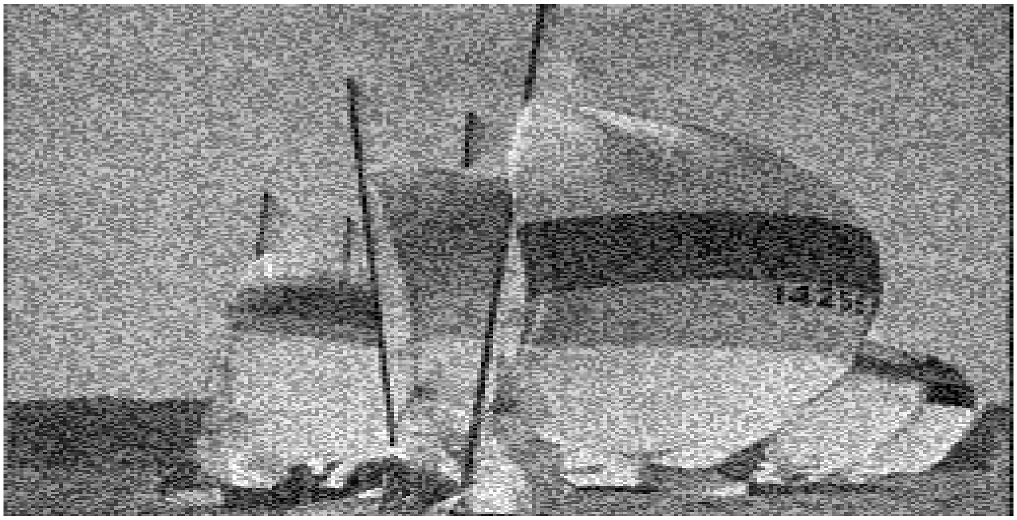
```
library('dtt')
#soft-thresholding function

soft_thres <- function(dctmat,lambda) {
  if(missing(lambda)) lambda <- quantile(abs(dct),0.8)
  a <- dctmat[1,1]
  dctmat1 <- sign(dctmat)*pmax(abs(dctmat)-lambda,0)
  dctmat1[1,1] <- a
  clean <- mvdct(dctmat1,inverted=T)
  clean <- ifelse(clean<0,0,clean)
  clean <- ifelse(clean>1,1,clean)
  clean
  }
```

(c)

original image

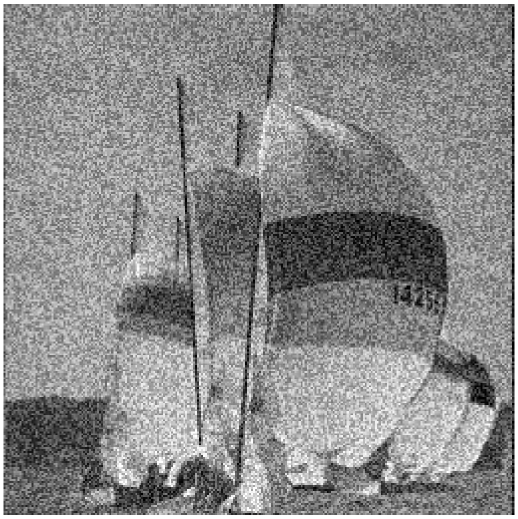
```r
#load the 'boats.txt' and create the original image
boats <- matrix(scan("boats.txt"),ncol=256,byrow=T)
image(boats, axes=F, col=grey(seq(0,1,length=256)),sub="original image")
```



original image

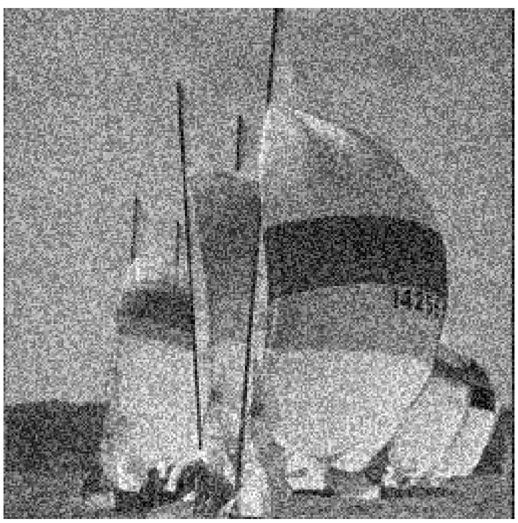
create four denoised images by using hard-thresholding with $\lambda = 5, 10, 20, 40$

```r
#create the images by hard-thres
boat1 <- hard_thres(mvdct(boats),lambda=5)
boat2 <- hard_thres(mvdct(boats),lambda=10)
boat3 <- hard_thres(mvdct(boats),lambda=20)
boat4 <- hard_thres(mvdct(boats),lambda=40)
image1<-image(boat1, axes=F, col=grey(seq(0,1,length=256)),
              sub="hard-thresholding with lambda = 5",asp=1)
```



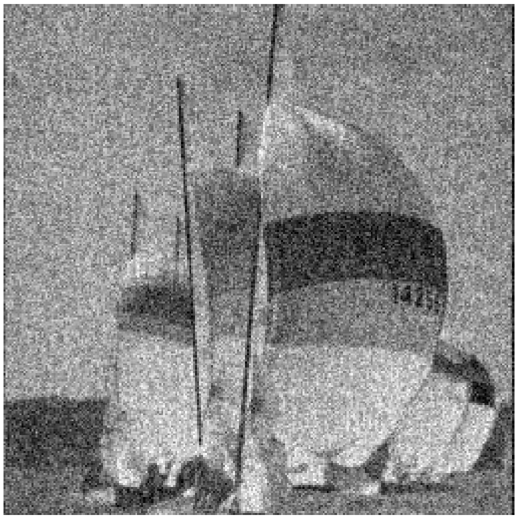
hard−thresholding with lambda = 5

```
image2<-image(boat2,axes=F, col=grey(seq(0,1,length=256)),
              sub="hard-thresholding with lambda = 10",asp=1)
```



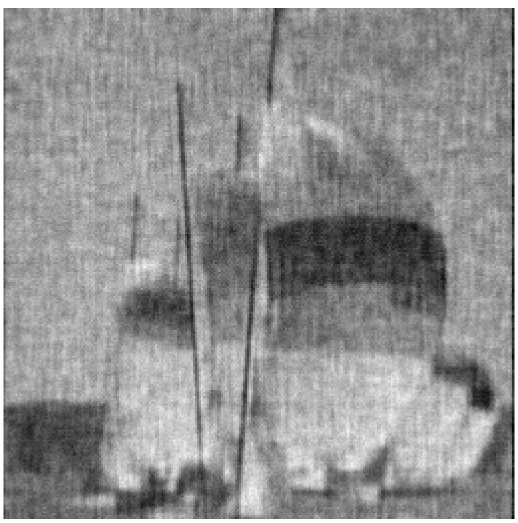
hard−thresholding with lambda = 10

```
image3<-image(boat3,axes=F, col=grey(seq(0,1,length=256)),
              sub="hard-thresholding with lambda = 20",asp=1)
```



hard−thresholding with lambda = 20

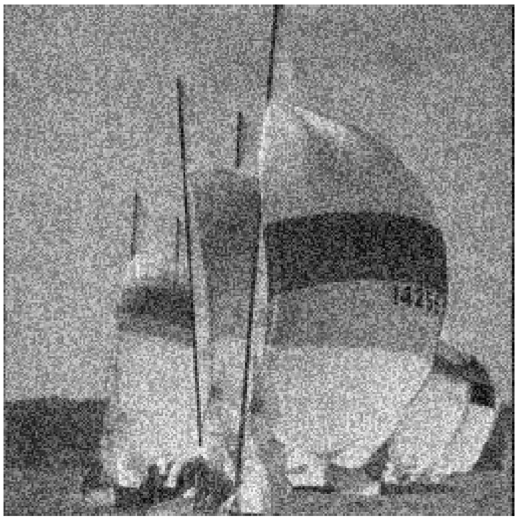
```r
image4<-image(boat4,axes=F, col=grey(seq(0,1,length=256)),
              sub="hard-thresholding with lambda = 40",asp=1)
```



hard−thresholding with lambda = 40

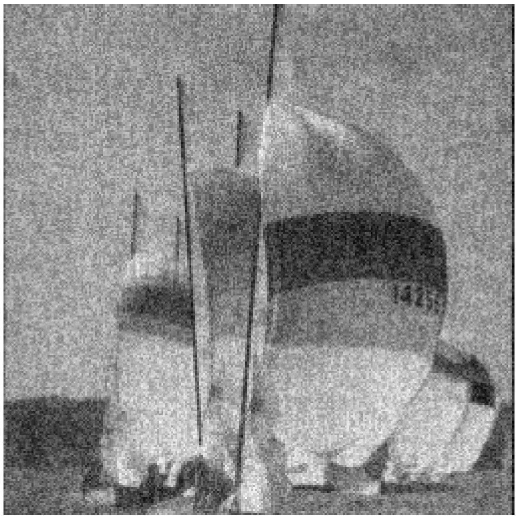
create four denoised images by using soft-thresholding with $\lambda = 5, 10, 20, 40$

```r
#create the images by soft-thres
boat5 <- soft_thres(mvdct(boats),lambda=5)
boat6 <- soft_thres(mvdct(boats),lambda=10)
boat7 <- soft_thres(mvdct(boats),lambda=20)
boat8 <- soft_thres(mvdct(boats),lambda=40)
image5<-image(boat5, axes=F, col=grey(seq(0,1,length=256)),
              sub="soft-thresholding with lambda = 5",asp=1)
```



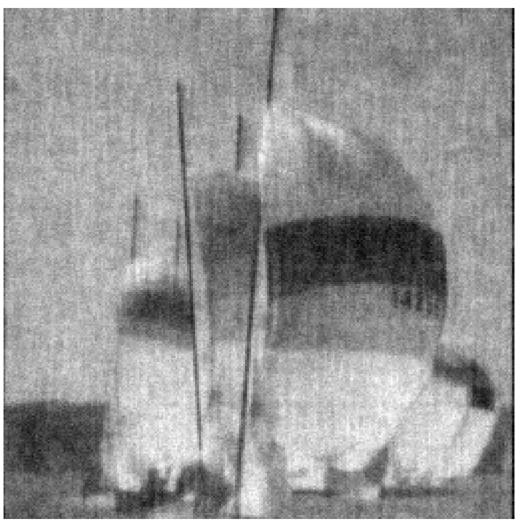
soft−thresholding with lambda = 5

```r
image6<-image(boat6,axes=F, col=grey(seq(0,1,length=256)),
              sub="soft-thresholding with lambda = 10",asp=1)
```



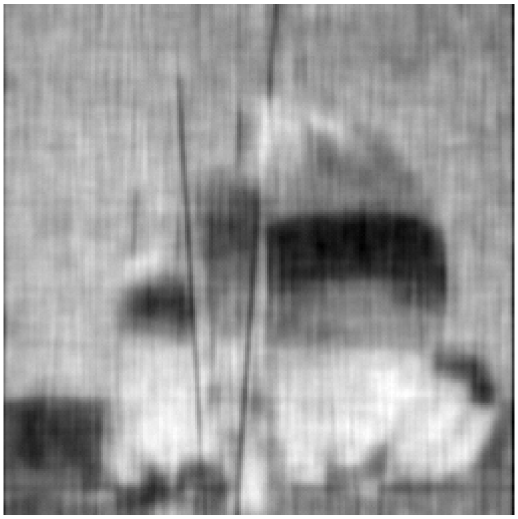
soft−thresholding with lambda = 10

```r
image7<-image(boat7,axes=F, col=grey(seq(0,1,length=256)),
              sub="soft-thresholding with lambda = 20",asp=1)
```



soft−thresholding with lambda = 20

```r
image8<-image(boat8,axes=F, col=grey(seq(0,1,length=256)),
              sub="soft-thresholding with lambda = 40",asp=1)
```



soft−thresholding with lambda = 40

conclusion

Compared the above eight denoised images and the original one, we can find that for both hard-thresholding and soft-thresholding functions, the smaller λ, the better image denoised, and the hard-thresholding function works better than soft-thresholding for the same λ.

Question2

(a)

Since U and V are independent, so

$$g(s) = E(s^X) = E(s^{U+2V}) = E(s^U \cdot s^{2V}) = E(s^U) \cdot E(s^{2V})$$

And we have:

$$\begin{aligned}
E(s^U) &= \sum_{k=0}^{\infty} s^k P(U = k) \\
&= \sum_{k=0}^{\infty} s^k \cdot \frac{e^{-\lambda_u} \lambda_u^k}{k!} \qquad \text{since } U \sim Pois(\lambda_u) \\
&= e^{-\lambda_u} \cdot \sum_{k=0}^{\infty} \frac{s^k \lambda_u^k}{k!} \\
&= e^{-\lambda_u} \cdot \sum_{k=0}^{\infty} \frac{(\lambda_u s)^k}{k!} \\
&= e^{-\lambda_u} \cdot e^{\lambda_u s} \\
&= e^{\lambda_u (s-1)}
\end{aligned}$$

$$\begin{aligned}
E(s^{2V}) &= \sum_{k=0}^{\infty} s^{2k} P(V = k) \\
&= \sum_{k=0}^{\infty} s^{2k} \cdot \frac{e^{-\lambda_v} \lambda_v^k}{k!} \qquad \text{since } V \sim Pois(\lambda_v) \\
&= e^{-\lambda_v} \cdot \sum_{k=0}^{\infty} \frac{s^{2k} \lambda_v^k}{k!} \\
&= e^{-\lambda_v} \cdot \sum_{k=0}^{\infty} \frac{(\lambda_v s^2)^k}{k!} \\
&= e^{-\lambda_v} \cdot e^{\lambda_v s^2} \\
&= e^{\lambda_v (s^2-1)}
\end{aligned}$$

Thus,

$$\begin{aligned}
g(s) = E(s^X) &= e^{\lambda_u (s-1)} \cdot e^{\lambda_v (s^2-1)} \\
&= \exp[\, \lambda_u (s-1) + \lambda_v (s^2 - 1)]
\end{aligned}$$

(b)

Using the inequality, for any $s > 1$, we can define M to satisfy

$$\frac{\exp[\,\lambda_u(s-1) + \lambda_v(s^2-1)]}{s^M} = \epsilon$$

then we have

$$M(s) = \frac{\lambda_u(s-1) + \lambda_v(s^2-1) - ln(\epsilon)}{ln(s)}$$

where $P(X \geq M(s)) \leq \epsilon$. Since this is true for any $s > 1$, we can take $M(s)$ to be as small as possible:

$$M = \inf_{s>1} \frac{\lambda_u(s-1) + \lambda_v(s^2-1) - ln(\epsilon)}{ln(s)}$$

with $P(X \geq M) \leq \epsilon$.

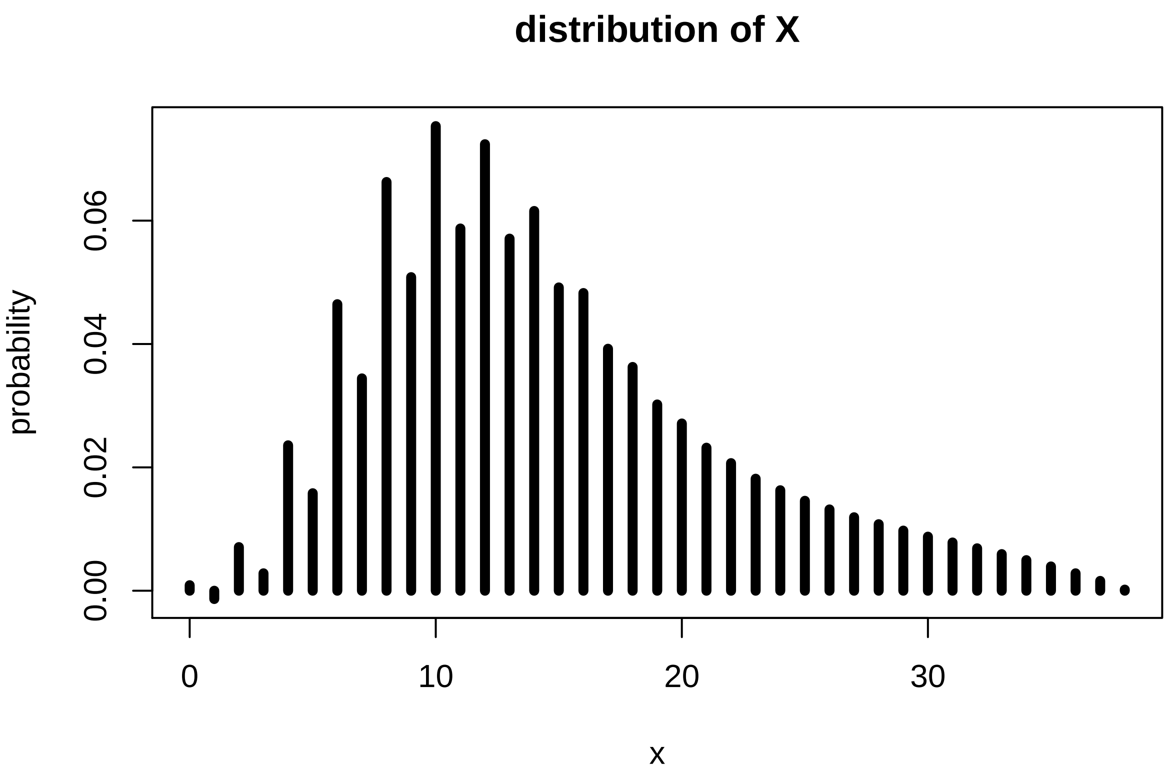
(c)

The R function named "find_dist" that implements the algorithm.

```
find_dist <- function(lambda_u,lambda_v,epsilon){
  # three inputs: lambda_u, lambda_v, and epsilon
  S <- c(1001:10000)/1000 #a discrete set of points S
  M <- min((lambda_u*(S-1)+lambda_v*(S^2-1)-log(epsilon))/log(S)) #value of M
  s <- exp(-2*pi*1i*c(0:(M-1))/M) #value of s
  gs <- exp(lambda_u*(s-1) + lambda_v*(s^2-1)) #g(s): pgf of X
  px <- Re(fft(gs,inverse=T))/M #evaluate p(X=x) by computing the inverse fft of g(s)
  r <- list(x=c(0:(M-1)),probs=px)
  plot(r$x,r$probs,type="h",lwd=5,main="distribution of X", xlab="x",ylab="probability")
}
```

(i) When $\lambda_u = 1$ and $\lambda_v = 5$

```
find_dist(lambda_u=1,lambda_v=5,epsilon=10^(-5))
```



(ii) When $\lambda_u = 0.1$ and $\lambda_v = 2$

```
find_dist(lambda_u=0.1,lambda_v=2,epsilon=10^(-5))
```