統計模擬第一次作業

統碩一 106354010 陳焌彥 統碩一 106354013 林奕志

1. (a)用**R**語言撰寫**Mid square method**函數來生成**10000**筆資料，再將生成出來的亂數利用**ks.test**檢定是否為服從均勻分配。($\alpha = 0.05$)

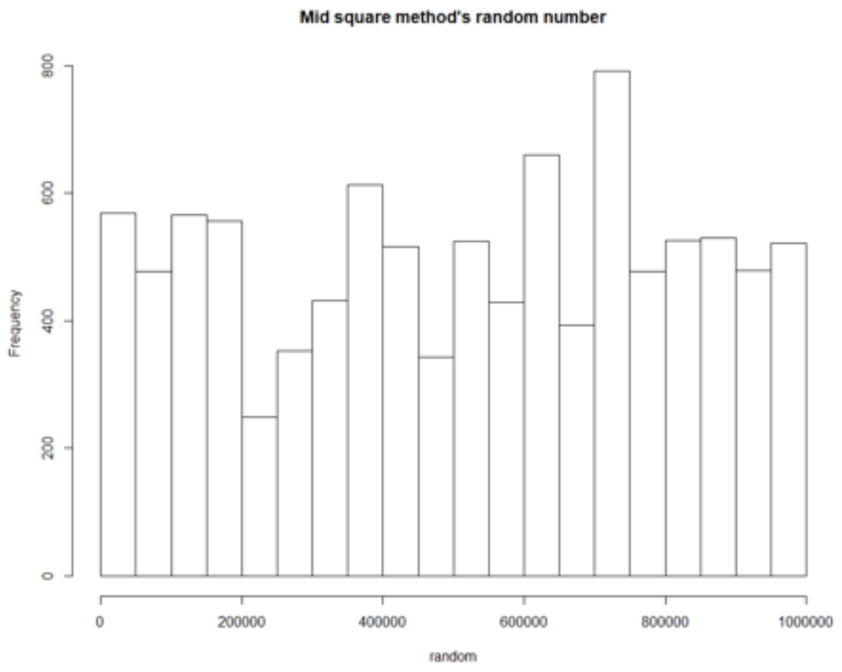
Mid square method的做法是先訂一個起始值(6位數)，接著取其平方值除以1000的商後再取其除以1000000的餘數，此時餘數會是個1~6位數的值，以此當作第一個亂數，再繼續以同樣方法產生之後的亂數，重複一萬次之後，將其作為我們要檢定的亂數。

H_0 :此數列服從均勻分配從0到1 H_1 : $not\ H_0$

程式碼與生成出來的數的直方圖如下：**(Note：Initial seed number=123457)**

——————————————————R code————————————————————

```
> midSquareRand <- function(seed, length) {
+   randvector <- NULL
+   for(i in 1:length) {
+     value <- seed * seed
+     seed <- (value %/% 1000) %% 1000000
+     randvector <- c(randvector, seed)
+   }
+   return(randvector)
+ }
> random <- midSquareRand(123457, 10000)
> hist(random,main="Mid square method's random number")
```



```
> ks.test(random , "punif")
```

```
        One-sample Kolmogorov-Smirnov test

data:  random
D = 1, p-value < 0.00000000000000022
alternative hypothesis: two-sided

Warning message:
In ks.test(random, "punif") :
  ties should not be present for the Kolmogorov-Smirnov test
```

－－－－－－－－－－－－－－－R code end－－－－－－－－－－－－－－－

由報表可知，用此檢定有一些問題。我們將**random**計算其各數列出現的次數 **(**程式：**table(random)**，發現其有重複的數。這是為什麼**K-S**檢定會出現問題。因為**K-S**檢定是利用數列的**CDF**與**Empirical function**去做比較，取距離最大的跟其臨界值相比，而連續分配**(uniform**分配**)**的**CDF**為嚴格遞增函數，所以理當不會出現相同的值。

(b)用**R**來撰寫生成μ_i的函數，**再利用K-S test及**χ^2**-test來檢定此亂數是否服從均勻分配，從0到1。(**$\alpha = 0.05$**)**

H_0 **:此數列服從均勻分配從0到1** $H_1 : not\ H_0$

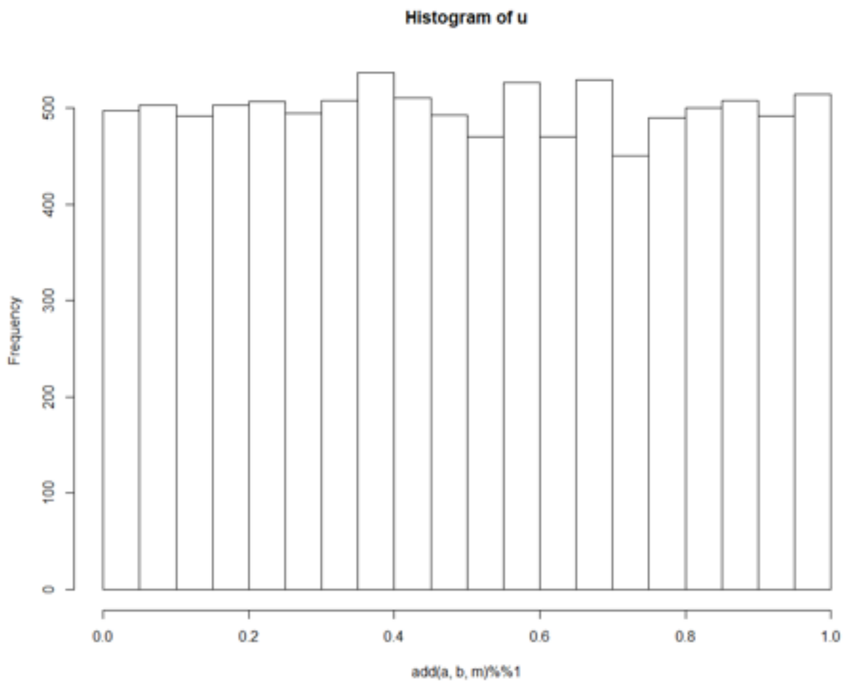
(Note：Initial seed number are x=3,y=69,z=87)

程式碼與直方圖如下：

－－－－－－－－－－－－－－－R code－－－－－－－－－－－－－－－－－

```
> add=function(x,y,z){
+   (x/30269)+(y/30307)+(z/30323)
+ }
> x=3
> a=NULL
> for(i in 1:10000){
+   x=x*171
+   x=x%%30269
+   a=c(a,x)
+ }
> y=69
> b=NULL
> for(i in 1:10000){
+   y=y*172
+   y=y%%30307
+   b=c(b,y)
+ }
```

```
> z=87
> m=NULL
> for(i in 1:10000){
+   z=z*170
+   z=z%%30323
+   m=c(m,z)
+ }
> hist(add(a,b,m)%%1,main="Histogram of u”)
```



```
> ks.test(add(a,b,m)%%1,"punif")

	One-sample Kolmogorov-Smirnov test

data:  add(a, b, m)%%1
D = 0.0064517, p-value = 0.7995
alternative hypothesis: two-sided
```

————————————————R code end————————————————

由報表，**p-value=0.7995>**$\alpha = 0.05$，不拒絕H_0，無證據顯示μ_i不服從均勻分配從**0**到**1**。
再利用χ^2**-test**來檢定此亂數是否服從均勻分配，從**0**到**1**。**(**$\alpha = 0.05$**)**

————————————————R code————————————————

```
> L=add(a,b,m)%%1
> range(L)
[1] 0.0000299915 0.9999617522
```

```
> length(L)
[1] 10000
>n1=length(which(L>0&L<0.1));n2=length(which(L>0.1&L<0.2));n3=length(wh
ich(L>0.2&L<0.3));n4=length(which(L>0.3&L<0.4));n5=length(which(L>0.4&L
<0.5))
>n6=length(which(L>0.5&L<0.6));n7=length(which(L>0.6&L<0.7));n8=length(
which(L>0.7&L<0.8));n9=length(which(L>0.8&L<0.9));n10=length(which(L>0.
9&L<1))
> teststat=(n1-1000)^2/1000+(n2-1000)^2/1000+(n3-1000)^2/1000+
(n4-1000)^2/1000+(n5-1000)^2/1000+(n6-1000)^2/1000+(n7-1000)^2/1000+
(n8-1000)^2/1000+(n9-1000)^2/1000+(n10-1000)^2/1000
> teststat<qchisq(0.95,9)#Do not reject H0，have no evidence show that L
is not a uniform distribution
[1] TRUE
```

註：我們將此數列分十組，所以=1000，。Test statistic=5.674

—————————————————R code end—————————————————

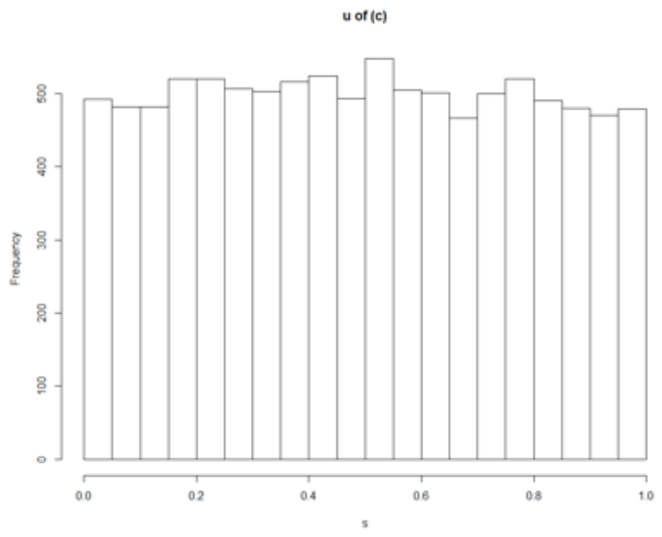
c) 用R來撰寫生成μ_i的函數，再利用χ^2-test來檢定此亂數是否服從均勻分配，從0到1。($\alpha = 0.05$)

H_0 :此數列服從均勻分配從0到1 H_1 : $not\ H_0$

(Note：Initial seed number u=10)，程式碼與直方圖如下：

—————————————————R code——————————————————

```
> add2=function(p){
+   ((pi+p)^5)%%1
+ }
> u=10
> s=NULL
> for(i in 1:10000){
+   u=add2(u)
+   s=c(s,u)
+ }
> hist(s,main="u of (c)")
```



```
> ks.test(s,"punif")

        One-sample Kolmogorov-Smirnov test

data:  s
D = 0.011754, p-value = 0.1261
alternative hypothesis: two-sided
```

————————————————R code end————————————————

由報表，**p-value=0.1261>**$\alpha = 0.05$，不拒絕H_0，無證據顯示μ_i不服從均勻分配從0到1

我們比較以上三種生成亂數的方式，以**K-S**檢定的檢定統計量為比較依據，會發現**(b)**的亂數與均勻分配從0到**1**最接近，**(c)**仍接近均勻分配從0到**1**但亂數可能沒有**(b)**的方式佳，而**(a)**的方式則最不接近均勻分配從0到**1**，因為**(a)**會有週期性，這樣對於亂數的產生非常不利。

2. 將**t**分配自由度為**10**、**20**與標準常態的每個分配都生成**10**、**50**、**100**次，再用**K-S**、**A-D**、**CVM**、**Lillie**、**Pearson**、**S-F test**分別對每個亂數做檢定，看看他們是否服從常態。

H_0 :**此數列服從常態分配從0到1** $H_1 : not\ H_0$

接下來要比各檢定的**power**；我們會對每一組資料做每一個檢定**1000**次看拒絕幾次，將拒絕次數除以**1000**，即得到**power**。程式及繪製成的表如下：

————————————————R code——————————————————

```
library(nortest)
r <- NULL
r[[1]] <- rnorm(10)
r[[2]] <- rnorm(50)
r[[3]] <- rnorm(100)
r[[4]] <- rt(10,10)
r[[5]] <- rt(50,10)
r[[6]] <- rt(100,10)
r[[7]] <- rt(10,20)
r[[8]] <- rt(50,20)
r[[9]] <- rt(100,20)

ks <- NULL
ad <- NULL
cvm <- NULL
lillie <- NULL
pearson <- NULL
```

```
sf <- NULL
p_value <- NULL
for(i in 1:9){
  ks <- ks.test(r[[i]],"pnorm")$p.value
  ad <- ad.test(r[[i]])$p.value
  cvm <- cvm.test(r[[i]])$p.value
  lillie <- lillie.test(r[[i]])$p.value
  pearson <- pearson.test(r[[i]])$p.value
  sf <- sf.test(r[[i]])$p.value
  p_value <- rbind(p_value,c(ks,ad,cvm,lillie,pearson,sf))
}
colnames(p_value)[c(1:6)] <-
     c("ks.test","ad.test","cvm.test","lillie.test","pearson.test","sf.test")
rownames(p_value)[c(1:9)] <-
     c("n=10,N(0,1)","n=50,N(0,1)","n=100,N(0,1)","n=10,t(10)","n=50,t(10)","n=10
     0,t(10)","n=10,t(20)","n=20,t(20)","n=100,t(20)")
View(p_value)
```

	ks.test	ad.test	cvm.test	lillie.test	pearson.test	sf.test
n=10,N(0,1)	0.92930	0.98311	0.99383	1.00000	0.84947	0.99862
n=50,N(0,1)	0.16816	0.29713	0.30689	0.29087	0.16702	0.33290
n=100,N(0,1)	0.66562	0.78130	0.76905	0.81625	0.16670	0.80814
n=10,t(10)	0.40290	0.35469	0.29964	0.36845	0.04601	0.42987
n=50,t(10)	0.87415	0.75226	0.79746	0.75054	0.63557	0.75141
n=100,t(10)	0.10587	0.60501	0.45205	0.44885	0.58788	0.73367
n=10,t(20)	0.74902	0.44598	0.45688	0.39985	0.13278	0.44541
n=20,t(20)	0.57114	0.81816	0.75808	0.63049	0.58715	0.84510
n=100,t(20)	0.90257	0.51371	0.58352	0.25569	0.46539	0.67922

```
#power
nfun <- function(n){
  c(ks.test(rnorm(n),"pnorm")$p.value,
    ad.test(rnorm(n))$p.value,
    cvm.test(rnorm(n))$p.value,
    lillie.test(rnorm(n))$p.value,
    pearson.test(rnorm(n))$p.value,
    sf.test(rnorm(n))$p.value)
}
tfun <- function(n,df){
  c(ks.test(rt(n,df),"pnorm")$p.value,
    ad.test(rt(n,df))$p.value,
    cvm.test(rt(n,df))$p.value,
    lillie.test(rt(n,df))$p.value,
    pearson.test(rt(n,df))$p.value,
    sf.test(rt(n,df))$p.value)
}
q01 <- c(0,0,0,0,0,0);q02 <- c(0,0,0,0,0,0);q03 <- c(0,0,0,0,0,0);q11 <-
     c(0,0,0,0,0,0); q12 <- c(0,0,0,0,0,0) ;q13 <- c(0,0,0,0,0,0) ; q21 <-
     c(0,0,0,0,0,0) ; q22 <- c(0,0,0,0,0,0) ; q23 <-c(0,0,0,0,0,0)
for(i in 1:1000){
  q01 <- q01+(1*nfun(10)<0.05)
  q02 <- q02+(1*nfun(50)<0.05)
  q03 <- q03+(1*nfun(100)<0.05)
  q11 <- q11+(1*(tfun(10,10)<0.05))
  q12 <- q12+(1*(tfun(50,10)<0.05))
  q13 <- q13+(1*(tfun(100,10)<0.05))
  q21 <- q21+(1*(tfun(10,20)<0.05))
  q22 <- q22+(1*(tfun(50,20)<0.05))
  q23 <- q23+(1*(tfun(100,20)<0.05))
}
power <-
     rbind(q01/1000,q02/1000,q03/1000,q11/1000,q12/1000,q13/10000,q21/1000,
     q22/1000,q23/1000)
colnames(power)[c(1:6)] <-
     c("ks.test","ad.test","cvm.test","lillie.test","pearson.test","sf.test")
rownames(power)[c(1:9)] <-
     c("n=10,N(0,1)","n=50,N(0,1)","n=100,N(0,1)","n=10,t(10)","n=50,t(10)","n=10
     0,t(10)","n=10,t(20)","n=20,t(20)","n=100,t(20)")
View(power)
```
————————————————R code end————————————————

	ks.test	ad.test	cvm.test	lillie.test	pearson.test	sf.test
n=10,N(0,1)	0.05	0.049	0.056	0.042	0.059	0.045
n=50,N(0,1)	0.06	0.057	0.051	0.037	0.048	0.058
n=100,N(0,1)	0.05	0.049	0.062	0.044	0.052	0.05
n=10,t(10)	0.052	0.065	0.074	0.057	0.086	0.086
n=50,t(10)	0.05	0.12	0.119	0.091	0.067	0.203
n=100,t(10)	0.0033	0.0141	0.0137	0.0113	0.0063	0.0289
n=10,t(20)	0.043	0.067	0.048	0.062	0.075	0.078
n=20,t(20)	0.049	0.069	0.085	0.063	0.05	0.119
n=100,t(20)	0.043	0.094	0.074	0.081	0.051	0.141

3. 我們選用**Excel**與**R**生成均勻分配從**0**到**1**的亂數，再用**Gap test**、**Permutation test**、**Run test** 作檢定。

Gap test的做法是先訂定一個範圍，接著看亂數有落在此範圍內的位置是多少，然後用後一項減前一項的方式觀察亂數落在範圍內與沒落在範圍內的位置間的gap，此時gap=1就是連續兩個數都落在範圍內。gap的機率分配是服從幾何分配，機率是$P(K=k)=(\beta-\alpha)*(1-(\beta-\alpha))^{k-1}, k=1,2,...$(k即為gap的大小)，接著將gap分成11組，gap1~10各自一組，11以上個數加總為一組，接著做卡方檢定，以上述的機率來算理論值，最終判斷是否拒絕抽出之亂數間是獨立的。

Run test的做法是判斷產生的亂數後一項是否大於前一項，若是大於的話就給1，反之則給0，連續k個1稱run=k(ex:111代表亂數一直變大，此時run為3)，而run test的檢定統計量為$Z=\dfrac{U-(2N-1)/3}{[(16n-29)/90]^{\frac{1}{2}}}$~N(0,1)，其中U為run的個數、N為亂數個數，若$|Z|<Z_{\alpha/2}=1.96$即不拒絕H_0：亂數間是互相獨立的。

Permutation test的做法是生成300個亂數，接著組成3x100的矩陣，然後對各個欄做rank，然後將各欄的rank做成3位數(ex:若rank之結果為3,1,2，此時三位數即為312)，這時會產生100個三位數，共3!種三位數，接著計算各種三位數之個數，此時各個三位數出現的機率應該要是一樣的，對此作卡方檢定，若檢定統計量$Q<\chi^2_{0.95}(5)$即不拒絕H_0：亂數間是互相獨立的。
看看兩組亂數是否獨立。程式碼如下：

————————————————R code——————————————————

```
 > set.seed(87)
> unif <- read.csv("~/unif.csv",header=F)#Excel亂數存的地方
> en=unif[,1]#Excel生成的亂數
> rn=runif(1000)#R生成的亂數
> set.seed(87)#Gap test對rn
> y=runif(2)
> z=which(rn>y[1]&rn<y[2])
> z1=z[-1]-z[-length(z)]
> zz=table(z1)
> sum(zz)
[1] 265
> p=max(y)-min(y)
> ad=function(k){
+   p*((1-p)^(k-1))
+ }
> z=c()
> for(i in 1:16){
+   z[i]=ad(i)
+ }
> z1=z*265
> t=c(z1[1:10],sum(z1[11:16]))
> o=c(zz[1:10],sum(zz>10)/1000)
> tt=sum((o-t)^2/t)
> tt<qchisq(0.95,10) # Do not reject H0，have no evidence show
that the series(generated by r) are not independent
[1] TRUE
> set.seed(87)
#Gap test 對en
> y=runif(2)
> z=which(en>y[1]&en<y[2])
> z1=z[-1]-z[-length(z)]
> zz=table(z1)
> sum(zz)
[1] 289
> p=max(y)-min(y)
> ad=function(k){
+   p*((1-p)^(k-1))
+ }
> z=c()
> for(i in 1:16){
+   z[i]=ad(i)
+ }
> z1=z*sum(zz)
> t=c(z1[1:10],sum(z1[11:16]))
> o=c(zz[1:10],sum(zz>10)/1000)
> tt=sum((o-t)^2/t)
> tt<qchisq(0.95,10) # Do not reject H0，have no evidence show
that the series(generated by r) are not independent
[1] TRUE
> y1 <- 1*(rn[-1]>rn[-1000] ) #Run test 對rn
> u=sum(y1[-1] != y1[-999]) + 1 # 共有幾個run
> y2 <- which(y1[-1] != y1[-999]) #run的位置
> y3 <- table(y2[-1]-y2[-length(y2)]) #看run＝n的個數 N=5
> sum(y3)
[1] 646
```

```
> z=function(u,n){
+   (u-(2*n-1)/3)/(((16*n-29)/90)^(0.5))
+ }
> z(sum(y3),1000)>1.96*(-1) #Do not reject H0，have no evidence
show that series rn is not independent
[1] TRUE
> y1 <- 1*(en[-1]>en[-1000] ) #Run test 對en
> u=sum(y1[-1] != y1[-999]) + 1 # 共有幾個run
> y2 <- which(y1[-1] != y1[-999]) #run的位置
> y3 <- table(y2[-1]-y2[-length(y2)]) #看run＝n的個數 N=5
> sum(y3)
[1] 675
> z=function(u,n){
+   (u-(2*n-1)/3)/(((16*n-29)/90)^(0.5))
+ }
> z(sum(y3),1000)<1.96 #Do not reject H0，have no evidence show
that series en is not independent
[1] TRUE
> set.seed(87)#Permutation test 對rn
> a=matrix(runif(300),nrow=3)
> t=apply(a,2,rank)
> z=c()
> for(i in 1:100){
+   z[i]=t[1,i]*100+t[2,i]*10+t[3,i]
+ }
> table(z)
z
123 132 213 231 312 321
 18  25  13  16  17  11
> ti=100/6
> test=(18-ti)^2/ti+(25-ti)^2/ti+(13-ti)^2/ti+(16-ti)^2/ti+(17-ti)^2/
ti+(11-ti)^2/ti
> test<qchisq(0.95,5)#alpha=0.05，Do not reject H0 ，have no
evidence show that the series are not independent
[1] TRUE
> newen=en[1:300] #Permutation test 對en
> a=matrix(newen,nrow=3)
> t=apply(a,2,rank)
> z=c()
> for(i in 1:100){
+   z[i]=t[1,i]*100+t[2,i]*10+t[3,i]
+ }
> table(z)
z
123 132 213 231 312 321
 18  15  15  13  24  15
> ti=100/6
> test=(18-ti)^2/ti+(15-ti)^2/ti+(15-ti)^2/ti+(13-ti)^2/ti+(24-ti)^2/
ti+(15-ti)^2/ti
> test< qchisq(0.95,5)#alpha=0.05，Do not reject H0 ，have no
evidence show that the series are not independent
[1] TRUE
```

——————————————R code end——————————————

Gap test	Chi-square test statistic
R random numbers	17.848
Excel random numbers	17.019

Run test	Z test statistic
R random numbers	-1.526384
Excel random numbers	0.605

Permutation test	Chi-square test statistic
R random numbers	7.04
Excel random numbers	4.64

根據以上結果，經過三種獨立性檢定，可以看出**excel**的亂數較為隨機。

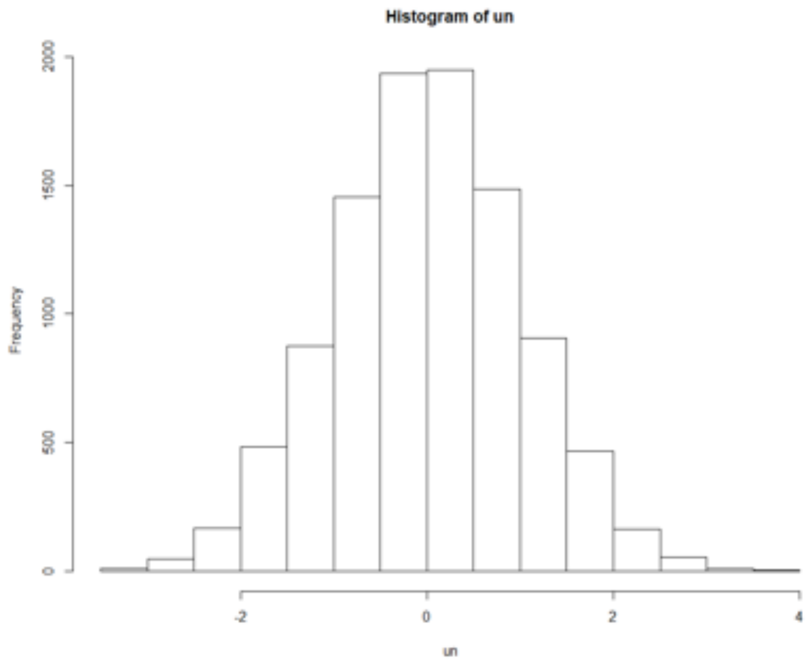
4. 可以來模擬**Normal (0, 1)**，$u_{ni} = \sum_{i=1}^{12} u_i - 6$，**再利用K-S test及$\chi^2$—test來檢定此亂數是否服常態分配。($\alpha = 0.05$)**

H_0 **:此數列服從常態分配從0到1** H_1 : $not\ H_0$

用**R**程式寫函數來生成u_n，程式碼及u_n的直方圖如下：

R code

```
> set.seed(87)
> un=c()
> for(i in 1:10000){
+ r=runif(12,0,1)
+ un[i]=sum(r)-6}
> hist(un)
```

```
>
```


```
ks.test(un,"pnorm")

        One-sample Kolmogorov-Smirnov test

data:  un
D = 0.0087485, p-value = 0.4284
alternative hypothesis: two-sided
```

——————————————————R code end——————————————————

由報表，**p-value=0.4284<**$\frac{\alpha}{2}$，不拒絕H_0，無證據顯示u_i不服從常態分配。

——————————————————R code ————————————————————

```
> range(un)
[1] -3.445401  3.718967
> f=function(c){
+   length(which(un<=c&un>c-1))
+ }
> z=c()
> for(i in 1:8){
+   z[i]=f(i-4)
+ }
> z
[1]    8  215 1359 3386 3433 1373  215   11
> p1=pnorm(-2)-pnorm(-3);p2=pnorm(-1)-pnorm(-2);p3=pnorm(0)-pnorm(-1)
> p4=pnorm(1)-pnorm(0);p5=pnorm(2)-pnorm(1);p6=pnorm(3)-pnorm(2)
> p7=pnorm(4)-pnorm(3);p8=pnorm(5)-pnorm(4)
> pp1=10000*p1;pp2=10000*p2;pp3=10000*p3;pp4=10000*p4;pp5=10000*p5;pp6=10000*p6
> pp7=10000*p7;pp8=10000*p8
> chi=(z[1]+z[2]-pp1)^2/pp1+(z[3]-pp2)^2/pp3+(z[4]-pp3)^2/pp3+(z[5]-pp4)^2/pp4+(z[6]-pp5)^2/pp5+
(z[7]+z[8]-pp6)^2/pp6
> chi<qchisq(0.95,5)#Do not reject Ho，no evidence show that un is not a normal distribution
[1] TRUE
```

——————————————————R code end——————————————————

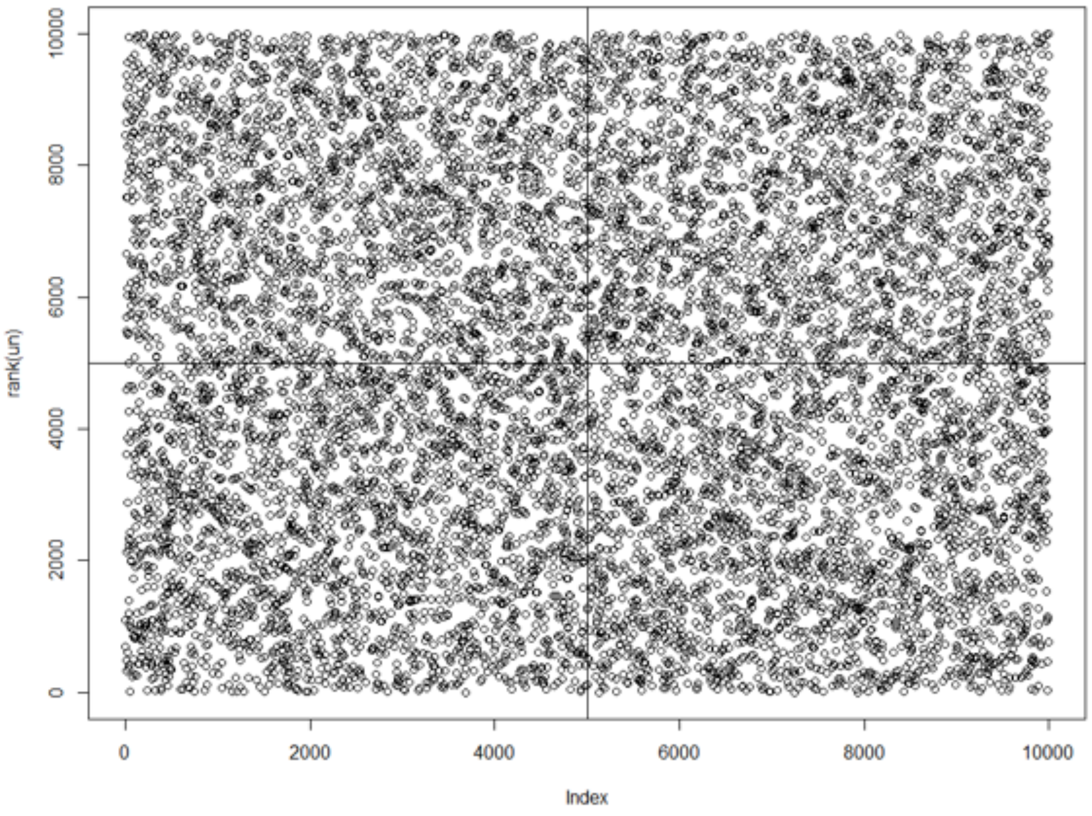
註：我們將此數列分六組(次數太少者合併)。**Test statistic=1.5268**

檢定的結論皆為不拒絕H_0，無證據顯示此亂數不為常態分配。

(b)我們將設計兩種檢定方法，第一種是將亂數做排序(rank)，再將之繪製其散步圖，x軸為第i個數，y軸為第i個的rank。將散佈圖切成四塊，再做卡方檢定。以下為程式範例模擬：(un為(a)的亂數，有10000個數)

——————————————————R code ——————————————————

```
> plot(rank(un));abline(v=5000);abline(h=5000);a=rank(un)
>
a1=length(which(a[1:5000]>5000));a2=length(which(a[1:5000]<5000));a3=length(which
(a[5001:10000]>5000));a4=length(which(a[5001:10000]<=5000))
>
test=(2480-2500)^2/2500+(2520-2500)^2/2500+(2520-2500)^2/2500+(2480-2500)^2/25
00 #Perform Chi-square test
> test<qchisq(0.95,3)#Do not reject H0，have no evidence   show that series un is not
independent
[1] TRUE
```



↑un的rank的散佈圖，若為隨機，每塊理論上會有2500個
註(規定)：第5000的rank歸在第四塊(右下角)

————————————————R code end————————————————

我們將其稱作**Lin-Yen's Test**
推廣：也能將多維度的資料映射到二度空間，在做卡方檢定來看看多維資料是否隨機。而多維資料是將每一資料取馬氏距離(Mahalanobis distance)的rank，再繪製散佈圖。

第二種方法是將其亂數取**rank(**總共**10000**個**rank)**，依序輸進每列，形成一**42500**的矩陣。再將每行取**rank**，將第一行的**rank**乘**1000**加第二行的**rank**乘**100**加第三行的**rank**乘**10**加第四行的**rank**。會得到**2500**個值，將其**table**化，算得其每個值出現的次數。若為隨機，其每個值出現的次數為，最後以卡方檢定檢定之。我們稱其為**rank-permutation test**。程式碼如下：
————————————————R code ————————————————

```
> a=rank(un);DD=apply((matrix(a,ncol=2500)),2,rank)
> z=c()
> for(i in 1:2500){
+   z[i]=DD[1,i]*1000+DD[2,i]*100+DD[3,i]*10+DD[4,i]
+ }
> sum((table(z)-ttt)^2/ttt)<qchisq(0.95,23)#Do not reject H0，have no evidence show that series un is not independent
[1] TRUE
```

————————————————R code end————————————————

5.　Bisection method的做法是先設定兩個起始點，函數值必須是一正一負，最好是先看過圖之後選定在根附近的值，之後求兩點之中點，看此點知函數值正還負，若是正的就替代原本還數值為正的點，反之亦然，重複此方法幾次之後會越來越接近函數值為0的點，接著訂定一個可接受誤差，當函數值與0之誤差小於此誤差即停止迭代，最後的點即為解。

False positions method的做法是先設定兩個起始點，函數值必須是一正一負，最好是先看過圖之後選定在根附近的值，接著將兩點連線，將其中一點替換成切到的x軸的x對應到圖形上的點，接著繼續連兩點，替換已經變動的那點，重複上述動作直到函數值與0之差距小於可接受誤差，最後的點即為解。

Secant method的做法是先設定兩個起始點，接著將兩點連線，將其中一點替換成切到的x軸的x對應到圖形上的點，繼續將兩點連線，替換上一步未變動的點，重複上述動作，直到兩點的函數值之差距小於誤差點就停止，最後的點即為解。

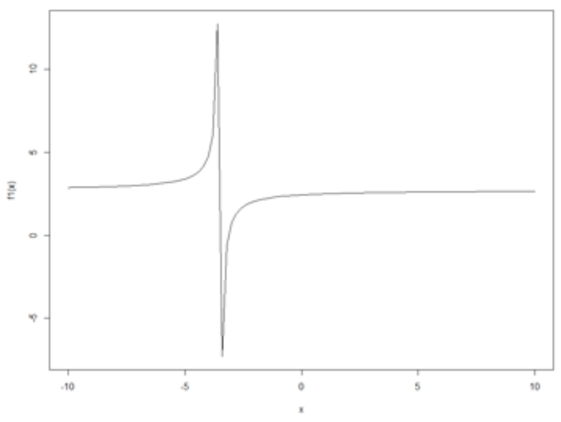
我們先寫出**bisection**、**false positions**、**secant methods**的程式。程式如下：

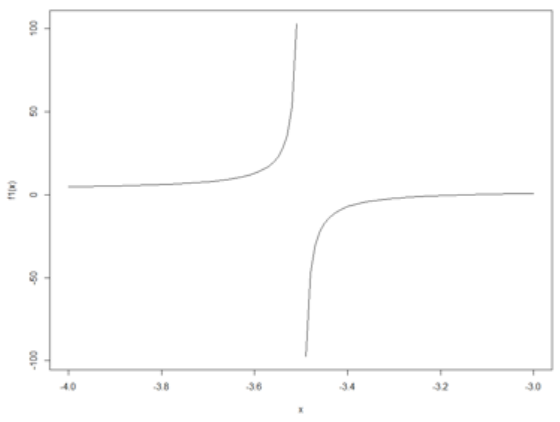
——————————————————R code ———————————————————

```
> bis <- function(f,a,b){
+   z <- abs(b-a)
+   i <- 0
+   print(z)
+   while (z > 0.000001) {
+     c <- (a+b)/2
+     ifelse(f(c)<0, a <- c , b <- c)
+     z <- abs(b-a)
+     i <- i+1+
+   }
+   print(c(c,i))
+ }
> fal <- function(f,a,b){
+   z <- abs(b-a)
+   i <- 0
+   while(z >0.000001){
+     c <- b-(f(b)*((b-a)/(f(b)-f(a))))
+     if(f(b)*f(c)>0) b <- c
+     z <- f(b)
+     i <- i+1
+   }
+   print(c(c,i))
+ }
> sec <- function(f,a,b){
+   z <- abs(f(b)-f(a))
+   i <- 0
+   while (z > 0.000001) {
+     c <- b-(f(b)*((b-a)/(f(b)-f(a))))
+     a <- b
+     b <- c
+     z <- abs(f(b)-f(a))
+     i <- i+1
+     +   }
+   print(c(c,i))
+}
```

——————————————————R code end————————————————

(a) 先來看看函數的圖形，右邊那張比較清楚





可以看出根大致落在**-3.2**到**-3**之間，所以我們的起始點選用**-3.2**、**-3**，收斂準則為當疊代後的兩個函數值相減小於則停止疊代。程式碼如下：

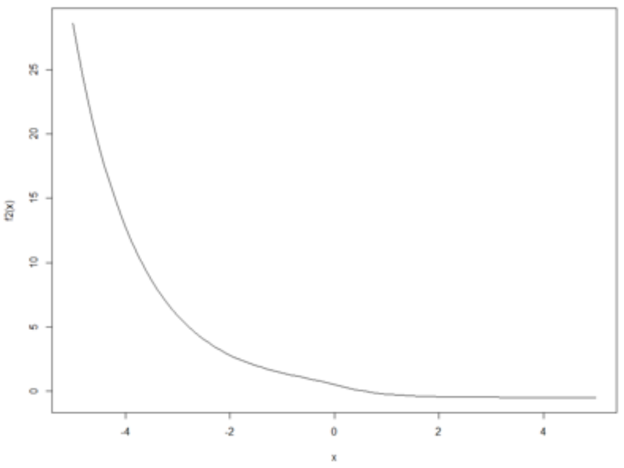
——————————————————R code ——————————————————

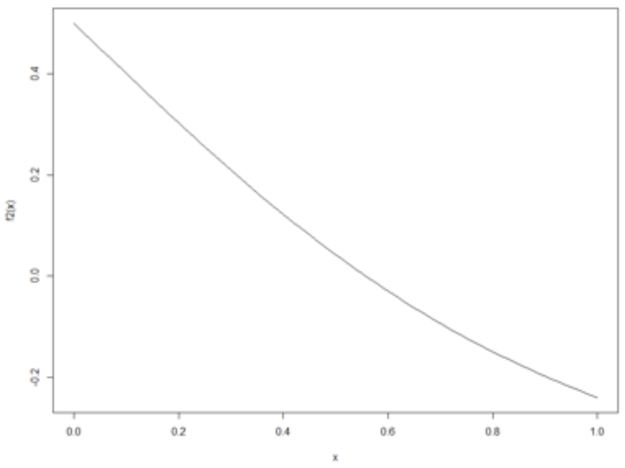
```
> f1=function(x){
+   exp(1)-(1/(3.5+x))
+ }
> curve(f1,-10,10)
> curve(f1,-4,-3)
> uniroot(f1,c(-3.2,-3))
$root
[1] -3.132115
$f.root
[1] 0.00004459005
$iter
[1] 4
$init.it
[1] NA
$estim.prec
[1] 0.00006103516
> bis(f1,-3.2,-3)
[1] 0.2
[1] -3.132121 18.000000
> fal(f1,-3.2,-3)
[1] -3.132121  9.000000
> sec(f1,-3.2,-3)
[1] -3.132121  6.000000
```

——————————————————R code end——————————————————

	root	iterations
uniroot	-3.132115	4
bisection	-3.132121	18
false positions	-3.132121	9
secant	-3.132121	6

(b) 先來看看函數的圖形，右邊那張比較清楚





可以看出根大致落在**0.4**到**0.6**之間，所以我們的起始點選用**0.4**、**0.6**，收斂準則為當疊代後的兩個函數值相減小於則停止疊代。程式碼如下：

———————————————R code ——————————————————

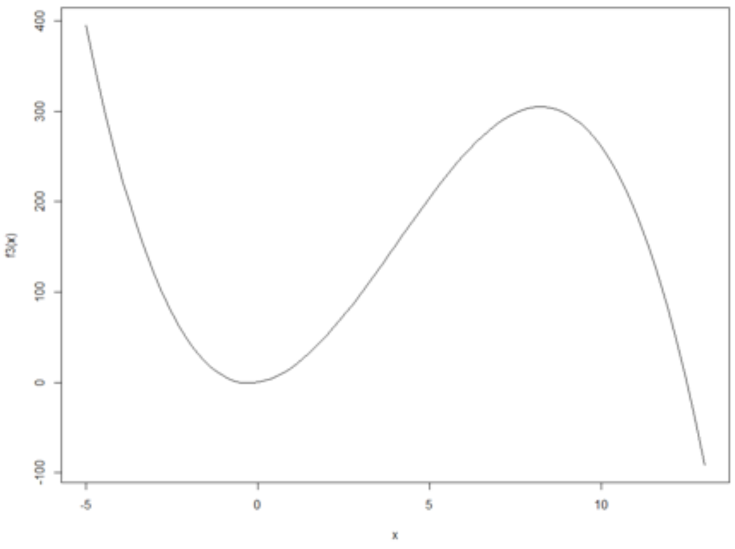
```
> f2=function(x){
+   exp(-x)/((1+x^2)^(0.5))-0.5
+ }
> curve(f2,-5,5)
> curve(f2,0,1)
> uniroot(f2,c(0,1))
$root
[1] 0.5577421
$f.root
[1] -0.000009543937
$iter
[1] 5
$init.it
[1] NA
$estim.prec
[1] 0.00006103516
> bis(f2,0.6,0.4)
[1] 0.2
[1]  0.5577293 18.0000000
```

```
> fal(f2,0.4,0.6)
[1] 0.5612611 1.0000000
> sec(f2,0.4,0.6)
[1] 0.5577287 4.0000000
```

————————————————————R code end————————————————————

	root	iterations
uniroot	0.5577421	5
bisection	0.5577293	18
false positions	0.5612611	1
secant	0.5577287	4

(c) 我們知道，**eigenvalue**的求法為：$det(A-\lambda I)=0$， I 是單位矩陣。
可求得一個三次方程式，進而利用**R**的函數求根。**(**此矩陣為一不可逆矩陣**)**
此函數圖形如下：



可以看出根大致落在**0.4**到**0.6**之間，所以我們的起始點選用**0.4**、**0.6**，收斂準則為當疊代後的兩個函數值相減小於則停止疊代。程式碼如下：
(因為有**3**個根，所以每個方法根據根大致落的範圍做三次，三個範圍分別為**(12,13)**、**(-0.5,-0.4)**、**(-0.4,0))**

— — — — — — — — — — — — — — —R code — — — — — — — — — — — — — — — — — —

```
> f3=function(x){
+   (2-x)*(4-x)*(6-x)+60+60-16*(4-x)-9*(6-x)-25*(2-x)
+ }
```

```
> curve(f3,-5,13)
> eigen(matrix(c(2,3,4,3,4,5,4,5,6),ncol=3))
$values
[1] 12.480740698407867483866 -0.000000000000000721645
[3] -0.480740698407859934349
$vectors
         [,1]       [,2]       [,3]
[1,] 0.4303622  0.4082483  0.8050601
[2,] 0.5665422 -0.8164966  0.1111905
[3,] 0.7027221  0.4082483 -0.5826791
> uniroot(f3,c(12,13));uniroot(f3,c(-0.4,-0.5));uniroot(f3,c(-0.4,0))
$root
[1] 12.48074
$f.root
[1] 0.0004214783
$iter
[1] 4
$init.it
[1] NA
$estim.prec
[1] 0.00006103516
$root
[1] -0.4807408
$f.root
[1] 0.0000006237919
$iter
[1] 4
$init.it
[1] NA
$estim.prec
[1] 0.00006103516
$root
[1] 0
$f.root
[1] 0
$iter
[1] 0
$init.it
[1] NA
$estim.prec
[1] 0
> bis(f3,-0.5,-0.4);bis(f3,-0.4,0);bis(f3,13,12)
```

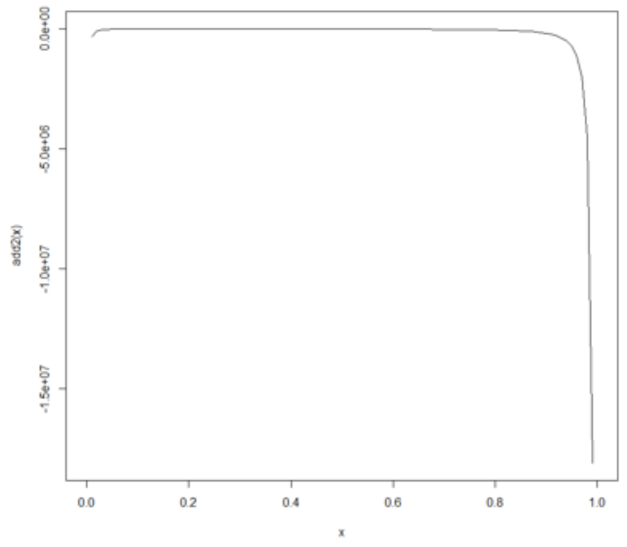
```
[1] 0.1
[1] -0.4000008 17.0000000
[1] 0.4
[1] -0.0000007629395 19.0000000000000
[1] 1
[1] 12.48074 20.00000
> fal(f3,-0.5,-0.4);fal(f3,-0.4,0);fal(f3,13,12)
[1] -0.4768946  1.0000000
[1] 0 1
[1] 12.48074  8.00000
> sec(f3,-0.5,-0.4);sec(f3,-0.4,0);sec(f3,13,12)
[1] -0.4807407  5.0000000
[1] 0 1
[1] 12.48074  6.00000
```

————————————R code end————————————

	root	iterations
uniroot	12.48074 -0.4807408 0	4 4 0
bisection	12.48074 -0.4000008 -0.0000007629395	20 17 19
false positions	12.48074 -0.4768946 0	8 1 1
secant	12.48074 -0.4807407 0	6 5 1
eigen	12.480740698407867483866 -0.480740698407859934349 -0.000000000000000721645	No information

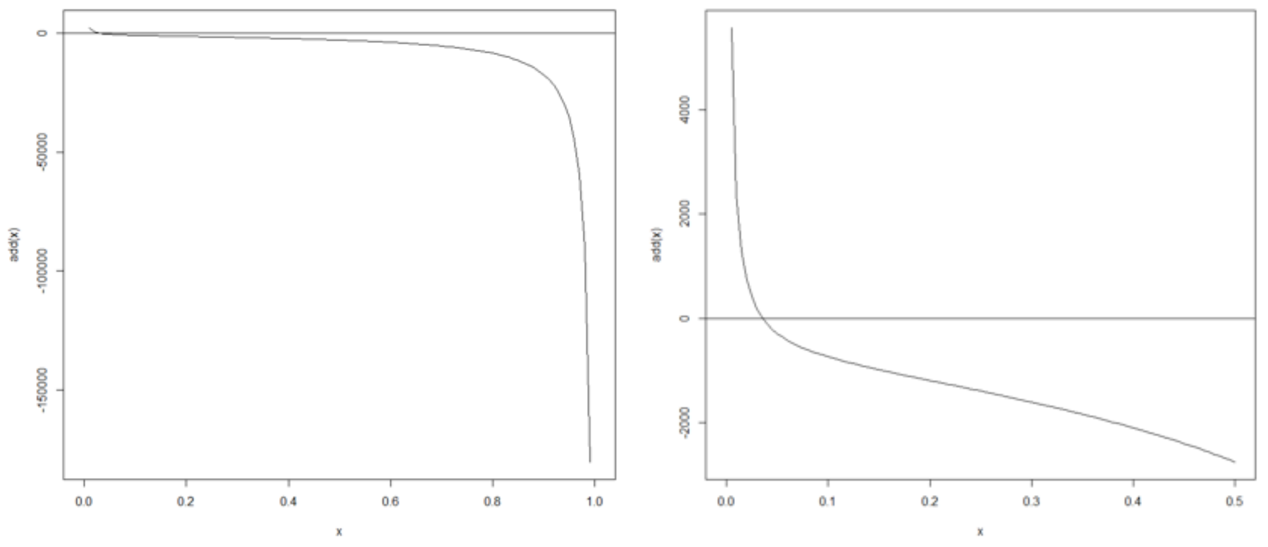
6. 我們先建立$f(\underline{x}\,|\,\theta)$的概似函數，$L(\theta\,|\,\underline{x})$，對概似函數微分，記$S(\theta\,|\,\underline{x})$。對$S(\theta\,|\,\underline{x})$微分，記$C(\theta\,|\,\underline{x})$。另$S(\theta\,|\,\underline{x})=\mathbf{0}$，求其$\hat{\theta}$等於多少，再代入$C(\theta\,|\,\underline{x})$看看會不會小於**0**，若小於**0**，$\hat{\theta}$即為$\theta$的**M.L.E**。經過運算，可知

$\frac{1997}{2+\theta} - \frac{906}{1-\theta} - \frac{904}{1-\theta} + \frac{32}{\theta} = 0$，求出的$\hat{\theta}$即為我們要求的**M.L.E**。**(**$\hat{\theta}$代到$C(\theta\,|\,\underline{x})$
小於**0)**



↑為一恆負函數

看看$h(\theta) = \frac{1997}{2+\theta} - \frac{906}{1-\theta} - \frac{904}{1-\theta} + \frac{32}{\theta}$的圖形：**(**右邊那張比較清楚**)**



可以看出根大致落在$0 \le \hat{\theta} \le 0.1$，所以我們的起始點選用**0.01**到**0.05(**牛頓法選用**0.05**當作起始值**)**，收斂準則為當疊代後的兩個函數值相減小於10^{-7}則停止疊代。
三種方式的程式碼如下：

R code end

```
> options(scipen=999)
> library(pracma)
> add=function(a){
+   (1997/(2+a))-(906/(1-a))-(904/(1-a))+(32/(a))
+ }
> curve(add,0,1);abline(h=0)
> curve(add,0,0.5);abline(h=0)
> add2=function(a){
+   (-1997/(2+a)^2)-(1810/(1-a)^2)-(32/a^2)}
> curve(add2,0,2)
> ridders(add, 0.01, 0.05, maxiter = 50, tol = 10^(-7))#ridders method
$root
[1] 0.0357123
$f.root
[1] -0.00000000009629275
$niter
[1] 10
$estim.prec
[1] 0.00000001356312
> sec <- function(f,a,b){
+   z <- abs(f(b)-f(a))
+   i <- 0
+   while (z > 10^(-7)) {
+     c <- b-(f(b)*((b-a)/(f(b)-f(a))))
+     a <- b
+     b <- c
+     z <- abs(f(b)-f(a))
+     i <- i+1
+   }
+   print(c(c,i))
+ }
> sec(add,0.01,0.05)#secant method
[1] 0.0357123 8.0000000
> newton=function(f,fp,start){
+   i=0
+   new=start
+   r=c(i,new,f(new))
+   while(abs(f(new))>10^(-7)){
+     i=i+1
+     new=new-f(new)/fp(new)
+     r=rbind(r,c(i,new,f(new)))}
+     r}
> newton(add,add2,0.05)#newton method
```

```
  [,1]       [,2]                  [,3]
r    0 0.05000000 -291.116816431322149583
     1 0.03094877  149.444862911129234817
     2 0.03512080   16.526888832958775311
     3 0.03570337    0.245996883166412772
     4 0.03571230    0.000055976147677939
     5 0.03571230    0.000000000003069545
```

R code end

整理成一表格如下：

	M.L.E	疊代次數
Ridders Method	0.0357123	10
Secant Method	0.0357123	8
Newton Method	0.03571230	5