รายการอ้างอิง

ภาษาไทย

มานพ วราภักดิ์. การจำลองเบื้องต้น. กรุงเทพมหานคร : ศูนย์ผลิตตำราเรียนสถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ, 2547.

มานพ วราภักดิ์. ทฤษฎีความน่าจะเป็น. กรุงเทพมหานคร : สำนักพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย , 2548.

วีรวรรณ ศักดาจิวะเจริญ. ช่วงความเชื่อมั่นสำหรับค่าเฉลี่ยของประชากรที่มีการแจกแจงแบบเบ้ขวา. วิทยานิพนธ์ปริญญามหาบัณฑิต. ภาควิชาสถิติ. คณะพาณิชยศาสตร์และการบัญชี. จุฬาลงกรณ์มหาวิทยาลัย, 2544.

ภาษาอังกฤษ

Angus, J. E. (1994). 'Bootstrap one-sided confidence intervals for the log-normal mean', Statistician, 43:395-401.

Land, C. E. (1972). 'An evaluation of approximate confidence interval estimation methods for lognormal means'.Technometrics, 14:145-158.

Land, C. E. (1971). 'Confidence intervals for linear functions of the normal mean and variance', Annals of Mathematical Statistics, 42:1187-1205.

Patterson, R. L. (1966). 'Difficulties involved in the estimation of a population mean using transformed sample data', Technometrics, 8:535-537.

Zhou, X. H. and Gao, S. J. (1997). 'Confidence intervals for the log-normal mean', Statistics in Medicine., 16:783-790.

ภาคผนวก

โปรแกรมแสดงการประมาณค่าแบบช่วงสำหรับค่าเฉลี่ยของการแจกแจงแบบล็อกนอร์มอล ด้วยวิธีการประมาณของคอกซ์ (C) วิธีการประมาณแบบคอนเซอเวทีฟ (S) และวิธีการประมาณแบบพาราเมตริกบูตสแตรป (B)

```
program ConfidentInt;
{$apptype console}

uses
  Forms, Math;
const round=1000;
      maxn=100;
      mu=1;
type data = array[1..maxn] of single;
     Bound=array[1..round,1..2] of single;
var iround,ir,jr,seed : integer;
    n : integer;
    x : data;
    variance,t0,t1,Theta,alfa,Cvalue,Zalfa,tvalue,chi : single;
    naiveBound,coxBound,ConservativeBound,BootstrapBound : Bound;
    printout : text;

function Xrnd(var x : LongInt) : double     ;    {Uniform 0-1}
Const a = 16807 ;
Const q = 127773 ;
Const r = 2836;
Const m = 2147483647;
begin
  x := (a*x);
```

```
 if x < 0 then
   x := (x+m)+1;
 Xrnd := x/m;
end; {end Xrnd}

function Normal(m : double ; v : double) : double        ; {Normal}
var u1,u2,v1,v2,s,N01 : double;
begin
  repeat
    u1 := xrnd(seed);
    u2 := xrnd(seed);
    v1 := 2*u1-1;
    v2 := 2*u2-1;
     s := (v1*v1)+(v2*v2);
  until s<=1; {end repeat}
N01 := sqrt((-2*ln(s))/s)*v1;
Normal := m+(SQRT(v)*N01);
end; {end Normal}

function Mean(dat : data) : single;
var   i : integer;
      sum : single;
begin
sum:=0;
for i:=1 to n do
   sum := sum+dat[i];
Mean := sum/n;
```

```
end; {end Mean}

function Variant(dat : data) : single;
var   i : integer;
      sum,M : single;
begin
sum:=0;
M:=Mean(dat);
for i:=1 to n do
   sum := sum+SQR(dat[i]-M);
Variant := sum/(n-1);
end; {end Mean}

function CalP(B : Bound) : single;   {Calculate Power}
var   i : integer;
      itemcount : integer;
begin
itemcount := 0;
for i:=1 to round do begin
   if (B[i,1]<Theta) and (B[i,2]>Theta) then
      itemcount := itemcount+1
end ; {end i}
CalP := itemcount/round
end;{end Cal P}

function MeanDiff(B : Bound) : single;   {Mean of different lower/upper bound}
var   i : integer;
```

```
    sum : single;
begin
sum := 0;
for i:=1 to round do
  sum := sum+(B[i,2]-B[i,1]);
MeanDiff := sum/round;
end; {end Mean Diff Lower-Upper bound}

procedure Cox(dat : data) ;
var   i : integer;
    y : data;
    ybar,S,A,B : single;
begin
for i:=1 to n do
  y[i] := ln(dat[i]);
ybar := Mean(y);
S := Variant(y);
A := ybar+(S/2);
B := SQRT( (S/n)+(SQR(S)/(2*n-2)) );
//writeln(A:4:4,'  ',B:4:4);readln;

CoxBound[iround,1] := exp(A-Zalfa*B);
CoxBound[iround,2] := exp(A+Zalfa*B);
end; { end Naive }

procedure Conservative(dat : data) ;
var   i : integer;
```

```
    y : data;
    ybar,S,A,B,qvalue : single;
begin
qvalue :=SQRT ( (n/2)*( ((n-1)/chi) - 1  ) );
for i:=1 to n do
  y[i] := ln(dat[i]);
ybar := Mean(y);
S := Variant(y);
A := ybar+(S/2);
B := SQRT( S*(1+S/2) );
//writeln('t & q ',tvalue:2:2,'  ',qvalue:2:2);
ConservativeBound[iround,1] := exp(  A-(tvalue/SQRT(n))*B );
ConservativeBound[iround,2] := exp(  A+(qvalue/SQRT(n))*B );
end; { end Naive }

function SingToInt(x : Single) : Integer;  {Tranform Single to Integer}
var i : integer;
   Run : single;
begin
i := 0;
Run := 0;
if abs(Int(x)-x)>=0.5 then
  Run := Int(x)+1
else
  Run := Int(x);
while i <> Run do
  i := i+1;
```

```
SingToInt := i;
end; { end singtoint}

procedure GenerateBoots(dat : data);  {Find t0 , t1}
const maxBoots=2000;
type Tval = array[0..maxBoots] of single;
var y : data;
   T : Tval;
   i,j,index : integer;
   N01,Chi,SumN01,Sigma,A,B,TmpT,d : single;
begin
try
for i:=1 to n do
  y[i] := ln(dat[i]);
Sigma := SQRT(Variant(y));

for i:=1 to maxBoots do begin
 N01 := Normal(0,1);

 SumN01 := 0;
 for j:=1 to n-1 do  {degree of freedom n-1}
   SumN01 := SumN01 + SQR(Normal(0,1));

 Chi := SumN01;
 A := N01+Sigma*(SQRT(n)/2)*( (Chi/(n-1))-1   );
 B := SQRT( (Chi/(n-1))*(1+(SQR(Sigma)/2)*(Chi/(n-1))  ) );
 T[i] := A/B;
```

```
  //Writeln('T ',i,' ',T[i]:4:4);

end; {end for i}

for i:=1 to maxBoots do
begin
  TmpT := T[i];
  T[0] := TmpT;
  j := i-1;
    while  TmpT < T[j] do
      begin
        T[j+1] := T[j];
        j := j-1;
      end; { end while }
  T[j+1] := TmpT;
end;

//for i:=1 to maxBoots do
  //Writeln('T ',i,' ',T[i]:4:4);

d := (alfa/2)*maxBoots;
index := SingToInt(d);
t0 := T[index];
//writeln('  ',t0:4:4);

d := (1-(alfa/2))*maxBoots;
index := SingToInt(d);
```

```
t1 := T[index];
//writeln(d:4:4,' ',index,' ',t1:4:4); readln;
//writeln(' ',t1:4:4);
except
end;

end; {end GenerateBoots}

procedure Bootstrap(dat : data) ;
var   i : integer;
      y : data;
      ybar,S,A,B,qvalue,tmp : single;
begin

for i:=1 to n do
   y[i] := ln(dat[i]);
ybar := Mean(y);
S := Variant(y);
A := ybar+(S/2);
B := SQRT( S*(1+S/2) );
//writeln('t0/t1 ',t0:2:2,' ',t1:2:2);
//writeln('A/B ',A:2:2,' ',B:2:2);
BootstrapBound[iround,1] := exp(  A-(t1/SQRT(n))*B );
BootstrapBound[iround,2] := exp(  A-(t0/SQRT(n))*B );
//writeln(A-(t0/SQRT(n))*B:4:4);
//writeln(BootstrapBound[iround,1]:4:4,' ',BootstrapBound[iround,2]:4:4,'
',BootstrapBound[iround,2]-BootstrapBound[iround,1]:4:4);
```

```
//readln;
end; { end Naive }

function Chisquare(al : single) : single; { find percentile of chi square}
const maxBoots=10000;
type Tval = array[0..maxBoots] of single;
var y : data;
   T : Tval;
   i,j,index : integer;
   N01,SumN01,Chivalue,TmpT,d : single;
begin
for i:=1 to maxBoots do begin
 SumN01 := 0;
 for j:=1 to n-1 do  {degree of freedom n-1}
   SumN01 := SumN01 + SQR(Normal(0,1));
 T[i] := SumN01;
end; {end for i}
for i:=1 to maxBoots do
begin
 TmpT := T[i];
 T[0] := TmpT;
 j := i-1;
  while  TmpT < T[j] do
    begin
      T[j+1] := T[j];
      j := j-1;
    end; { end while }
```

```
  T[j+1] := TmpT;
end;
d := (al)*maxBoots;
index := SingToInt(d);
Chivalue := T[index];
Chisquare := Chivalue;
end; {end Chisquare}

function Tdist(al : single) : single ;
const maxBoots=10000;
type Tval = array[0..maxBoots] of single;
var y : data;
    T : Tval;
    i,j,index : integer;
    N01,Tdis,SumN01,TmpT,d,Chiv : single;
begin
for i:=1 to maxBoots do begin
  N01 := Normal(0,1);

  SumN01 := 0;
  for j:=1 to n-1 do  {degree of freedom n-1}
    SumN01 := SumN01 + SQR(Normal(0,1));
  Chiv := SumN01;

  T[i] := N01/SQRT(Chiv/(n-1));
  //Writeln('T ',i,' ',T[i]:4:4);
end; {end for i}
```

```
for i:=1 to maxBoots do
begin
  TmpT := T[i];
  T[0] := TmpT;
  j := i-1;
    while  TmpT < T[j] do
      begin
        T[j+1] := T[j];
        j := j-1;
      end; { end while }
  T[j+1] := TmpT;
end;

d := (1-alfa/2)*maxBoots;
index := SingToInt(d);
Tdist := T[index];

end; {end t Distribution}


function Zdist(al : single) : single ;
const maxBoots=10000;
type Tval = array[0..maxBoots] of single;
var y : data;
    T : Tval;
    i,j,index : integer;
```

```
   N01,TmpT,d : single;
begin
for i:=1 to maxBoots do begin
 N01 := Normal(0,1);
 T[i] := N01;
end; {end for i}

for i:=1 to maxBoots do
begin
 TmpT := T[i];
 T[0] := TmpT;
 j := i-1;
   while  TmpT < T[j] do
     begin
       T[j+1] := T[j];
       j := j-1;
     end; { end while }
 T[j+1] := TmpT;
end;

d := (1-alfa/2)*maxBoots;
index := SingToInt(d);
Zdist := T[index];
end; {end Normal Distribution}
```

```
procedure ACE();begin;end;

begin  {################## MAIN PROGRAM ##################}
seed:=4528392;
variance:=0.01;
n:=5;alfa:=0.1;
Cvalue:=0.8878;
Zalfa:=1.645;tvalue:=2.1318;chi:=0.7107;
Theta := exp(mu+variance/2);

//chi:=Chisquare(alfa);
//tvalue:=Tdist(alfa);
//Zalfa:= Zdist(alfa);
//writeln('Chi value ',chi:2:4) ;
//writeln('T   value ',tvalue:2:4) ;
//writeln('Z   value ',Zalfa:2:4) ;
//readln;

for iround := 1 to round do begin

  for ir:=1 to n do
    x[ir] :=  exp(Normal(mu,variance));

  {-- View data & summary --}
  //for ir:=1 to n do
    // writeln(ir,'  ',x[ir]:4:4);
  //writeln(exp(mu+variance/2):4:4,'  ',Mean(x):4:4);
```

```
  //writeln(Variant(x):4:4);readln;

  {-- Calculate CI --}
  Writeln('Progess ',(iround/round)*100:3:2,' % ');
{Method 1}
  Cox(x);
{Method 2}
  Conservative(x);
{Method 3}
  GenerateBoots(x);
  Bootstrap(x);

end; {end iround}

Writeln('-------------------------------------');
Write('1.Cox P hat : ',CalP(coxBound):4:4,' / ');
if CalP(coxBound)> Cvalue then
  Writeln('  AVG-Diff : ',MeanDiff(coxBound):4:4)
else
  Writeln('  AVG-Diff : ','N/A');

Write('2.Conservative P hat : ',CalP(ConservativeBound):4:4,' / ');
if CalP(ConservativeBound)> Cvalue then
  Writeln('  AVG-Diff : ',MeanDiff(ConservativeBound):4:4)
else
  Writeln('  AVG-Diff : ','N/A');
```

```
Write('3.Bootstrap P hat : ',CalP(BootstrapBound):4:4,' / ');
if CalP(BootstrapBound)> Cvalue then
  Writeln('  AVG-Diff : ',MeanDiff(BootstrapBound):4:4)
else
  Writeln('  AVG-Diff : ','N/A');



Readln;

end.
```

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวจตุพร กัลยากิตติกุล เกิดวันศุกร์ที่ 16 กุมภาพันธ์ พ.ศ.2522 ที่จังหวัดขอนแก่น สำเร็จมัธยมศึกษาตอนปลายจากโรงเรียนขอนแก่นวิทยายน จังหวัดขอนแก่น สำเร็จการศึกษาปริญญาวิทยาศาสตรบัณฑิต (วท.บ.) สาขาสถิติ ภาควิชาสถิติ คณะวิทยาศาสตร์ มหาวิทยาลัยขอนแก่น ในปีการศึกษา 2543 และเข้าศึกษาต่อในหลักสูตรสถิติศาสตรมหาบัณฑิต (สต.ม.) สาขาวิชาสถิติ ภาควิชาสถิติ คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย ในปี พ.ศ.2545