STA410 A1

Name: Harold Hyun Woo Lee

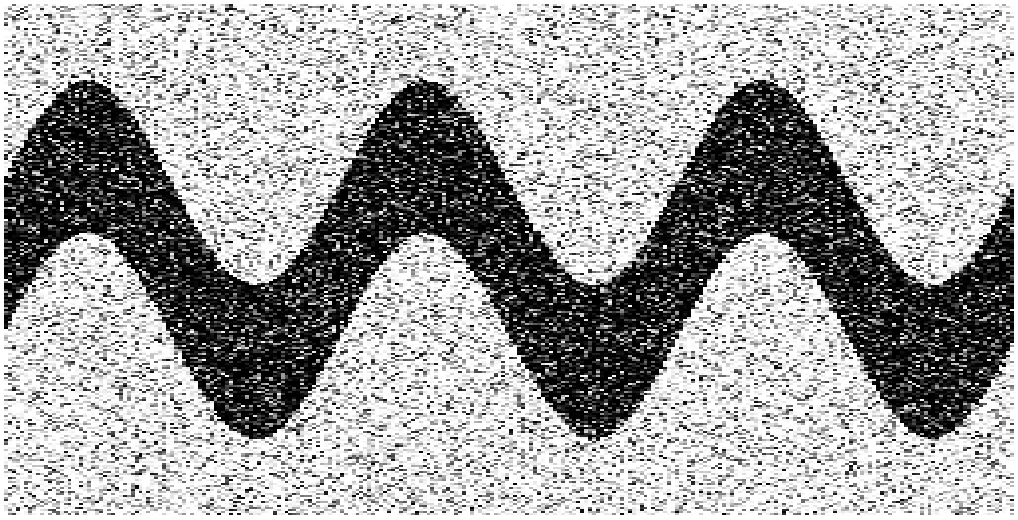
R coding part of the assignment
##Question1)

```
#fwht2d
fwht2d <- function(x) {
             h <- 1
             len <- ncol(x)
             while (h < len) {
                for (i in seq(1,len,by=h*2)) {
                   for (j in seq(i,i+h-1)) {
                     a <- x[,j]
                     b <- x[,j+h]
                     x[,j] <- a + b
                     x[,j+h] <- a - b
                     }
                }
             h <- 2*h
             }
             h <- 1
             len <- nrow(x)
             while (h < len) {
                for (i in seq(1,len,by=h*2)) {
                   for (j in seq(i,i+h-1)) {
                     a <- x[j,]
                     b <- x[j+h,]
                     x[j,] <- a + b
                     x[j+h,] <- a - b
                     }
                }
             h <- 2*h
             }
             x
             }
```

#1b&c)

```
#1b Hard and soft thresholding)
hard <- function(x, lambda){
  #hard is a function that takes in a single element and a lambda
  if (abs(x) >= lambda){

    return(x)
  }
```

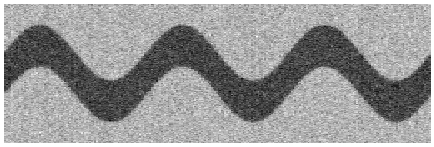
```
  else{
    return(0)
  }
}

soft <- function(x, lambda){
  #soft is a function that takes in a single element and a lambda
  if ((abs(x) - lambda)>=0){
    return(sign(x)*(abs(x) - lambda))
  }
  else{
    return(0)
  }
}

#Image is the original image.
design <- matrix(scan("design.txt"),ncol=256,byrow=T)
colours <- grey(seq(0,1,length=256))
image(design, axes=F, col=colours)
```

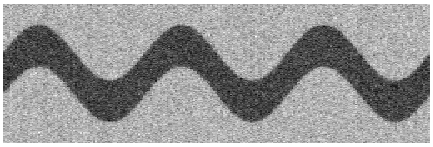


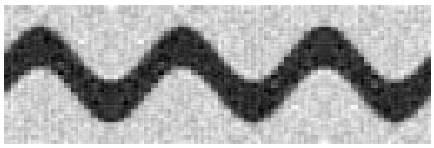
```
par(mfrow=c(2,2))

#I will make 4 denoised images using hard threshold
xhat <- fwht2d(design) # x is a 2^k x 2^k matrix
xhat_2 <- fwht2d(design)
```

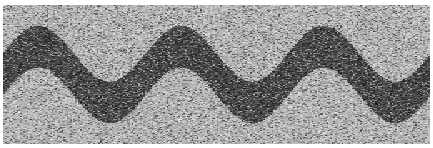
```
xhat_3 <- fwht2d(design)
xhat_4 <- fwht2d(design)

#First denoised image using Hard
#Every element in the first column will be thresholded against lambda of 0
for (x in (1:256)){
  xhat[x] <- hard(xhat[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 150
for (x in (257:length(design))){
  xhat[x] <- hard(xhat[x], 150)
}


#Second denoised image using Hard
#Every element in the first two column will be thresholded against lambda of 0
for (x in (1:512)){
  xhat_2[x] <- hard(xhat_2[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 150
for (x in (573:length(design))){
  xhat_2[x] <- hard(xhat_2[x], 150)
}


#Third denoised image using Hard
#Every element in the first column will be thresholded against lambda of 0
for (x in (1:256)){
  xhat_3[x] <- hard(xhat_3[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 300
for (x in (257:length(design))){
  xhat_3[x] <- hard(xhat_3[x], 300)
}


#Fourth denoised image using Hard
#Every element in the first column will be thresholded against lambda of 0
for (x in (1:256)){
  xhat_4[x] <- hard(xhat_4[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 75
for (x in (257:length(design))){
  xhat_4[x] <- hard(xhat_4[x], 75)
}


xhat_inverse <- fwht2d(xhat)/ncol(xhat)^2 # inverse transform
xhat_2_inverse <- fwht2d(xhat_2)/ncol(xhat_2)^2
xhat_3_inverse <- fwht2d(xhat_3)/ncol(xhat_3)^2
xhat_4_inverse <- fwht2d(xhat_4)/ncol(xhat_4)^2

#These are the images when Hard threshold is used.
```

```
image(xhat_inverse, axes=F, col=colours)
image(xhat_2_inverse, axes=F, col=colours)
image(xhat_3_inverse, axes=F, col=colours)
image(xhat_4_inverse, axes=F, col=colours)
```







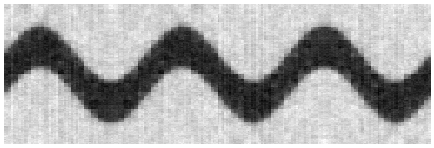
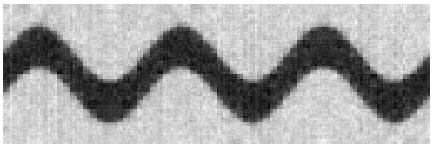
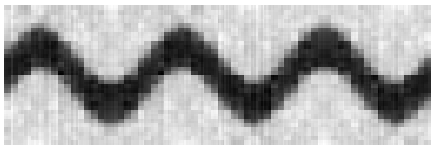
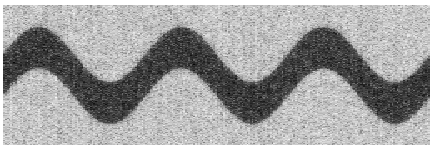


Note that the above 4 graphs used hard thresholding method.
Top left: first column lambda is 0, other columns lambda is 150
Top right: first 2 column lambda is 0, other columns lambda is 150
Bottom left: first column lambda is 0, other columns lambda is 300
Bottom right: first column lambda is 0, other columns lambda is 75

We can see that using hard threshold with lambda of around 150 is enough to denoise the image just right. Anything more or less than lambda value of 150, we get to see more blurriness in the denoised image.

```
par(mfrow=c(2,2))

#I will make 4 denoised images using soft threshold now
soft_1 <- fwht2d(design) # x is a 2^k x 2^k matrix
soft_2 <- fwht2d(design)
soft_3 <- fwht2d(design)
soft_4 <- fwht2d(design)

#First denoised image using soft
#Every element in the first column will be thresholded against lambda of 0
for (x in (1:256)){
```

```
  soft_1[x] <- soft(soft_1[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 150
for (x in (257:length(design))){
  soft_1[x] <- soft(soft_1[x], 150)
}


#Second denoised image using soft
#Every element in the first two column will be thresholded against lambda of 0
for (x in (1:512)){
  soft_2[x] <- soft(soft_2[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 150
for (x in (573:length(design))){
  soft_2[x] <- soft(soft_2[x], 150)
}


#Third denoised image using soft
#Every element in the first column will be thresholded against lambda of 0
for (x in (1:256)){
  soft_3[x] <- soft(soft_3[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 300
for (x in (257:length(design))){
  soft_3[x] <- soft(soft_3[x], 300)
}


#Fourth denoised image using soft
#Every element in the first column will be thresholded against lambda of 0
for (x in (1:256)){
  soft_4[x] <- soft(soft_4[x], 0)
}
#Then elements in other columns will be thresholded against lambda of 75
for (x in (257:length(design))){
  soft_4[x] <- soft(soft_4[x], 75)
}


soft_1_inverse <- fwht2d(soft_1)/ncol(soft_1)^2 # inverse transform
soft_2_inverse <- fwht2d(soft_2)/ncol(soft_2)^2
soft_3_inverse <- fwht2d(soft_3)/ncol(soft_3)^2
soft_4_inverse <- fwht2d(soft_4)/ncol(soft_4)^2

#These are the images when Hard threshold is used.
image(soft_1_inverse, axes=F, col=colours)
image(soft_2_inverse, axes=F, col=colours)
image(soft_3_inverse, axes=F, col=colours)
image(soft_4_inverse, axes=F, col=colours)
```

Note that the above 4 graphs used soft thresholding method.
Top left: first column lambda is 0, other columns lambda is 150
Top right: first 2 column lambda is 0, other columns lambda is 150
Bottom left: first column lambda is 0, other columns lambda is 300
Bottom right: first column lambda is 0, other columns lambda is 75

We can see that using soft threshold with lambda of around 75 is enough to denoise the image just right. Anything more or less than lambda value of 75, we get to see more blurriness in the denoised image.

I will now try the same thing but instead of hard threshold, using soft threshold.

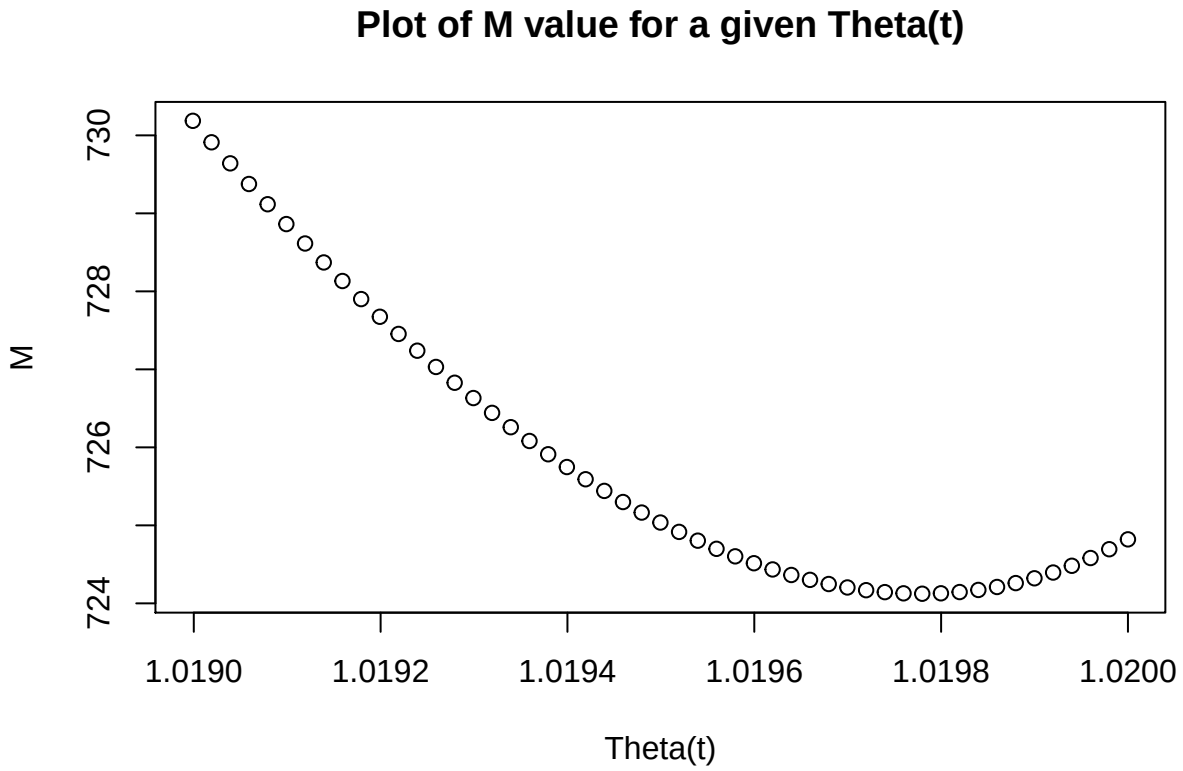
##Question 2d)

```
#Define our variables
theta = 0.9
e = 10^(-5)

#Let us first find the M value
#compute the value of t such that Phi(t) = 1/theta
upper_t = ((1/theta)*(2^10))^(1/10) - 1
lower_t = 1
#Hence (lower_t, upper_t) is the interval for Phi(t) such that 1<Phi(t)<1/theta
#So let's take a discrete set of points T such that T = {1< t_1, t_2, - - - , t_k}
#Let t_k = 1.02
t<- seq(1, 1.02, len=1000)
```

```r
#Define phi function
Phi <- function(t){
  ((1+t)^10)/(2^10)
}

#Let's compute for M
find_M <- function(t){
  M <- min((log(1 - theta) - log(1-theta*Phi(t)) - log(e))/log(t))
}
#I'll use floor function here to keep M as an integer
M <- floor(find_M(t))
print(M)
```

```
## [1] 724
```

```r
x_value <- seq(1, 1.02, len=1000)
y_value <- (log(1 - theta) - log(1-theta*Phi(x_value)) - log(e))/log(x_value)
x_new <- x_value[950:1000]
y_new <- y_value[950:1000]
plot(x_new,y_new, xlab = "Theta(t)", ylab = "M", main = "Plot of M value for a given Theta(t)")
```




```r
#using the plot we can tell that M=724 is the minimum value.
```

```
#2d
#Define distribution of{X_i}
p <- function(x){
  choose(10, x)*(1/2)^10
}
#Define DFT of {p(x) :x=0, ... , M-1}
phat <- function(j){
  x = seq(0, (M-1), 1)
  i = complex(real=0, imaginary=1)
  sum(exp(-2*pi*i*(j/M)*x)*p(x))
}

#Define g function
g <- function(x){
  (1-theta)/(1-theta*x)
}
#Define inverse FFT
PS <- function(s){
  result = 0
  i = complex(real=0, imaginary=1)
  for (j in (0:723)){
    var = exp(2*pi*i*(s/M)*j)*g(phat(j))
    result = result+var
  }
  return((1/M)*result)
}
```

#2d part 1)

```
# Evaluate the DFT
x = c(seq(0, M-1, 1))
DFT = c()
for (var in x){
  DFT_var = phat(var)
  DFT = c(DFT, DFT_var)
}

#DFT now stores phat(j) values where j = 0, --- , M-1

#DFT
```

#2d Part 2

```
# Evaluate g(phat(j))
#we already computed and stored phat(j) values in variable called DFT.
#Loop through this and calculate g(phat(j)) for j = 0, --- , M-1

gphat <- c()
for (var in DFT){
  gphat = c(gphat, g(var))
}
#gphat now stores all g(phat(j)) for j = 0, --- , M-1
#gphat
```

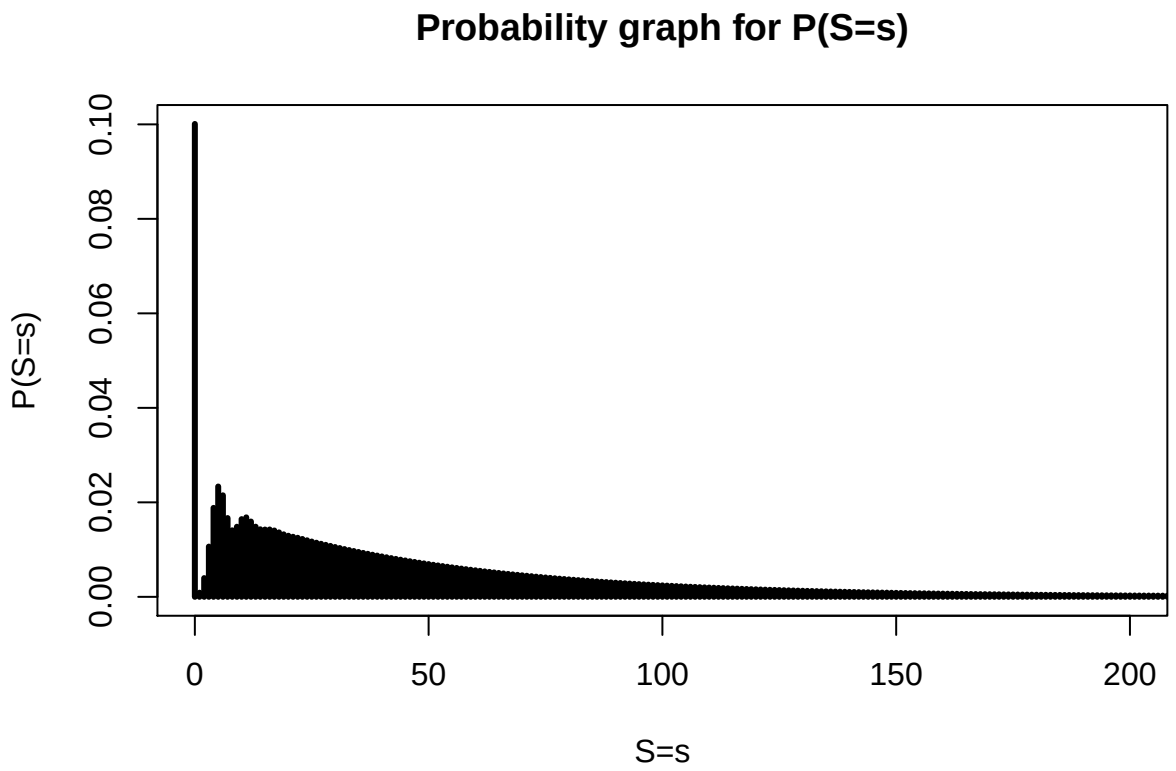
#2d Part 3

```
S = seq(0, 723, 1)
ps <- c()
for (var in S){
  ps = c(ps, PS(var))
}
#ps
#ps is the P(S=s) for s = {0,...,M-1}
#Here using sum(ps) we can see that it totals up to 1, which is what we want.
sum(ps)
```

```
## [1] 1+0i
```

```
plot(c(0:(M-1)),ps,type="h",lwd=3, xlim=c(0,200), xlab = "S=s", ylab = "P(S=s)",
     main="Probability graph for P(S=s)")
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): imaginary parts discarded in
## coercion
```



For the sake of graph, I only graphed up to s value of 200 as we can tell that P(S=s) approach 0. Nonetheless, sum of P(S=s) is 1 and this is exactly what we want.

Written part of the assignment is below.

1. Suppose that Z is an $m \times n$ pixel image where $m = 2^k$ and $n = 2^\ell$. If H_m and H_n are $m \times m$ and $n \times n$ Hadamard matrices then we can define the Walsh-Hadamard tranform of Z by

$$\hat{Z} = H_m Z H_n.$$

In this problem, we will explore using this Walsh-Hadamard transform to denoise an image.

(a) Show that $Z = H_m \hat{Z} H_n/(mn)$. (This gives the inverse Walsh-Hadamard transform.)

Proof :

Let Z = $m \times n$ matrix

Let H_m = $m \times m$ Hadamard matrix

Let H_n = $n \times n$ Hadamard matrix

Then $H_n^{-1} = \frac{H_n^T}{n}$, $H_m^{-1} = \frac{H_m^T}{m}$

So, $\hat{Z} = H_m Z H_n$

$H_m^{-1} \hat{Z} H_n^{-1} = Z$

$\frac{1}{mn} H_m^T \hat{Z} H_n^T = Z$

2. Suppose that $X_1, X_2, \cdots$ is an infinite sequence of independent identically distributed random variables with some distribution function F and N has a Geometric distribution with

$$P(N = n) = (1 - \theta)\theta^n \quad \text{for } n = 0, 1, 2, \cdots$$

(for some $0 < \theta < 1$) where N is independent of the sequence $\{X_i\}$. Then we can define a *compound Geometric* random variable S by

$$S = \sum_{i=1}^{N} X_i$$

where $S = 0$ if $N = 0$. Compound distributions arise naturally in risk theory and insurance – for example, if N represents the number of claims and $X_1, X_2, \cdots$ the amounts paid for each claim then S is the total sum paid. For the purposes of risk management, it is useful to know the distribution of S, particularly its tail.

(a) Suppose that $\{X_i\}$ are discrete integer-valued random variables with probability generating function $\phi(t) = E(t^{X_i})$. Show that the probability generating function of S is $g(\phi(t))$ where $g(t)$ is the probability generating function of N, which is given by

$$g(t) = \frac{1 - \theta}{1 - \theta t}$$

provided that $\theta t < 1$ or $t < 1/\theta$. (Hint: Write

$$E(t^S) = \sum_{n=0}^{\infty} E(t^S | N = n) P(N = n)$$

and note that given $N = n$, $S = X_1 + \cdots + X_n$.)

$E(t^S) = \sum_{n=0}^{\infty} \left(E(t^S | N=n) P(N=n) \right.$

$= \sum_{n=0}^{\infty} \left(E(t^S | N=n)(1-\theta)\theta^n \right.$

$= E(t^S | N=0)(1-\theta) + E(t^S | N=1)(1-\theta)\theta + E(t^S | N=2)(1-\theta)\theta^2 + \cdots$

$= (1-\theta) + \phi(t)(1-\theta)\theta + \phi(t)^2(1-\theta)\theta^2 + \cdots$

$= (1-\theta)\left[1 + \phi(t)\theta + \phi(t)^2\theta^2 + \phi(t)^3\theta^3 + \cdots \right]$

$= (1-\theta)\left[\sum_{i=0}^{\infty} (\phi(t)\theta)^i \right]$

$= (1-\theta)\left(\frac{1}{1-\phi(t)\theta} \right)$ By power series

$E(t^S) = \frac{1-\theta}{1-\phi(t)\theta}$

$= g(\phi(t))$

(b) The distribution of S can be approximated using the Discrete Fourier Transform. Assume that the random variables $\{X_i\}$ have a distribution $p(x)$ on the integers $0, 1, \cdots, \ell$. The complication is that, unlike the distribution of $X_1 + \cdots + X_n$, the distribution of S is not concentrated on a finite set of integers. Therefore, we need to find an integer M such that $P(S \geq M)$ is smaller than some pre-determined threshold ϵ; M will depend on the parameter θ as well as the integer ℓ (or more precisely, the distribution $p(x)$). (Also to optimize the FFT algorithm, M should be a power of 2 or a product of small prime numbers although this isn't absolutely necessary unless M is very large.)

Show that if $P(N \geq m) \leq \epsilon$ then $P(S \geq m\ell) \leq \epsilon$ and so we can take $M \geq m\ell$.

Proof.

$P(N \geq m) \leq \epsilon \Rightarrow P(S \geq m\ell) \leq \epsilon$

$P(N \geq m) = 1 - P(N < m) \leq \epsilon$

$= 1 - P(N\ell < m\ell) \leq \epsilon$

$= 1 - P(S < m\ell) \leq \epsilon$

$= P(S \geq m\ell) \leq \epsilon$

(c) The bound M determined in part (b) is typically very conservative (i.e. too large) and can be decreased substantially. One approach to determining a better bound is based on the probability generating function of S derived in part (a) and Markov's inequality. Specifically, if $\theta^{-1} > \phi(t) > 1$ (in which case $t > 1$), we have

$$P(S \geq M) = P(t^S \geq t^M) \leq \frac{E(t^S)}{t^M} = \frac{1}{t^M}\left(\frac{1-\theta}{1-\theta\phi(t)}\right).$$

Use this fact to show that for $P(S \geq M) < \epsilon$, we can take

$$M = \inf_{1<\phi(t)<\theta^{-1}} \frac{\ln(1-\theta) - \ln(1-\theta\phi(t)) - \ln(\epsilon)}{\ln(t)}.$$

WTS : For $P(S \geq M) < \varepsilon$ $\quad M = \inf_{1<\phi(t)<\frac{1}{\theta}} \frac{\ln(1-\theta) - \ln(1-\theta\phi(t)) - \ln(\varepsilon)}{\ln(t)}$

Proof :

Assume $1 < \phi(t) < \theta^{-1}$

Then,

$$P(t^S \geq t^M) \leq \frac{E(t^S)}{t^M}$$

$$= \frac{1}{t^M}\left(\frac{1-\theta}{1-\theta\phi(t)}\right)$$

Then,

$$\frac{1}{t^M}\left(\frac{1-\theta}{1-\theta\phi(t)}\right) < \varepsilon$$

$$\frac{1}{\varepsilon}\left(\frac{1-\theta}{1-\theta\phi(t)}\right) < t^M$$

$$\ln\left(\frac{1}{\varepsilon}\left(\frac{1-\theta}{1-\theta\phi(t)}\right)\right) < M\ln(t)$$

$$\ln\left(\frac{1}{\varepsilon}\right) + \ln\left(\frac{1-\theta}{1-\theta\phi(t)}\right) < M\ln(t)$$

$$\ln(1) - \ln(\varepsilon) + \ln(1-\theta) - \ln(1-\theta\phi(t)) < M\ln(t)$$

So,

$$\frac{\ln(1-\theta) - \ln(1-\theta\phi(t)) - \ln(\varepsilon)}{\ln(t)} < M$$

we would like to take the smallest M and we know $1 < \phi(t) < \theta^{-1}$

Hence let $M = \inf_{1<\phi(t)<\theta^{-1}} \frac{\ln(1-\theta) - \ln(1-\theta\phi(t)) - \ln(\varepsilon)}{\ln(t)}$ ∎